UNIT I

Open Source Programming

writing code that other people can freely use and modify. - GitHub, Bitbucket, SourceForge

What is Open Source software

A software whose source code is available to the public to view, use, modify, and distribute to any one for any purpose under a permissive license.

It may or may not be free.

You may or may not be able to modify the code, sell the code depending on the license agreement.'

The most common type of open source license is GNU(GPL). With this one, you can do whatever you want with the software.

- The source code must be open and available under the same terms as which you got the code
- Copyleft says that anyone who redistributes the software, with or without changes, must pass along the freedom to further copy and change it. Copyleft guarantees that every user has freedom.

Advantages of Open Source Software

Cost: Open source software is typically free to use, modify and distribute.

Customization: The source code of open source software is available to everyone, allowing users to modify and customize it to suit their needs.

Community support: Open source software often has a large community of developers and users who contribute to its development and provide support.

Transparency: The source code of open source software is open for everyone to see, making it easier to identify and fix bugs and vulnerabilities.

Flexibility: Open source software can be used on a wide range of platforms and devices.

Disadvantages of open Source Software

Support: While open source software does have a large community of developers and users, it may not always have the same level of professional support as commercial software.

Compatibility: Open source software may not always be compatible with other software applications and hardware devices.

Security: Because the source code of open source software is available to everyone, it may be easier for malicious actors to identify and exploit vulnerabilities.

Complexity: Open source software can be more complex and difficult to use than commercial software, especially for non-technical users.

Documentation: Open source software may not always have the same level of documentation and user guides as commercial software.

Examples:

- Operating systems – Android, Ubuntu, Linux
- Internet browsers – Mozilla Firefox, Chromium
- Integrated Development Environment (IDEs) – Vs code (Visual Studio Code), Android Studio, PyCharm, Xcode

- Relational database - MySQL
- office suite - LibreOffice, Open Office
- Apache web server, Apache HTTP Server
- GIMP, VLC Media Player
- High level languages - Python, Perl,
- Web development - PHP

2. Free Software

"Free software" means software that respects users' freedom and community. Roughly, it means that the users have the freedom to run, copy, distribute, study, change, and improve the software. The term "free software" is sometimes misunderstood—it has nothing to do with price. It is about freedom.

Every free software is open source

Every open-source software is not free software.

Advantages of Free Software

Cost: Free software is typically free to use, modify, and distribute.

Freedom: Free software is often accompanied by a set of ethical principles that promote users' freedom to use, study, modify, and share the software.

Collaboration: Free software often encourages collaboration among developers and users, leading to faster development and better quality software.

Transparency: The source code of the software is available for public scrutiny.

Flexibility: Free software can be used on a wide range of platforms and devices.

Disadvantages of Free Software

Support: While free software does have a community of developers and users, it may not always have the same level of professional support as commercial software.

Compatibility: Free software may not always be compatible with other software applications and hardware devices.

Security: Because free software is available for everyone to use and modify, it may be easier for malicious actors to identify and exploit vulnerabilities.

Complexity: Free software can be more complex and difficult to use than commercial software, especially for non-technical users.

Documentation: Free software may not always have the same level of documentation and user guides as commercial software.

Difference between Free Software and Open Source Software - No license issue

3. 2. Closed Source Software

Source code is closed means the public is not given access to the source code

3. **Proprietary Software:**

Proprietary software is computer software where the source codes are publicly not available. Only the company that has created them can modify it. Here the software is developed and tested by the individual or organization by which it is owned not by the public. This software is managed by a closed team of individuals or groups that developed it. We have to pay to get this software and its commercial support is available for maintenance. The company gives a valid and authenticated license to the users to use this software. But this license puts some restrictions on users also like.

- Number of installations of this software into computers
- Restrictions on sharing of software illegally
- Time period up to which software will operate
- Number of features allowed to use

Some **examples of Proprietary software** include Windows, macOS, Internet Explorer, Google Earth, Microsoft Office, etc.

Difference between Open-source Software and Proprietary Software:

In open-source software the source code is public.	In proprietary software, the source code is protected.
Users do not need to have any authenticated license to use this software.	Users need to have a valid and authenticated license to use this software.
Open-source software is modified by an open-source community of developers.	Proprietary software is modified by a closed team of individuals or groups that developed it.
The price of open source software is very less.	The price of closed source software is high.
Open source software fails fast and fix faster.	Closed source software has no room for failure.
In open source software no one is responsible for the software.	In closed source software the vendor is responsible if anything happened to software.

Examples Microsoft Windows, macOS, Internet Explorer, Google Earth, Microsoft Office, Adobe Flash Player, Adobe reader, Skype, etc.

Need of OSS

1. To reduce dependency on closed source vendors.

For upgraded products with needed features, we should depend on the closed source vendors.

- forced to use supported version of the product

2. To reduce budget in **software maintenance**

Cost will be lower than commercial software.

3. To use more tools

Without asking the owner a variety of development and testing tools, project management tools, network monitoring, security, content management can be used.

4. To try before buying

Before buying a software, **g**et familiar with the OSS. If it can do the required job no need to lock in to another vendor.

5. **To get 24/7 support**

If software with strong community backing is used, we can get great support from community members throughout the world most likely within minutes.

6. To use software with needed features

Closed source software consume huge amount of memory and CPU power for the features never used. In open source software, most features depends on the demand of community.

7. **To fix bugs fastly**.

Many bugs are fixed by the community before they are even reported by the users.

Process

Process Types in Linux

Linux has two types of process

- Real-time Process - Process which have strict deadlines
- Normal Process / Conventional

Real-time processes are required to 'obey' response time constraints without any regard to the system's load. In different words, real-time processes are **urgent and cannot be delayed** no matter the circumstances.

- Strict response time constraints deadlines have to be met
- Cannot be delayed
- Should never be blocked by a low priority task

Eg. Robotics, data acquisition

Normal processes don't have strict response time constraints and they can suffer from delays in case the system is 'busy'. An example of a conventional process can be the browser process . Two types

1. Interactive - Constantly interact with users, Spend a lot of time waiting for key presses and mouse operations - IO bound process.
2. Batch - Do not require any user interaction, often run in the background.

Each process type has a different scheduling algorithm

Process Scheduler

The process scheduler is responsible for

1. Choosing a process to be executed
2. Deciding how long the chosen process is to be executed .

It is the basic part of a multitasking operating system. The Linux scheduler supports different scheduling algorithms to schedule different types of processes. These are known as **scheduler classes**.

Processes assigned to a priority classes

• Each class has its own ready queue

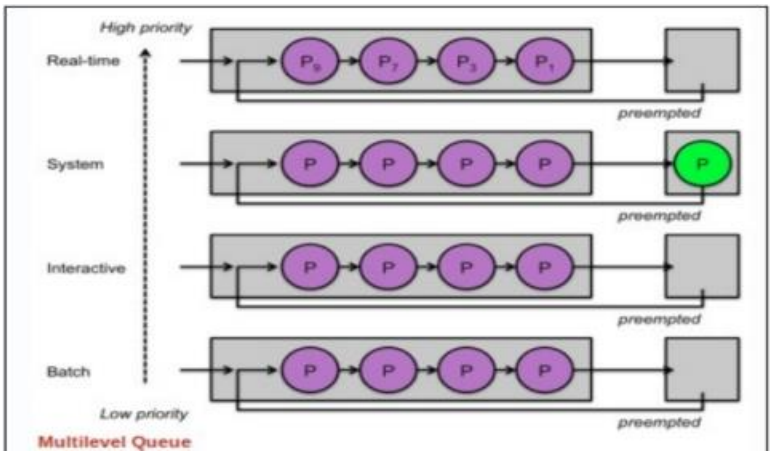
• Scheduler picks the highest priority queue (class) which has at least one ready process

• Selection of a process within the class could have its own policy

– Typically round robin (but can be changed)

– High priority classes can implement first come first serve in order to ensure quick response time for critical tasks

Each scheduler class has a different priority, the scheduler iterates over each scheduler class in the order of priority. The highest priority scheduler class with a runnable process wins, and the winning scheduler class selects which process to run next



The scheduler goals are to

- Maximize throughput - (amount of tasks completed per time unit) . Complete as many processes as possible per unit time
- Minimize wait time (amount of time passed since the process was ready until it started to execute) - • Process should not wait long in the ready queue
- Minimize response time (amount of time passed since the process was ready until it finished executing) - CPU should respond immediately

- Maximize fairness (distributing resources fairly for each process) - Give each process a fair share of CPU
- Maximize CPU utilization • CPU should not be idle

History of Linux scheduler

- Initially, the **LINUX kernel used the RR or Round-Robin approach to schedule the processes**. In the Round-Robin approach, a **circular queue** was maintained that held the processes. Linux 1.2
- After the introduction of **scheduling classes** in LINUX Kernel 2.2, the processes were divided into three process classes:
    - Real-time process
    - Non-preemptive process(a process does not leave the CPU until the process makes its system call)
    - Normal process.
- After the introduction of the O(N) scheduler in LINUX Kernel 2.4, a **queue** is used to store processes. At the time of scheduling, the best process in the queue is selected according to the the **priority** of the processes.
- In the LINUX Kernel 2.6, the complexity of the scheduler got reduced from O(N) to O(1).
- After O(1) scheduler, in Linux 2.6.23, CFS, or Completely Fair Scheduler was introduced.

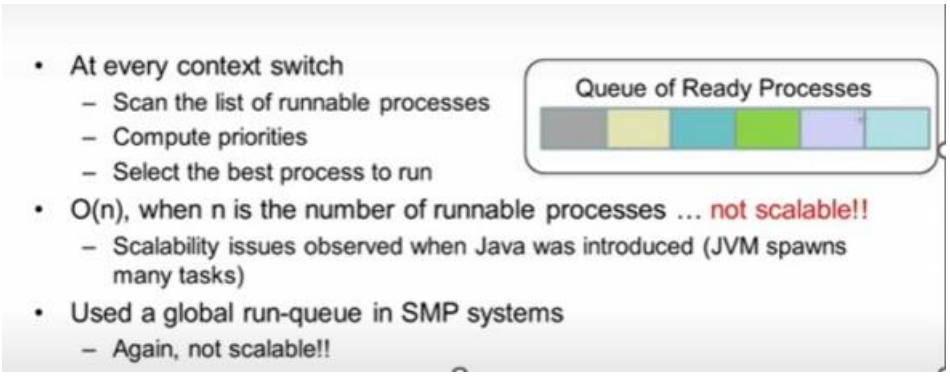
O(N) Scheduler

The LINUX Kernel used the O(N) scheduler between version 2.4 to 2.6.

- N is the number of runnable processes in the system.
- O(N) scheduler divides the **processor's time** into a unit called epochs.
- Each task is allowed to use a max 1 epoch.
- Within each epoch, every task can execute up to its time slice.
- If the task is not completed in the specified epoch, then the scheduler adds half of the remaining time in the next epoch.
- This allow it to execute longer in the next epoch

Advantage:

O(n) scheduler was better than the earlier used circular queue scheduler because O(N) scheduler could schedule N-processes at a time.



JVM - Java Virtual Machine

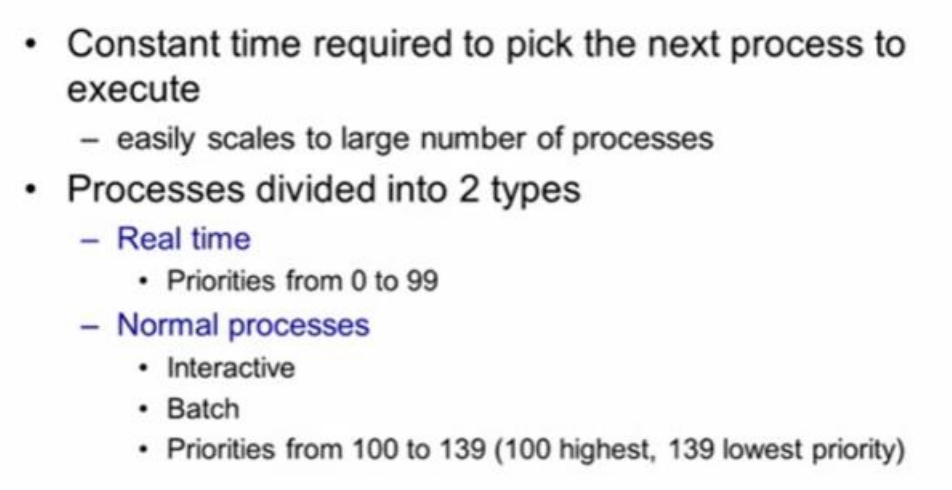
SMP - Symmetric Multi- Processing - refers to the computer architecture where multiple identical processors are interconnected to a single shared main memory, with full accessibility to all the I/O devices, unlike asymmetric MP. In other words, all the processors have common shared(common) memory and same data path or I/O bus

Disadvantage:

If the number of processes is more, the scheduler may use a notable amount of the processor time itself. Picking the next task to run requires iteration through all currently planned tasks, so the scheduler runs in O(n) time, where n is the number of the planned processes.

O(1) Scheduler

O(1) Scheduler was introduced in LINUX Kernel 2.6. O(1) scheduler is also called as **constant time scheduler**. As the name suggests, it can schedule the processes within a constant amount of time, regardless of the number of processes running on the system.



+

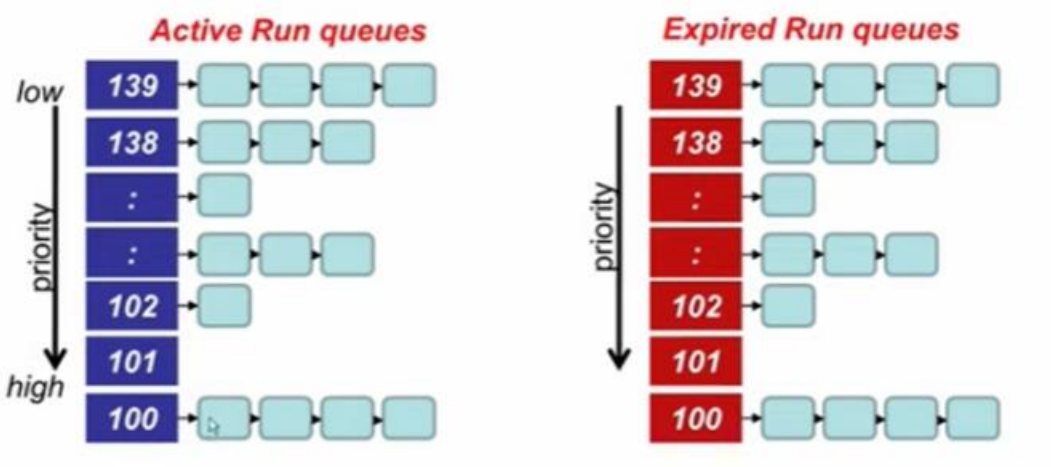
Scheduling normal process

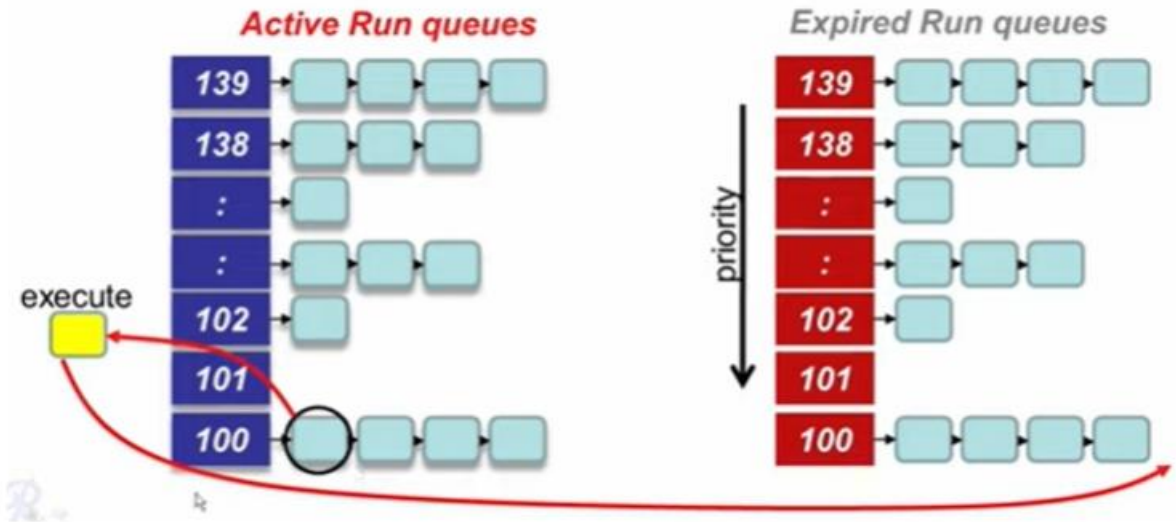
- O(1) scheduler uses two ready queues active run queue and expired run queue in each CPU.
- Each queue has 40 priority classes(100-139). 100 has highest priority, 139 has lowest priority.
- The active processes are placed in a queue that stores the priority value of each process. This queue is termed as active run queue, or run queue.
- The other queue is an array of expired processes called expired queue. Initially it will be empty. When the allotted time of a process expires, it is placed into the expired queue.

The scheduler gives a priority to interactive tasks and lowers the priorities of the non-interactive tasks.

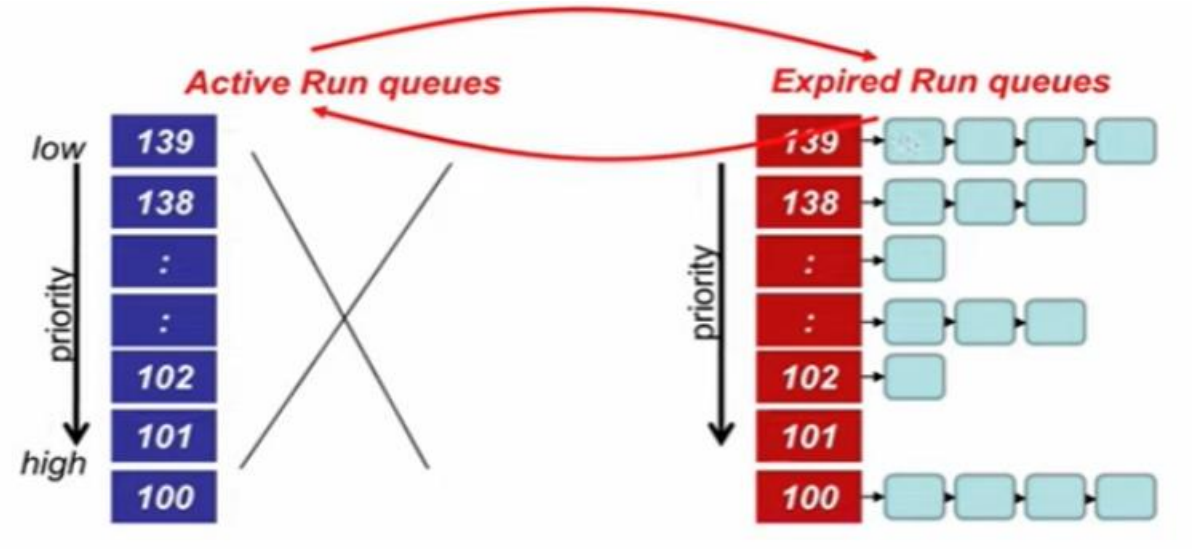
There are 2 steps in the scheduling

1. Find the lowest numbered queue with at least 1 process
2. Choose the first process from that queue.





Once active run queues are complete, make expired run queues active and vice versa. This prevents starvation. Entire process gets turn to execute



Note:

Preemptive priority based algm

0 - 99 real time range

100 - 140 nice value range

- step 2 is obviously constant time
- Is step 1 contant time?
  - Store bitmap of run queues with non-zero entries
  - Use special instruction '*find-first-bit-set*'
    - *bsfl* on intel

More on Priorities

- 0 to 99 meant for real time processes
- 100 is the highest priority for a normal process
- 139 is the lowest priority
- Static Priorities
  - 120 is the base priority (default)
  - **nice** : command line to change default priority of a process

    $nice –n N ./a.out
  - N is a value from +19 to -20;
    - most selfish '-20' ; (I want to go first)
    - most generous '+19' ; (I will go last)

Based on a heuristic

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a 'bonus', which changes based on a heuristic

dynamic priority = MAX(100, MIN(static priority – bonus + 5), 139))

Has a value between 0 and 10

If bonus < 5, implies less interaction with the user
thus more of a CPU bound process.
The dynamic priority is therefore decreased (toward 139)

If bonus > 5, implies more interaction with the user
thus more of an interactive process.
The dynamic priority is increased (toward 100).

Dynamic Priority (setting the bonus)

- To distinguish between batch and interactive processes
- Based on average sleep time
  - An I/O bound process will sleep more therefore should get a higher priority
  - A CPU bound process will sleep less, therefore should get lower priority

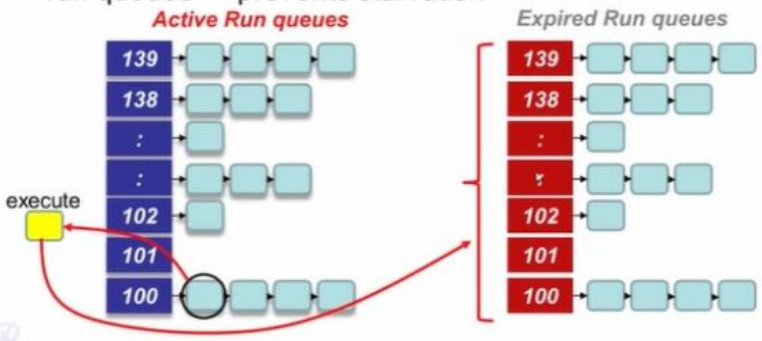
dynamic priority = MAX(100, MIN(static priority – bonus + 5), 139))

heuristic

Average sleep time	Bonus
Greater than or equal to 0 but smaller than 100 ms	0
Greater than or equal to 100 ms but smaller than 200 ms	1
Greater than or equal to 200 ms but smaller than 300 ms	2
Greater than or equal to 300 ms but smaller than 400 ms	3
Greater than or equal to 400 ms but smaller than 500 ms	4
Greater than or equal to 500 ms but smaller than 600 ms	5
Greater than or equal to 600 ms but smaller than 700 ms	6
Greater than or equal to 700 ms but smaller than 800 ms	7
Greater than or equal to 800 ms but smaller than 900 ms	8
Greater than or equal to 900 ms but smaller than 1000 ms	9
1 second	10

Dynamic Priority and Run Queues

- Dynamic priority used to determine which run queue to put the task
- No matter how 'nice' you are, you still need to wait on run queues --- prevents starvation



Setting the Timeslice

- Interactive processes have high priorities.
  - But likely to not complete their timeslice
  - Give it the largest timeslice to ensure that it completes its burst without being preempted. More heuristics

```
If priority < 120
      time slice = (140 – priority) * 20    milliseconds
else
      time slice = (140 – priority) * 5   milliseconds
```

Priority:	Static Pri	Niceness	Quantum
Highest	100	-20	800 ms
High	110	-10	600 ms
Normal	120	0	100 ms
Low	130	10	50 ms
Lowest	139	19	5 ms

Summarizing the O(1) Scheduler

- Multi level feed back queues with 40 priority classes
- Base priority set to 120 by default; modifiable by users using nice.
- Dynamic priority set by heuristics based on process' sleep time
- Time slice interval for each process is set based on the dynamic priority

---

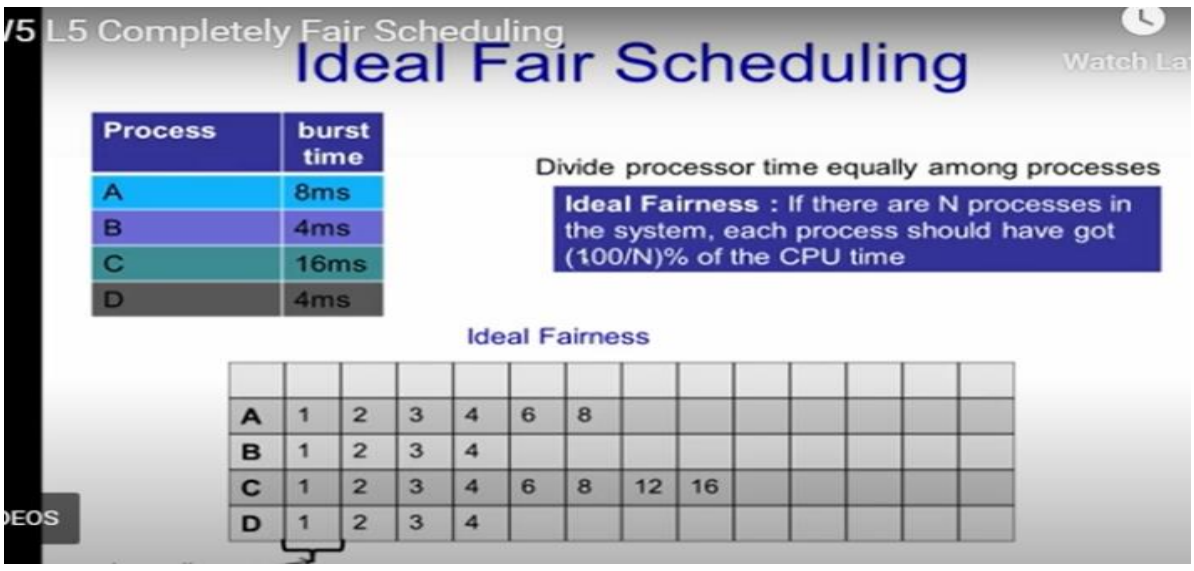
CFS Scheduler

CFS stands for Completely Fair Scheduler.

Introduced in 2.6.23 by Ingo Molnar

The main aim of CFS is to maximize the overall CPU utilization and performance. CFS uses a very simple algorithm for process scheduling.

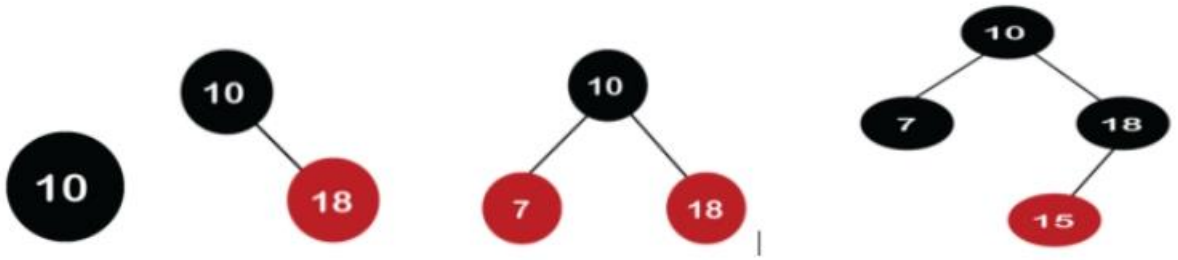
- **CFS uses a red-black tree instead of a queue for the scheduling**.
- All the processes are inserted into Red-Black trees and whenever a new process arrives, it is inserted into the tree.



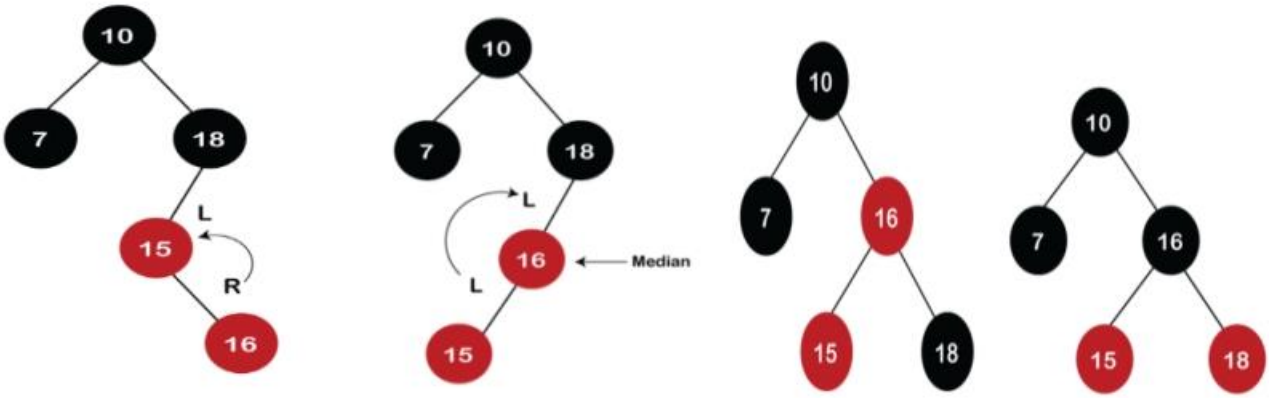
CFS is similar as ideal-based scheduling instead it priorities each process according to their virtual runnable time.

Iinsertion in the Red-Black tree.

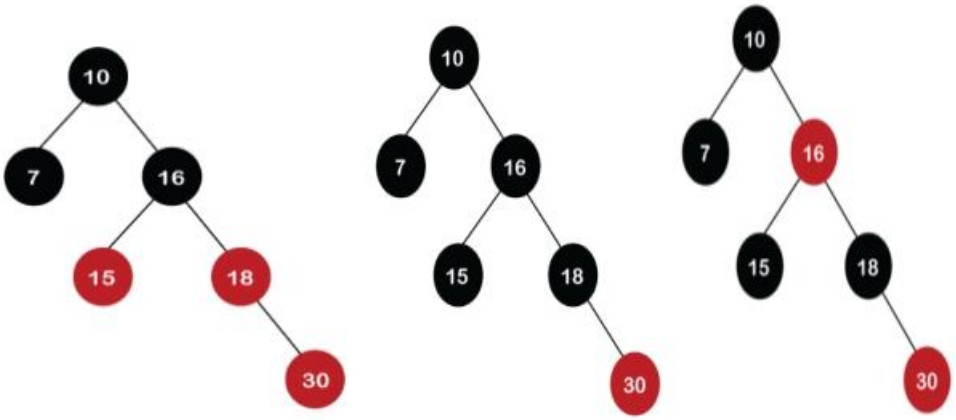
10, 18, 7, 15, 16, 30, 25, 40, 60



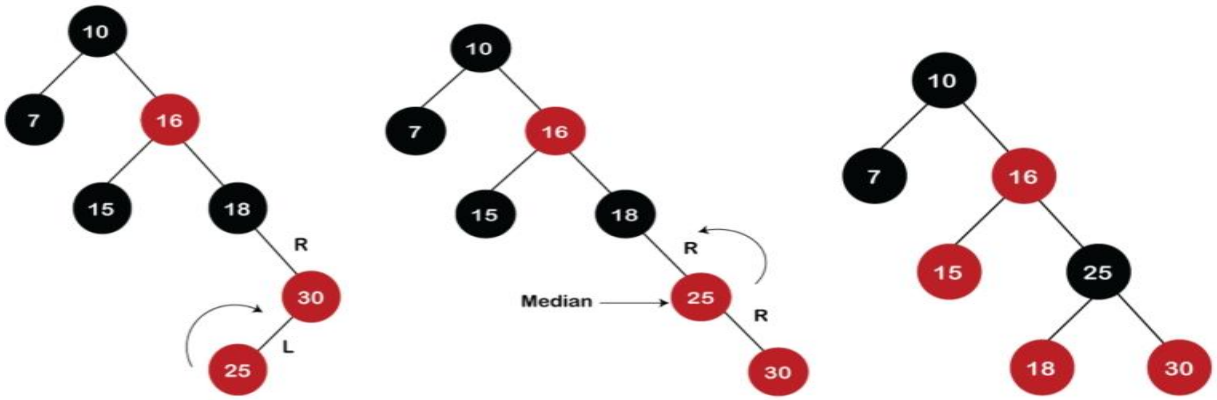
16



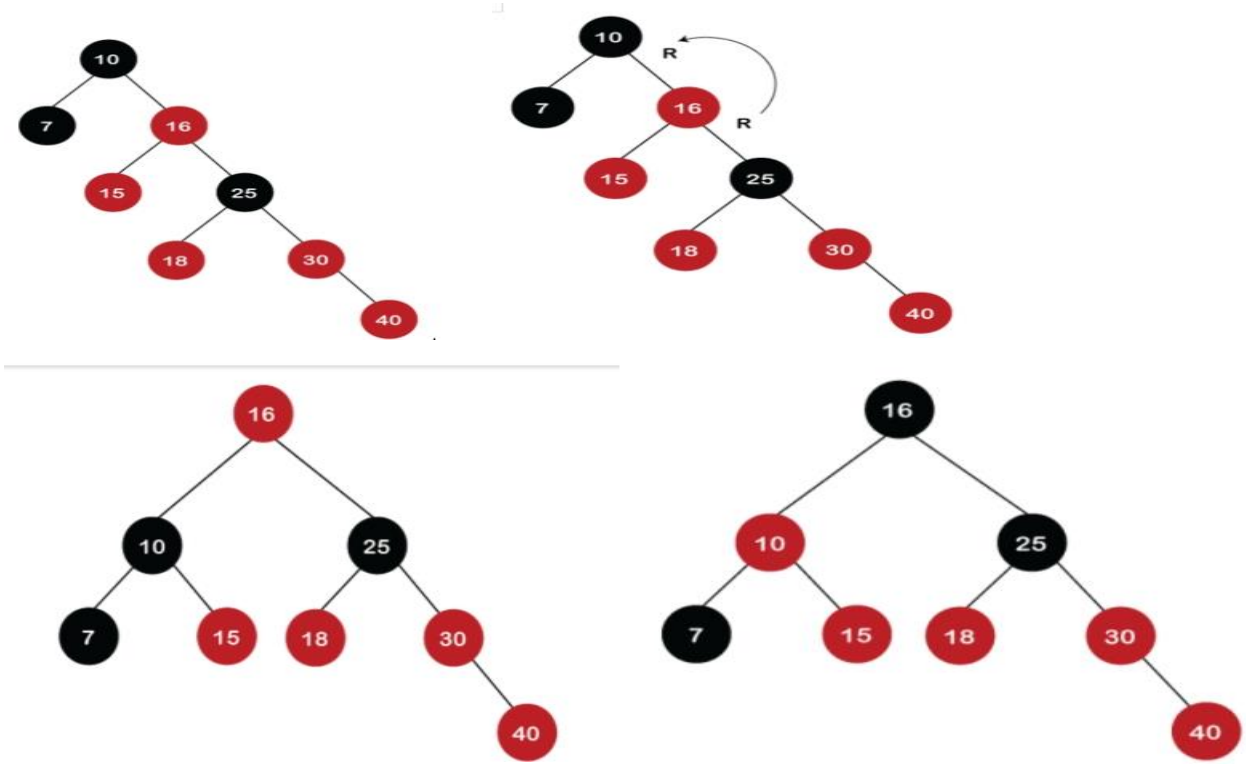
30



25



40



Personality

In the context of Linux, "personality" refers to a feature of the kernel that allows it to emulate the behavior of other operating systems. This is useful for compatibility purposes, enabling Linux to run software designed for other Unix-like systems.

Two ways

1. System Call - Personality
2. Command line utiliy - setarch

Common Personalities:

- PER_LINUX (0x00000000): The default Linux personality.
- PER_LINUX32 (0x00000008) : A 32-bit Linux personality on a 64-bit kernel.
- PER_SVR4 (0x00000001): System V Release 4 personality.

- PER_BSD: BSD personality.
- PER_SOLARIS: Solaris personality.

Example

```c
#include <stdio.h>
#include <sys/personality.h>

int main() {
    unsigned long persona = personality(0xFFFFFFFF); // Get current personality
    printf("Current personality: %lu\n", persona);

    personality(PER_BSD); // Set BSD personality
    persona = personality(0xFFFFFFFF); // Get new personality
    printf("New personality: %lu\n", persona);

    return 0;
}
```

This code snippet demonstrates how to get and set the personality of a process in C.

Include Headers:

- #include <stdio.h>: Includes the standard I/O library for input and output functions.
- #include <sys/personality.h>: Includes the header file for the personality system call.

Main Function:

- unsigned long persona = personality(0xFFFFFFFF);:
  - This line calls the personality function with the argument 0xFFFFFFFF, which retrieves the current personality of the process.
- printf("Current personality: %lu\n", persona);:
  - This line prints the current personality of the process.

PER_MASK (0xFFFFFFFF): Mask to retrieve the current personality.

Set New Personality:

- personality(PER_BSD);:
  - This line sets the personality of the process to BSD.
- persona = personality(0xFFFFFFFF);:
  - This line retrieves the new personality of the process after setting it to BSD.
- printf("New personality: %lu\n", persona);:
  - This line prints the new personality of the process.

Return Statement:

- return 0;:
  - This line indicates that the program finished successfully.

Running the Code

To compile and run this code, you can use the following commands in a Linux terminal:

gcc -o personality_example personality_example.c

./personality_example

This will compile the C program and produce an executable named personality_example. Running this executable will show the current and new personality of the process.

Command-Line Utility:

setarch can be used to invoke a program with a different personality. For example:

setarch i386 ./myprogram

The command setarch i386 ./myprogram is used to run a program (./myprogram) in a different architecture or personality environment than the one currently in use.

Breakdown of the Command:

setarch: setarch is a command used to set the architecture or personality of a process.

It allows to specify a particular architecture for running a program, which can be useful for testing or compatibility purposes.

i386: i386 specifies the architecture to be used.. This is to run a 32-bit application on a 64-bit system.

./myprogram: This specifies the program to run

Cloning

Creating copies.

Copying an entire disk, drives, files, directories, partitions, process , repositories.

1. Disk Cloning

Disk cloning involves copying the entire content of one disk to another. Tools like dd, Clonezilla, and Partimage are commonly used for this purpose.

Using dd

The dd command is a versatile utility for copying and converting data. To clone a disk:

```
sudo dd if=/dev/source_disk of=/dev/target_disk bs=64K conv=noerror,sync
```

- if (input file): Source disk (e.g., /dev/sda)
- of (output file): Target disk (e.g., /dev/sdb)
- bs: Block size (e.g., 64K)
- conv: Options to handle errors and ensure synchronization

sudo command allows a permitted user to execute a command as the superuser or another user, as specified by the security policy.

This provides a controlled way to grant administrative privileges to users without giving them full root access.

2. Partition Cloning

Partition cloning is similar to disk cloning but focuses on individual partitions. Tools like parted, gparted, or dd can be used for partition cloning.

Using dd

```
sudo dd if=/dev/source_partition of=/dev/target_partition bs=64K conv=noerror,sync
```

3. File System Cloning

Cloning a file system involves copying all files from one file system to another while preserving file attributes, permissions, and links. Tools like rsync are ideal for this task.

Using rsync

```
sudo rsync -aAXv /source_directory/ /target_directory/
```

- `-a` stands for archive mode, which preserves permissions, timestamps, symbolic links, and other attributes.
- `-A` preserves ACLs (Access Control Lists).
- `-X` preserves extended attributes.
- `-v` enables verbose output.

Process Cloning

In the context of processes, cloning refers to creating a new process that is a copy of an existing process. This is often achieved using the fork system call in programming.

Example in C

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("This is the child process\n");
    } else {
        // Parent process
        printf("This is the parent process\n");
    }

    return 0;
}
```

LINUX Signals

A feature of LINUX programming is the idea of sending and receiving signals. A signal is a kind of (usually software) interrupt, used to announce asynchronous events to a process.

There is a limited list of possible signals; we do not invent our own. (There might be 64 signals, for instance.) The name of a LINUX signal begins with "SIG". Although signals are numbered, we normally refer to them by their names. For example:

- SIGINT is a signal generated when a user presses Control-C. This will terminate the program from the terminal.
- SIGALRM is generated when the timer set by the alarm function goes off.
- SIGABRT is generated when a process executes the abort function.
- SIGSTOP tells LINUX to pause a process to be resumed later.
- SIGCONT tells LINUX to resume the processed paused earlier.
- SIGSEGV is sent to a process when it has a segmentation fault.
- SIGKILL is sent to a process to cause it to terminate at once.

What happens when a signal occurs?

When the signal occurs, the process has to handle it. There are three cases:

- Ignore it. Many signals can be and are ignored, but not all. Hardware exceptions such as "divide by 0" (with integers) cannot be ignored successfully and some signals such as SIGKILL cannot be ignored at all.
- Catch and handle the exception. The process has a function to be executed if and when the exception occurs. The function may terminate the program gracefully or it may handle it without terminating the program.
- Let the default action apply. Every signal has a default action. The default may be:
    - ignore
    - terminate
    - terminate and dump core
    - stop or pause the program
    - resume a program paused earlier

Each signal has a current "disposition" which indicates what action will be the default; an additional option is to have a programmer-defined function to serve as the signal handler.

An example of the use of signals is the use of the waitpid() function. It puts the calling process in a wait state (action = STOP) until the child process indicated has a change of status, which will be reported by a SIGCHILD signal (action = resume). By default, waitpid() expects the child to terminate, but there are ways to change this so other changes of status can be handled.

Example C Program to Catch a Signal

Most of the Linux users use the key combination Ctr+C to terminate processes in Linux.

Whenever Ctrl+C is pressed, a signal SIGINT is sent to the process. The default action of this signal is to terminate the process. But this signal can also be handled. The following code demonstrates this:

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
 void sig_handler(int signo)
{
  if (signo == SIGINT)
    printf("received SIGINT\n");
}

int main(void)
{
  if (signal(SIGINT, sig_handler) == SIG_ERR)
  printf("\ncan't catch SIGINT\n");
  // A long long wait so that we can easily issue a signal to this  process
  while(1)
    sleep(1);
  return 0;
}
```

UNIT II

PHP Variables

In PHP, a variable is declared using a $ sign followed by the variable name.

some important points to know about variables:

- As PHP is a loosely typed language, we need not declare the data types of the variables.
- Assignment Operator (=) is used to assign the value to a variable.

Syntax of declaring a variable in PHP is given below:

```
$variablename = value;
```

Rules for declaring PHP variable:

- A variable must start with a dollar ($) sign, followed by the variable name.
- It can only contain alpha-numeric character and underscore (A-z, 0-9, _).
- A variable name must start with a letter or underscore (_) character.
- A PHP variable name cannot contain spaces.

Variable name cannot start with a number or special symbols.

- PHP variables are case-sensitive, so $name and $NAME both are treated as different variable

---

PHP Data Types

PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types (predefined)
2. Compound Types (user-defined)

3. Special Types

PHP Data Types: Scalar Types

It holds only single value. There are 4 scalar data types in PHP.

1. boolean 2. integer 3.float 4. string

PHP Data Types: Compound Types

It can hold multiple values. There are 2 compound data types in PHP.

1. array 2.object

PHP Data Types: Special Types

There are 2 special data types in PHP.

1. resource 2. NULL

Testing the Type of a Variable

To determine the type of a variable at any time by using PHP ' s gettype() function.

```
$test_var = 8.23;
```

echo gettype($test_var)

PHP ' s type testing functions:

Function	Description
`is_int( value )`	Returns true if *value* is an integer
`is_float( value )`	Returns true if *value* is a float
`is_string( value )`	Returns true if *value* is a string
`is_bool( value )`	Returns true if *value* is a Boolean
`is_array( value )`	Returns true if *value* is an array
`is_object( value )`	Returns true if *value* is an object
`is_resource( value )`	Returns true if *value* is a resource
`is_null( value )`	Returns true if *value* is `null`

Changing a Variable ' s Data Type

use PHP ' s settype() function to change the type of a variable

$test_var = 8.23

settype($test_var, "integer");

echo $test_var // Displays "8"

settype($test_var, "boolean");

echo $test_var . // Displays "1"

By default, the data type of the variable accepted through readline() function is string. So for any other data type, we have to typecast it explicitly as described below.

```
<?php
$a = (int)readline('Enter an integer: ');
$b = (float)readline('Enter a floating  point number: ');
```

We can achieve the same thing without prompting the user also:

```
$a = readline();
```

Using Explode function

Accept multiple space separated inputs.

For this, we use another function explode() together with readline().

The first argument of explode() is the delimiter we want to use.

```
// Input 10 20
list($var1, $var2)  =  explode(' ', readline());
// Typecasting to integers
$var1 = (int)$var1;
$var2 = (int)$var2;
 // Printing the sum of var1 and var2.
echo "The sum of " . $var1 . " and " . $var2 . " is " . ($var1 + $var2);
 ?>
```

Output:

The sum of 10 and 20 is 30

We can also read an array through explode():

```
<?php
// For input  1 2 3 4 5 6
$arr = explode(' ', readline());
// For output
print_r($arr);
?>
```

Output:

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
```

[4] => 5

[5] => 6

)

Method 2:

Using fscanf() function works same as the fscanf() function in C. We can read 2 integers from Keyboard(STDIN) as below:

- This is different from the previous method

```
<?php

// Input 1 5

fscanf(STDIN, "%d %d", $a, $b);

// Output

// The sum of 1 and 5 is 6

echo "The sum of " . $a . " and " . $b . " is " . ($a + $b);

?>
```

Output:

The sum of 1 and 5 is 6

PHP Operators

PHP Operators can be categorized in following forms:

1. Arithmetic Operators
2. Assignment Operators
3. Bitwise Operators
4. Comparison Operators
5. Incrementing/Decrementing Operators
6. Logical Operators
7. String Operators
8. Array Operators
9. Type Operators
10. Execution Operators
11. Error Control Operators

We can also categorize operators on behalf of operands. They can be categorized in 3 forms:

- Unary Operators: works on single operands such as ++, -- etc.
- Binary Operators: works on two operands such as binary +, -, *, / etc.
- Ternary Operators: works on three operands such as "?:".

Arithmetic Operators

The PHP arithmetic operators are used to perform common arithmetic operations such as addition, subtraction, etc. with numeric values.

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

The exponentiation (**) operator has been introduced in PHP 5.6.

Assignment Operators

The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b
/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

Bitwise Operators

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

Operator	Name	Example	Explanation
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps

>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

Comparison Operators

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b
>	Greater than	$a > $b	Return TRUE if $a is greater than $b
<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal to $b Return 1 if $a is greater than $b

Incrementing/Decrementing Operators

The increment and decrement operators are used to increase and decrease the value of a variable.

Operator	Name	Example	Explanation
++	Increment	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	Decrement	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

Logical Operators

Operator	Name	Example	Explanation
and	And	$a and $b	Return TRUE if both $a and $b are true
or	or	$a or $b	Return TRUE if either $a or $b is true
xor	xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if both $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

String Operators

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b

Array Operators

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identical	$a === $b	Return TRUE if $a and $b have same value and same type in same order
!==	Non-Identical	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

Execution Operators

PHP has an execution operator backticks (``). PHP executes the content of backticks as a shell command. Execution operator and shell_exec() give the same result.

Operator	Name	Example	Explanation
``	backticks	echo `dir`;	Execute the shell command and return the result. Here, it will show the directories available in current folder.

Note: Note that backticks (``) are not single-quotes.

Error Control Operators

PHP has one error control operator, i.e., at (@) symbol. Whenever it is used with an expression, any error encountered by PHP parser will be suppressed and the expression will be ignored.

Operator	Name	Example	Explanation
@	at	@file ('non_existent_file')	Intentional file error

```
<?php
 // Statement 1
$result= $hello
 // Statement 2
$result= @$hello
?>
```

It will execute only statement 1 and display the error message

Undefined variable: hello

PHP Operators Precedence

Operators	Additional Information	Associativity
**	arithmetic	Right
++ -- ~ (int) (float) (string) (array) (object) (bool) @	increment/decrement and types	Right
instanceof	types	non-associative
!	logical (negation)	Right
* / %	arithmetic	Left
+ - .	arithmetic and string concatenation	Left
<< >>	bitwise (shift)	Left
< <= > >=	comparison	non-associative
== != === !== <>	comparison	non-associative
&	bitwise AND	Left
^	bitwise XOR	Left
|	bitwise OR	Left
&&	logical AND	Left
||	logical OR	Left
?:	ternary	Left
= += -= *= **= /= .= %= &=	assignment	Right

|= ^= <<= >>= =>		
and	logical	Left
xor	logical	Left
or	logical	Left
,	many uses (comma)	Left

PHP echo and print Statements

There are two basic ways to get the output in PHP:

- echo
- print

echo and print are language constructs, and they never behave like a function. Therefore, there is no requirement for parentheses. However, both the statements can be used with or without parentheses. We can use these statements to output variables or strings.

Difference between echo and print

echo

- echo is a statement, which is used to display the output.
- echo can be used with or without parentheses.
- echo does not return any value.
- We can pass multiple strings separated by comma (,) in echo.
- echo is faster than print statement.

print

- print is also a statement, used as an alternative to echo at many times to display the output.
- print can be used with or without parentheses.
- print always returns an integer value, which is 1.
- Using print, we cannot pass multiple arguments.
- print is slower than echo statement.

You can see the difference between echo and print statements with the help of the following programs.

Example (Check multiple arguments)

You can pass multiple arguments separated by a comma (,) in echo. It will not generate any syntax error.

```
1. <?php
2.     $fname = "Gunjan";
3.     $lname = "Garg";
4.     echo "My name is: ".$fname,$lname;
5. ?>
```

It will generate a syntax error because of multiple arguments in a print statement.

```
1. <?php
2.     $fname = "Gunjan";
3.     $lname = "Garg";
4.     print "My name is: ".$fname,$lname;
5. ?>
```

Example (Check Return Value)

echo statement does not return any value. It will generate an error if you try to display its return value.

```
1. <?php
2.     $lang = "PHP";
3.     $ret = echo $lang." is a web development language.";
4.     echo "</br>";
5.     echo "Value return by print statement: ".$ret;
6. ?>
```

Output:

As we already discussed that print returns a value, which is always 1.

```
1. <?php
2.     $lang = "PHP";
3.     $ret = print $lang." is a web development language.";
4.     print "</br>";
5. print "Value return by print statement: ".$ret;
6. ?>
```

Output:

PHP CONDITIONAL STATEMENTS

1. If Statement
2. Switch Statement

There are various ways to use if statement in PHP.

- if
- if-else
- if-else-if
- nested if

PHP If Statement

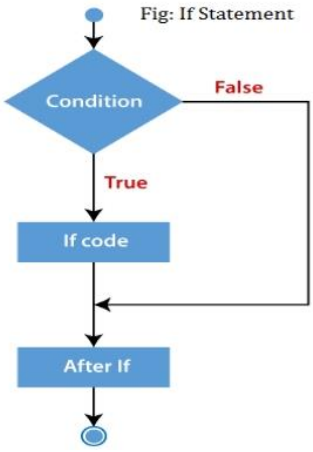
PHP if statement allows conditional execution of code. It is executed if condition is true.

If statement is used to executes the block of code exist inside the if statement only if the specified condition is true.

Syntax

```
1. if(condition){
2. //code to be executed
3. }
```

Flowchart



Example

```
1. <?php
2. $num=12;
3. If ($num<100) {
4. echo "$num is less than 100";
5. }
6. ?>
```

Output:

12 is less than 100

PHP If-else Statement

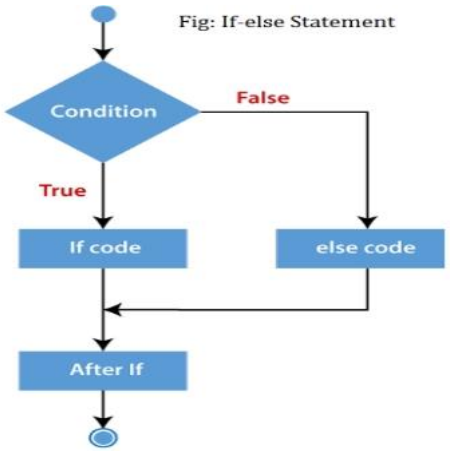
PHP if-else statement is executed whether condition is true or false.

If-else statement is slightly different from if statement. It executes one block of code if the specified condition is true and another block of code if the condition is false.

Syntax

```
1. If (condition) {
2. //code to be executed if true
3. } else {
4. //code to be executed if false
5. }
```

Flowchart



Example

```
1. <?php
2. $num=12;
3. if($num%2==0){
4. echo "$num is even number";
5. }else{
6. echo "$num is odd number";
7. }
8. ?>
```

Output:

12 is even number

PHP If-else-if Statement

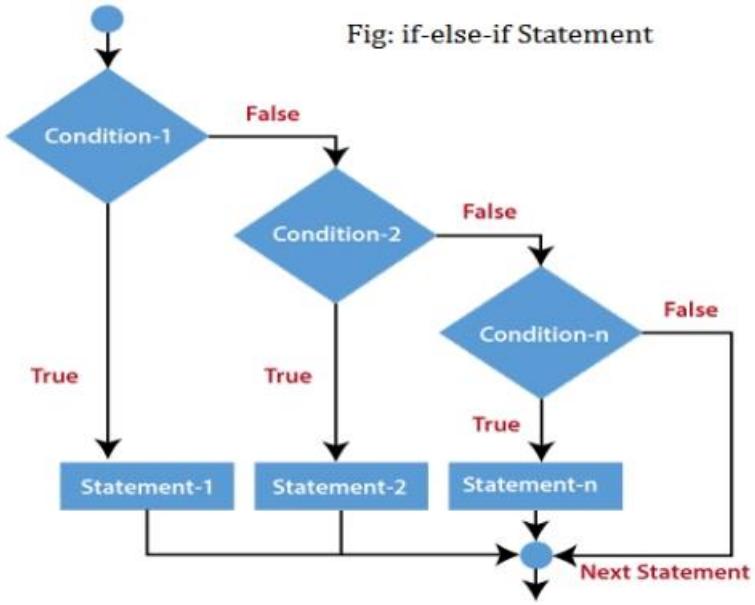
The PHP if-else-if is a special statement used to combine multiple if?.else statements. So, we can check multiple conditions using this statement.

Syntax

```
1. if (condition1){
2. //code to be executed if condition1 is true
3. } elseif (condition2) {
4. //code to be executed if condition2 is true
5. } elseif (condition3){
```

6. //code to be executed if condition3 is true
7. ....
8. } else {
9. //code to be executed if all given conditions are false
10. }

Flowchart



Example

```
<?php
if ($a > $b) {
echo "a is bigger than b";
} elseif ($a == $b) {
echo "a is equal to b";
} else {
echo "a is smaller than b";
}
```

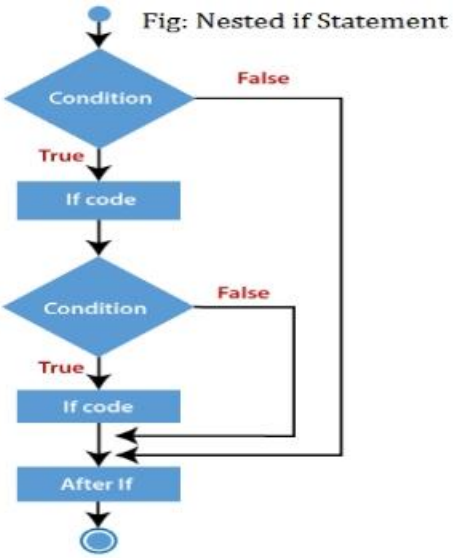
PHP nested if Statement

The nested if statement contains the if block inside another if block. The inner if statement executes only when specified condition in outer if statement is true.

Syntax

1. if (condition) {
2. //code to be executed if condition is true
3. if (condition) {
4. //code to be executed if condition is true
5. }
6. }

Flowchart



Example

```
1. <?php
2.  $age = 20;
3.   $nationality = "Indian";
4.   //applying conditions on nationality and age
5.   if ($nationality == "Indian")
6.   {
7.      if ($age >= 18) {
8.         echo "Eligible to give vote";
9.      }
10.     else {
11.        echo "Not eligible to give vote";
12.     }
13.  }
14. ?>
```

PHP Switch Example

```
1. <?php
2.          $a = 34; $b = 56; $c = 45;
3.   if ($a < $b) {
4.      if ($a < $c) {
5.         echo "$a is smaller than $b and $c";
6.      }
7.   }
```

8. ?>

Output:

34 is smaller than 56 and 45

PHP Switch

PHP switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

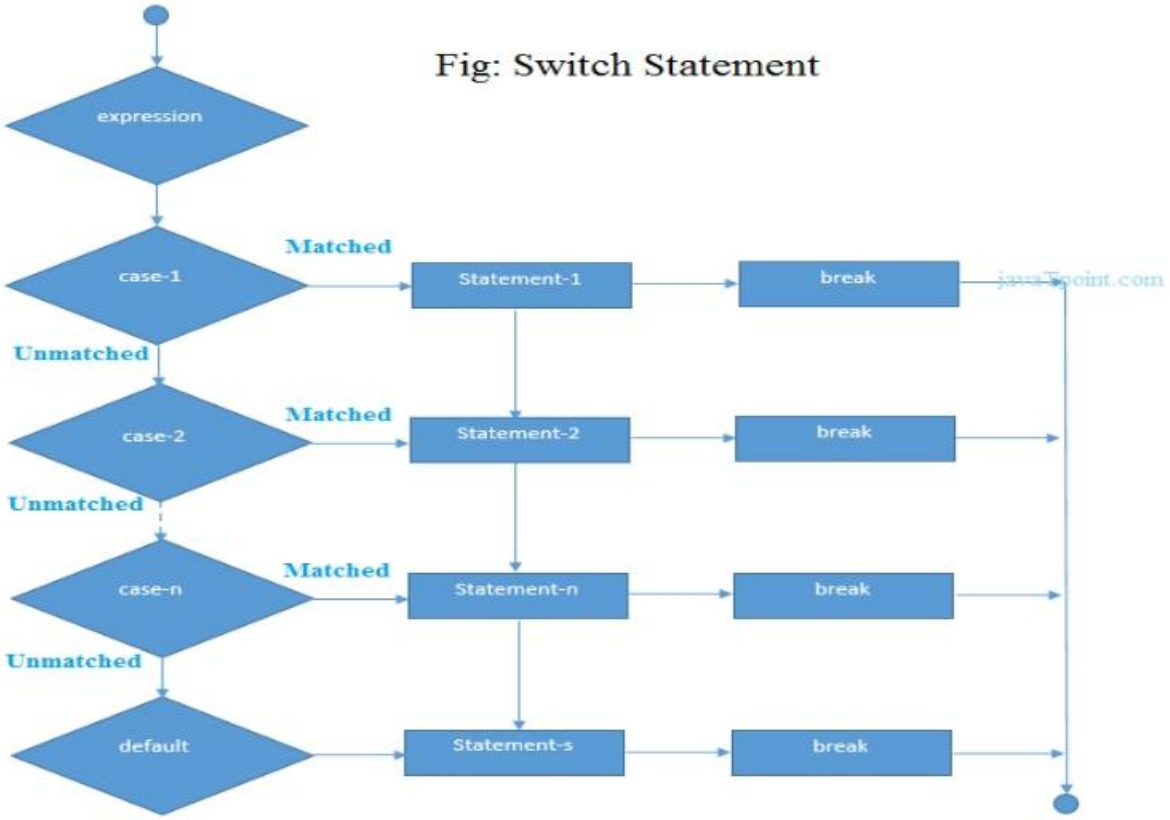
Syntax

1. switch(expression) {
2. case value1:
3. //code to be executed
4. break;
5. case value2:
6. //code to be executed
7. break;
8. .....
9. default:
10. code to be executed if all cases are not matched;
11. }

Important points to be noticed about switch case:

1. The default is an optional statement. Even it is not important, that default must always be the last statement.
2. There can be only one default in a switch statement. More than one default may lead to a Fatal error.
3. Each case can have a break statement, which is used to terminate the sequence of statement.
4. The break statement is optional to use in switch. If break is not used, all the statements will execute after finding matched case value.
5. PHP allows you to use number, character, string, as well as functions in switch expression.
6. Nesting of switch statements is allowed, but it makes the program more complex and less readable.
7. You can use semicolon (;) instead of colon (:). It will not generate any error.

PHP Switch Flowchart



PHP Switch Example

```
1. <?php
2. $num=20;
3. switch($num){
4. case 10:
5. echo("number is equals to 10");
6. break;
7. case 20:
8. echo("number is equal to 20");
9. break;
10. case 30:
11. echo("number is equal to 30");
12. break;
13. default:
14. echo("number is not equal to 10, 20 or 30");
15. }
16. ?>
```

PHP Looping Statements

1. For Statement
2. While Ststement

PHP For Loop

PHP for loop can be used to traverse set of code for the specified number of times.

It should be used if the number of iterations is known otherwise use while loop. This means for loop is used when you already know how many times you want to execute a block of code.

It allows users to put all the loop related statements in one place. See in the syntax given below:

Syntax

```
for(initialization; condition; increment/decrement)
{
//code to be executed
}
```

Parameters

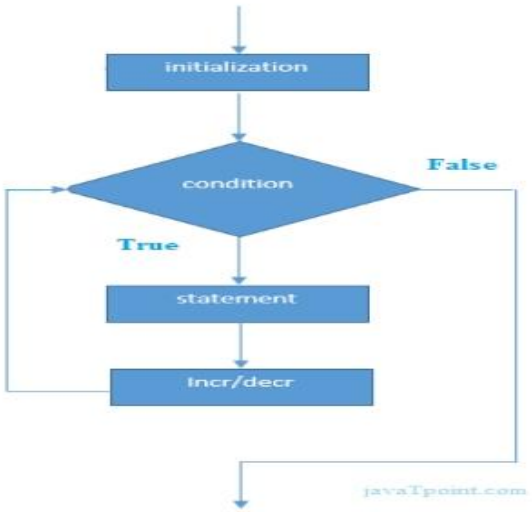
The php for loop is similar to the java/C/C++ for loop. The parameters of for loop have the following meanings:

initialization - Initialize the loop counter value. The initial value of the for loop is done only once. This parameter is optional.

condition - Evaluate each iteration value. The loop continuously executes until the condition is false. If TRUE, the loop execution continues, otherwise the execution of the loop ends.

Increment/decrement - It increments or decrements the value of the variable.

Flowchart



Example

```
?php
for($n=1;$n<=10;$n++){
echo "$n<br/>";
}
```

```
5. ?>
```

Example

All three parameters are optional, but semicolon (;) is must to pass in for loop. If we don't pass parameters, it will execute infinite.

```
1. <?php
2.     $i = 1;
3.     //infinite loop
4.     for (;;) {
5.         echo $i++;
6.         echo "</br>";
7.     }
8. ?>
```

PHP Nested For Loop

We can use for loop inside for loop in PHP, it is known as nested for loop. The inner for loop executes only when the outer for loop condition is found true.

In case of inner or nested for loop, nested for loop is executed fully for one outer for loop. If outer for loop is to be executed for 3 times and inner for loop for 3 times, inner for loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example

```
1. <?php
2. for($i=1;$i<=3;$i++){
3. for($j=1;$j<=3;$j++){
4. echo "$i   $j<br/>";
5. }
6. }
7. ?>
```

Output:

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

3 3

PHP For Each Loop

PHP for each loop is used to traverse array elements.

It works only on array and object. It will issue an error if you try to use it with the variables of different datatype.

The foreach loop works on elements basis rather than index. It provides an easiest way to iterate the elements of an array.

In foreach loop, we don't need to increment the value.

Syntax

```
1. for each( $array as $var ){
2.  //code to be executed
3. }
4. ?>
```

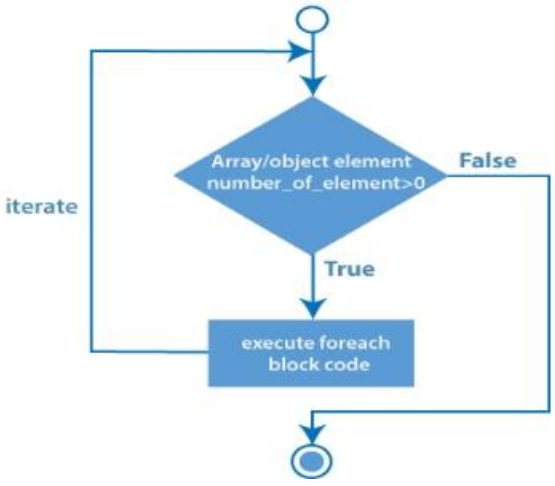
Example

```
1. <?php
2. $season=array("summer","winter","spring","autumn");
3. for each( $season as $arr ){
4.   echo "Season is: $arr<br />";
5. }
6. ?>
```

Output:

Season is: winter
Season is: summer
Season is: winter
Season is: spring
Season is: autumn

Flowchart



PHP While Loop

PHP while loop can be used to traverse set of code like for loop. The while loop executes a block of code repeatedly until the condition is FALSE. Once the condition gets FALSE, it exits from the body of loop.

It should be used if the number of iterations is not known.

The while loop is also called an Entry control loop because the condition is checked before entering the loop body. This means that first the condition is checked. If the condition is true, the block of code will be executed.

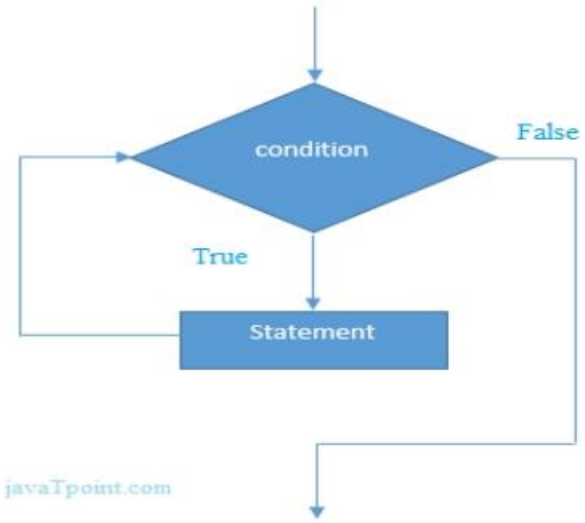
Syntax

```
1. while(condition){
2. //code to be executed
3. }
```

Alternative Syntax

```
1. while(condition):
2. //code to be executed
3.  endwhile;
```

PHP While Loop Flowchart



PHP While Loop Example

```
1. <?php
2. $n=1;
3. while($n<=10){
4. echo "$n<br/>";
5. $n++;
6. }
7. ?>
```

Output:

1
2
3
4
5
6
7
8
9
10

PHP Nested While Loop

We can use while loop inside another while loop in PHP, it is known as nested while loop.

In case of inner or nested while loop, nested while loop is executed fully for one outer while loop. If outer while loop is to be executed for 3 times and nested while loop for 3 times, nested while loop will be executed 9 times (3 times for 1st outer loop, 3 times for 2nd outer loop and 3 times for 3rd outer loop).

Example

```
1. <?php
```

```
2. $i=1;
3. while($i<=3){
4. $j=1;
5. while($j<=3){
6. echo "$i   $j<br/>";
7. $j++;
8. }
9. $i++;
10. }
11. ?>
```

Output:

1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3

PHP Infinite While Loop

If we pass TRUE in while loop, it will be an infinite loop.

Syntax

```
1. while(true) {
2. //code to be executed
3. }
```

Example

```
1. <?php
2.     while (true) {
3.         echo "Hello ";
4.         echo "</br>";
5.     }
6. ?>
```

Output:

Hello
Hello
Hello
Hello
.
.
.

PHP do-while loop

PHP do-while loop can be used to traverse set of code like php while loop. The PHP do-while loop is guaranteed to run at least once.

The PHP do-while loop is used to execute a set of code of the program several times. If we have to execute the loop at least once and the number of iterations is not even fixed, it is recommended to use the do-while loop.

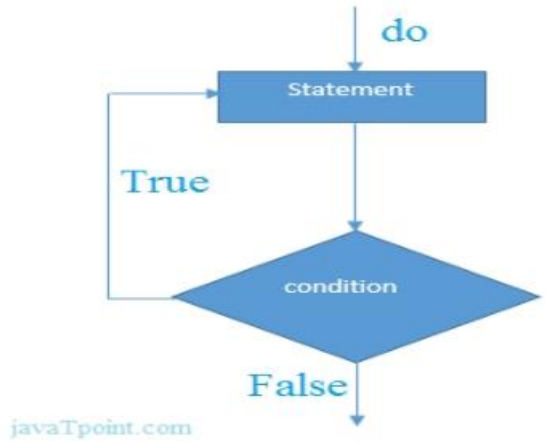
It executes the code at least one time always because the condition is checked after executing the code.

The do-while loop is very much similar to the while loop except the condition check. The main difference between both loops is that while loop checks the condition at the beginning, whereas do-while loop checks the condition at the end of the loop.

Syntax

```
1. do{
2. //code to be executed
3. }while(condition);
```

Flowchart



Example

```
1. <?php
2. $n=1;
3. do{
4. echo "$n<br/>";
5. $n++;
6. }while($n<=10);
7. ?>
```

Output:

1
2
3
4
5
6
7
8
9
10

Example

A semicolon is used to terminate the do-while loop. If you don't use a semicolon after the do-while loop, it is must that the program should not contain any other statements after the do-while loop. In this case, it will not generate any error.

Difference between while and do-while loop

while Loop	do-while loop
The while loop is also named as entry control loop.	The do-while loop is also named as exit control loop.

The body of the loop does not execute if the condition is false.	The body of the loop executes at least once, even if the condition is false.
Condition checks first, and then block of statements executes.	Block of statements executes first and then condition checks.
This loop does not use a semicolon to terminate the loop.	

PHP Break

PHP break statement breaks the execution of the current for, while, do-while, switch, and for-each loop. If you use break inside inner loop, it breaks the execution of inner loop only.

The break keyword immediately ends the execution of the loop or switch structure. It breaks the current flow of the program at the specified condition and program control resumes at the next statements outside the loop.

The break statement can be used in all types of loops such as while, do-while, for, foreach loop, and also with switch case.

Flowchart

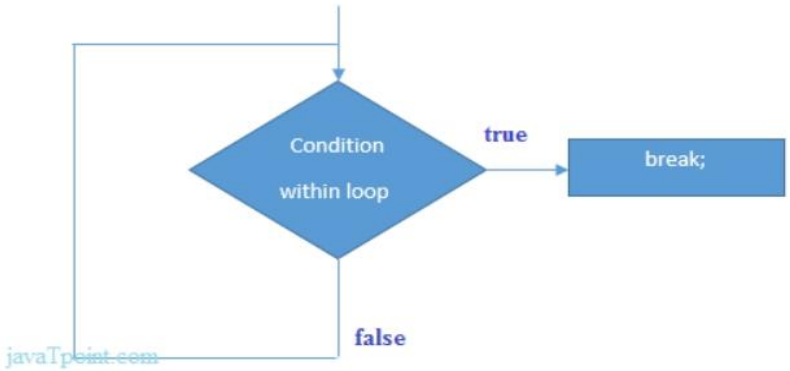


Figure: Flowchart of break statement

PHP Break: inside loop

Let's see a simple example to break the execution of for loop if value of i is equal to 5

```
1. <?php
2. for($i=1;$i<=10;$i++){
3. echo "$i <br/>";
4. if($i==5){
5. break;
6. }
7. }
8. ?>
```

Output:

1
2
3
4
5

PHP Break: inside inner loop

The PHP break statement breaks the execution of inner loop only.

```
<?php
for($i=1;$i<=3;$i++){
 for($j=1;$j<=3;$j++){
  echo "$i   $j<br/>";
  if($i==2 && $j==2){
   break;
  }
 }
}
?>
```

Output:

```
1 1
1 2
1 3
2 1
2 2
3 1
3 2
3 3
At matched condition i = 10; quitting
```

PHP continue statement

The PHP continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

The continue statement is used within looping and switch control structure when you immediately jump to the next iteration.

The continue statement can be used with all types of loops such as - for, while, do-while, and foreach loop. The continue statement allows the user to skip the execution of the code for the specified condition.

PHP Continue Example with for loop

Example

In the following example, we will print only those values of i and j that are same and skip others.

```
<?php
    //outer loop
    for ($i =1; $i<=3; $i++) {
        //inner loop
        for ($j=1; $j<=3; $j++) {
            if (!($i == $j) ) {
                continue;      //skip when i and j does not have same values
            }
            echo $i.$j;
            echo "</br>";
        }
    }
?>
```

Output:

```
11
22
```

33
PHP continue Example in while loop
Example

In the following example, we will print the even numbers between 1 to 20.

```
1.  <?php
2.     //php program to demonstrate the use of continue statement
3.
4.     echo "Even numbers between 1 to 20: </br>";
5.     $i = 1;
6.     while ($i<=20) {
7.        if ($i %2 == 1) {
8.           $i++;
9.           continue;   //here it will skip rest of statements
10.       }
11.       echo $i;
12.       echo "</br>";
13.       $i++;
14.    }
15. ?>
```

Output:

Even numbers between 1 to 20:
2
4
6
8
10
12
14
16
18
20

Arrays

Array is a list of values. Each value within an array is called an element , and each element is referenced by its own index , which is unique to that array. To access an element use that element ’ s index.

PHP — support two types of arrays:

Indexed arrays — These are arrays where each element is referenced by a numeric index, usually starting from zero. For example, the first element has an index of 0, the second has an index of 1, and so on

Associative arrays — This type of array is also referred to as a hash or map. With associative arrays, each element is referenced by a string index. For example, you might create an array element representing a customer ’ s age and give it an index of “ age.

Creating Arrays

The simplest way to create a new array variable is to use PHP ’ s built - in array() construct. This takes a list of values and creates an array containing those values, which you can then assign to a variable:

$authors = array(“Steinbeck”, “Kafka”, “Tolkien”, “Dickens”);

This array is an indexed array, which means that each of the array elements is accessed via its own numeric index, starting at zero. In this case, the “ Steinbeck ” element has an index of 0 , “ Kafka ” has an index of 1 , “ Tolkien ” has an index of 2 , and “ Dickens ” has an index of 3

Following all three examples produce exactly the same array:

```
// Creating an array using the array() construct
$authors1 = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );

// Creating the same array using [] and numeric indices
$authors2[0] = "Steinbeck";
$authors2[1] = "Kafka";
$authors2[2] = "Tolkien";
$authors2[3] = "Dickens";

// Creating the same array using the empty [] syntax
$authors3[] = "Steinbeck";
$authors3[] = "Kafka";
$authors3[] = "Tolkien";
$authors3[] = "Dickens";
```

If you want to create an associative array, where each element is identified by a string index rather than a number, you need to use the => operator, as follows:

```
$myBook = array( "title" => "The Grapes of Wrath",
                 "author" => "John Steinbeck",
                 "pubYear" => 1939 );
```

Here an associative array is populated in two ways: first using the array() construct, and second using the square bracket syntax:

```
// Creating an associative array using the array() construct
$myBook = array( "title" => "The Grapes of Wrath",
                 "author" => "John Steinbeck",
                 "pubYear" => 1939 );

// Creating the same array using [] syntax
$myBook = array();
$myBook["title"] = "The Grapes of Wrath";
$myBook["author"] = "John Steinbeck";
$myBook["pubYear"] = 1939;
```

This creates an array with three elements: “ The Grapes of Wrath ” , which has an index of “ title “ ; “ John Steinbeck ”, which has an index of “ author “ ; and 1939 , which has an index of “ pubYear ”.

Accessing Arrays

- Use numeric indices in index array
- Use string indices in associative array

$myTitle = $myBook[“title”]; // $myTitle contains “The Grapes of Wrath”

$myAuthor = $myBook[“author”]; // $myAuthor contains “Steinbeck”

Changing Elements

$authors = array(“Steinbeck”, “Kafka”, “Tolkien”, “Dickens”);

$authors[2] = “Melville”;

Changing elements of associative arrays works in a similar fashion to indexed arrays:

```
$myBook["title"] = "East of Eden";
$myBook["pubYear"] = 1952;
```

Outputting an Entire Array with print_r()

You can ’ t just print an array with print() or echo() , like you can with regular variables, because these functions can work with only one value at a time. However, PHP does give you a function called print_r() that you can use to output the contents of an array

print_r ($authors);

print_r ($myBook);

Extracting a Range of Elements with array_slice()

To use it, pass it the array to extract the slice from, followed by the position of the first element in the range (counting from zero), followed by the number of elements to extract.

```
$authors = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );
$authorsSlice = array_slice( $authors, 1, 2 );

// Displays "Array ( [0] => Kafka [1] => Tolkien )"
print_r( $authorsSlice );
```

This example extracts the second and third elements from the $authors array and stores the resulting array in a new variable, $authorsSlice . The code then uses print_r() to display the slice.

you can use array_slice() with associative arrays. Although associative arrays don ’ t have numeric indices, PHP does remember the order of the elements in an associative array. So you can still tell array_slice() to extract, say, the second and third elements of an associative array:

```
$myBook = array( "title" => "The Grapes of Wrath",
                 "author" => "John Steinbeck",
                 "pubYear" => 1939 );
$myBookSlice = array_slice( $myBook, 1, 2 );

// Displays "Array ( [author] => John Steinbeck [pubYear] => 1939 )";
print_r( $myBookSlice );
```

if you leave out the third argument to array_slice() , the function extracts all elements from the start position to the end of the array:

```
$authors = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );
$authorsSlice = array_slice( $authors, 1 );

// Displays "Array ( [0] => Kafka [1] => Tolkien [2] => Dickens )";
print_r( $authorsSlice );
```

To preserve the indices, you can pass a fourth argument, true , to array_slice()

```
$authors = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );

// Displays "Array ( [0] => Tolkien [1] => Dickens )";
print_r( array_slice( $authors, 2, 2 ) );

// Displays "Array ( [2] => Tolkien [3] => Dickens )";
print_r( array_slice( $authors, 2, 2, true ) );
```

Counting Elements in an Array

pass the array to count()

```
echo count( $authors ) . "<br/>"; // Displays "4"
echo count( $myBook ) . "<br/>";  // Displays "3"
```

Array handling using while and for

Using while

while ($element = each($myBook)) {

echo " $element[0] ";

echo " $element[1] ";

}

Using foreach to Loop Through Values

```
foreach ( $array as $value ) {
  // (do something with $value here)
}
```

Here's an example:

```
$authors = array( "Steinbeck", "Kafka", "Tolkien", "Dickens" );

foreach ( $authors as $val ) {
  echo $val . "<br/>";
}
```

To use foreach to retrieve both keys and values, use the following syntax:

```
foreach ( $array as $key => $value ) {
  // (do something with $key and/or $value here
}
```

$myBook = array("title" = > "The Grapes of Wrath",

"author" = > "John Steinbeck",

"pubYear" = >1939);

foreach ($myBook as $key = > $value) {

```
echo " $key ";
echo "  $value ";
}
```

Working with Multidimensional Arrays

Id	Name	Salary
1	sonoo	400000
2	john	500000
3	rahul	300000

```
$emp = array
(
array(1,"sonoo",400000),
array(2,"john",500000),
array(3,"rahul",300000)
);
```

Example of PHP multidimensional array to display tabular data

```
<?php
$emp = array
  (
  array(1,"sonoo",400000),
  array(2,"john",500000),
  array(3,"rahul",300000)
  );

for ($row = 0; $row < 3; $row++) {
  for ($col = 0; $col < 3; $col++) {
    echo $emp[$row][$col]."  ";
  }
  echo "<br/>";
}
?>
```

Creating a Multidimensional Associative Array

```php
<?php

$myBooks = array(
  array(
    "title" => "The Grapes of Wrath",
    "author" => "John Steinbeck",
    "pubYear" => 1939
  ),
  array(
    "title" => "The Trial",
    "author" => "Franz Kafka",
    "pubYear" => 1925
  ),
  array(
    "title" => "The Hobbit",
    "author" => "J. R. R. Tolkien",
    "pubYear" => 1937
  ),
  array(
    "title" => "A Tale of Two Cities",
    "author" => "Charles Dickens",
    "pubYear" => 1859
  ),
);
```

Accessing Elements of Multidimensional Arrays

print_r ($myBooks);

Displays all elements

print_r($myBooks[1])

echo $myBooks[3][“pubYear”] . “ < br/ > ”;

PHP ’ s powerful array - manipulation functions, including:

- Sorting functions— sort() , asort() , ksort() and array_multisort()
- Functions for adding and removing elements — array_unshift() , array_

shift() , array_push() , array_pop() and array_splice()

- array_merge() – – This function is useful for merging two or more arrays together
- explode() and implode() — to convert between arrays and strings
- list() – – to store array elements in a list of individual variables

---

Strings

1.String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator −

```php
<?php
   $string1="Hello World";
   $string2="1234";

   echo $string1 . " " . $string2;
?>
```

Output: Hello World 1234

2.The strlen() function is used to find the length of a string.
Let's find the length of our string "Hello world!" –

```
<?php
  echo strlen("Hello world!");
?>
```

output
12

3. str_word_count() - Count Words in a String
Example
Count the number of word in the string "Hello world!":

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

4. strrev() - Reverse a String
The PHP strrev() function reverses a string.
Example
Reverse the string "Hello world!":

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

5. str_replace() - Replace Text Within a String
The PHP str_replace() function replaces some characters with some other characters in a string.
Example
Replace the text "world" with "Dolly":

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

6. strpos – Find the position of a string in another string

```
<?php
  echo strpos("Hello world!","world");
?>
```

This will produce the following result –
6

7. trim() function: This function allows us to remove whitespaces or strings from both sides of a string.
Example:

```
<?php
 echo trim("Hello World!", "Hel!");
 ?>
```

Output:
llo Worl
8. explode() function: This function converts a string into an array.
Example:

```
<?php
```

```
 $input  = "Welcome to PHP Class";
 print_r(explode(" ",$input));
  ?>
```

Output:

Array ([0] => Welcome [1] => to [2] => PHP [3] => Class)

9. strtolower() function: This function converts a string into the lowercase string.

Example:

```
<?php
 $input  = "WELCOME “;
  echo strtolower($input);
  ?>
```

Output:

welcome

10. strtoupper() function: This function converts a string into the uppercase string.

Example:

```
<?php
  $input  = "Welcome ";
  echo strtoupper($input);
 ?>
```

Output:

WELCOME

PHP Functions

Function is a named block of code for a specific task that can be reused many times.

Advantage of Functions

Code Reusability: PHP functions are defined only once and can be invoked many times, like in other programming languages.

Less Code: No need to write the logic many times. Write the logic only once and reuse it.

Easy to understand: It is easier to understand the flow of the application because every logic is divided in the form of functions.

Syntax

```
function functionname(){
//code to be executed
}
```

Example

```
<?php
function sayHello($name,$age){
echo "Hello $name, you are $age years old<br/>";
}
sayHello("aaa",20);
?>
```

PHP Function Arguments

We can pass the data in PHP function through arguments which is separated by comma.

PHP supports Call by Value (default), Call by Reference, Default argument values and Variable-length argument list.

1. Call by Value

In call by value, actual value is not affected if it is modified inside the function.

Example:

```
<?php
function   increment($i)
{
   ++$i;
}
$i =10;
Increment($i);
echo $i;
?>
```

Output

10

2. Call By Reference

In call by reference, actual value is affected if it is modified inside the function. In this case, use & (ampersand) symbol with formal arguments. The & represents reference of the variable.

```
<?php
function increment(&$i)
{
   $i++;
}
$i = 10;
increment($i);
echo $i;
?>
```

Output:

11

3. Default argument values

In this case, if we don't pass any value to the function, it will use default argument value.

```php
<?php
function add($n1=10,$n2=10){
$n3=$n1+$n2;
echo "Total is: $n3<br/>";
}
add();
add(20);
add(40,40);
?>
```

Output:

Total is: 20

Total is: 30

Total is: 80

4. Variable Length Arguments

This means we can pass 0, 1 or n number of arguments in function. To do so, use 3 dots before the argument name.

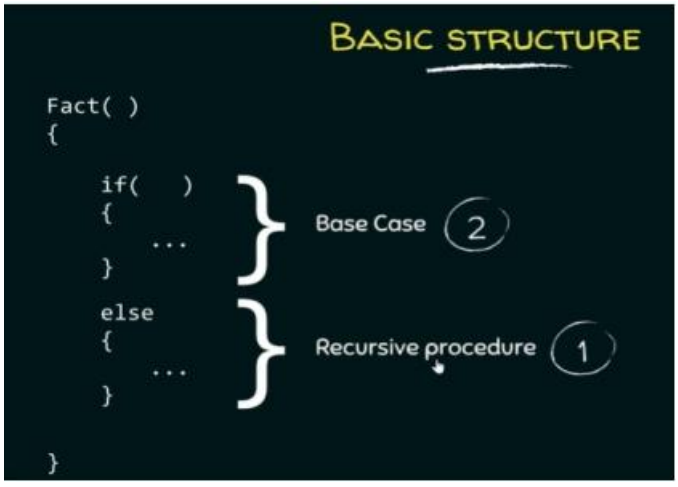
```php
<?php
function add(...$n) {
   $sum = 0;
   foreach ($n as $i) {
      $sum += $i;
   }
   return $sum;
}

echo add(1, 2, 3, 4);
?>
```

Output:

10

Recursive Function

Function calling itself. It is important to specify the base condition to stop recursion. Base condition is the one which doesn't call the same function again.



Factorial Number

```
<?php
function factorial($n)
{
   if ($n == 0)
      return 1; /*base case*/
   return ($n * factorial ($n -1));
}
echo factorial(5);
?>
```

UNIT – III

File Handling

Opening and Closing Files

To work with a file, first open the file. When you open a file, you create a file handle. A file handle is a pointer associated with the open file that you can then use to access the file ' s contents.

Opening a File with fopen()

The fopen() function opens a file and returns a file handle associated with the file. The first argument passed to fopen() specifies the name of the file you want to open, and the second argument specifies the mode , or how the file is to be used.

Value	Description
r	Open the file for reading only. The file pointer is placed at the beginning of the file.
r+	Open the file for reading and writing. The file pointer is placed at the beginning of the file.
w	Open the file for writing only. Any existing content will be lost. If the file does not exist, PHP attempts to create it.
w+	Open the file for reading and writing. Any existing file content will be lost. If the file does not exist, PHP attempts to create it.
a	Open the file for appending only. Data is written to the end of an existing file. If the file does not exist, PHP attempts to create it.
a+	Open the file for reading and appending. Data is written to the end of an existing file. If the file does not exist, PHP attempts to create it.

Example

```
1. <?php
2. $handle = fopen("c:\\folder\\file.txt", "r");
3. ?>
```

Close File

```
fclose($handle )
```

Read File

PHP provides various functions to read data from file. The available PHP file read functions are given below.

- fread() - read all file data,
- fgets() - read data line by line
- fgetc() - read data character by character.

fread()

fread($handle , $length)

Example

```
1. <?php
2. $filename = "c:\\myfile.txt";
3. $handle = fopen($filename, "r");//open file in read mode
4. $contents = fread($handle, filesize($filename));//read file
5. echo $contents;//printing data of file
6. fclose($handle);//close file
7. ?>
```

fgets()

The PHP fgets() function is used to read single line from the file.

Syntax

```
1. string fgets ( resource $handle [, int $length ] )
```

Example

```
1. <?php
2. $fp = fopen("c:\\file1.txt", "r");//open file in read mode
3. echo fgets($fp);
4. fclose($fp);
5. ?>
```

fgetc()

The PHP fgetc() function is used to read single character from the file. To get all data using fgetc() function, use !feof() function inside the while loop.

Syntax

```
1. string fgetc ( resource $handle )
```

Example

```
1. <?php
2. $fp = fopen("c:\\file1.txt", "r");//open file in read mode
3. while(!feof($fp)) {
4.   echo fgetc($fp);
5. }
6. fclose($fp);
7. ?>
```

Write File - fwrite()

Syntax

fwrite ($handle , $string)

Example

```
1. <?php
2. $fp = fopen('data.txt', 'w');//open file in write mode
3. fwrite($fp, 'hello ');
4. fwrite($fp, 'php file');
5. fclose($fp);
6.
7. echo "File written successfully";
8. ?>
```

Append to File - fwrite()

The PHP fwrite() function is used to write and append data into file.

Example

```
1. <?php
2. $fp = fopen('data.txt', 'a');//opens file in append mode
3. fwrite($fp, ' this is additional text ');
4. fwrite($fp, 'appending data');
5. fclose($fp);
6.
```

```
7. echo "File appended successfully";
8. ?>
```

Delete File - unlink()
The PHP unlink() function is used to delete file.

Syntax

```
unlink (  $filename  )
```

Example

```
1. <?php
2. unlink('data.txt');
3. echo "File deleted successfully";
4. ?>
```

Working with File Permissions

File system permissions determine what different users can do with each file and directory in the file system. For example, whereas one user might have permission to read and write to a file, another user may only be allowed to read the file. A third user might not even be allowed to do that.

Generally, PHP automatically gives file read and write permission and gives directory read, write, and execute permission for all users by default, meaning that anyone can create and delete files within that directory. There are special functions to change this default permission.

i) Changing Permissions

PHP ’ s chmod() function is used to change the mode, or permissions, of a file or directory. It functions much like the UNIX chmod command.

To change a file ’ s permissions with chmod() , pass it the filename and the new mode to use.

chmod(“myfile.txt”, 0644);

The 0 (zero) before the 644 is important, because it tells PHP to interpret the digits as an octal number.

File modes are usually expressed as octal numbers containing three digits.

- The first digit determines what the file ’ s owner – – usually the user that created the file — can do with the file.
- The second digit determines what users in the file ’ s group — again, usually the group of the user that created the file — can do with it.
- Finally, the last digit dictates what everyone else can do with the file.

The value of each digit represents the access permission for that particular class of user, as follows

Digit Value	Permission
0	Cannot read, write to, or execute the file
1	Can only execute the file
2	Can only write to the file
3	Can write to and execute the file
4	Can only read the file
5	Can read and execute the file
6	Can read and write to the file
7	Can read, write to, and execute the file

chmod() returns true if the permission change was successful, and false if it failed (for example, you ' re not the owner of the file).

Examples

```
// Owner can read and write the file; everyone else can just read it:
chmod( "myfile.txt", 0644 );

// Everyone can read and write the file:
chmod( "myfile.txt", 0666 );

// Everyone can read and execute the file, but only the owner can write to it:
chmod( "myfile.txt", 0755 );

// Only the owner can access the file, and they can only read and write to it:
chmod( "myfile.txt", 0600 );
```

[Note that you can only change the permissions of a file or directory if you own it, or if you ' re the super – user]

In the case of directories

To read the files in a directory, **you need to have both read and execute permissions** on that directory.

To create and delete files and subdirectories inside the directory, **you need to have write and execute permissions** on the directory.

ii) Checking File Permissions

PHP provides three handy functions.

- is_readable() - To check if you ' re allowed to read a file
- is_writable() - To check if you ' re allowed to write to a file
- is_writable() - To check if you ' re allowed to execute a file .

Each function returns true if the operation is allowed, or false if it ' s disallowed.

Example:

```
if ( is_readable( "myfile.txt" ) {
  echo "I can read myfile.txt";
}

if ( is_writable( "myfile.txt" ) {

  echo "I can write to myfile.txt";
}

if ( is_executable( "myfile.txt" ) {
  echo "I can execute myfile.txt";
}
```

Working with Directories

PHP lets you work with directories in much the same way as files, using a variety of equivalent functions.

- Some directory functions use a directory handle, whereas others use a string containing the name of the directory with which you want to work.
- A directory handle is similar to a file handle; it ' s a special variable pointing to a directory, which you can obtain via the opendir() function:

Directory functions

1. **opendir()**

$handle = opendir("/home/abc");

If there ' s a problem in opening the directory (for example, if the directory doesn ' t exist), opendir() returns false instead of the directory handle.

closedir()

closedir($handle);

To close a directory by passing the directory handle to the function

2. **readdir()**

$filename = readdir($handle);

- Give the directory handle of an opened directory.
- It returns the filename of the next entry in the directory.

[Each directory contains a list of entries for each of the files and subdirectories inside it, as well as entries for . (representing the directory) and .. (the parent of the directory).

PHP maintains an internal pointer referring to the next entry in the list, just as a file pointer points to the position in a file where the next file operation should occur.]

3. **rewinddir()**

Move the directory pointer back to the start of the list of entries

rewinddir($handle);

4. **chdir()**

Change the current directory

chdir("/home/abc/myfolder");

chdir() returns true if PHP managed to change to the specified directory, or false if there was an error (such as the directory not being found).

5. **mkdir()**

Create a directory .

mkdir("/home/matt/newfolder");

mkdir() returns true if the directory was created, or false if there was a problem. You can also set permissions for the directory at the time of creation by passing the mode as the second argument.

mkdir("/home/matt/newfolder", 0777);

6. **rmdir()**

Deletes a directory

rmdir("/home/abc/myfolder");

The rmdir() function removes a given directory. The directory must be empty, and you need appropriate permissions to remove it.

7. **dirname()** —

Returns the directory portion of a path

8. **basename()** –

Returns the filename portion of a given path

Example 1

$path = "/home/abc/docs/pgm1.html";

$directoryPath = dirname($path);

$filename = basename($path);

After running this code

$directoryPath contains " /home/abc/docs " ,

$filename contains " pgm1.html "

Example 2:

To determine if a file called myfile is a file or a directory

```
$filename = "myfile";

if ( is_dir( $filename ) ) {
  echo "$filename is a directory.";
} elseif ( is_file( $filename ) ) {
  echo "$filename is a file.";
} else {
  echo "$filename is neither a directory nor a file.";
}
```

Example 3: Recursive function

To create a script that lists all the files and subdirectories under a given directory — including subdirectories of subdirectories, and so on

Steps:

1. Read the entries in the current directory.

2. If the next entry is a file, display its name.

3. If the next entry is a subdirectory, display its name, then call the function recursively to read the

entries inside it.

```
<?php
$dirPath = “/home/abc/images”;

function traverseDir( $dir ) {
    if ( !( $handle = opendir( $dir ) ) ) echo ( “Cannot open $dir.” );

 $files = array();

 while ( $file = readdir( $handle ) ) {
 if ( $file != “.” && $file != “..” ) {
 if ( is_dir( $dir . “/” . $file ) )
   $file .= “/”;
  $files[] = $file;
 }
 }

foreach ( $files as $file ) {
 if ( substr( $file, -1 ) == “/” )
traverseDir( “$dir/” . substr( $file, 0, -1 ) );
 }

 closedir( $handle );
}
 traverseDir( $dirPath );

?>
```

Description

- open the directory with opendir()
  ```
  if ( !( $handle = opendir( $dir ) ) )
  ```

- set up a $files array to hold the list of filenames

```
$files = array();
```

- Use readdir() with a while loop to move through each entry in the directory, adding each filename to the array as it goes (”.” and “..” are skipped).
- If a particular filename is a directory, a slash (/) is added to the end of the filename to indicate to the user (and the rest of the function) that the file is in fact a directory:

```
$files = array();

 while ( $file = readdir( $handle ) ) {
```

```
if ( $file != “.” && $file != “..” ) {
if ( is_dir( $dir . “/” . $file ) ) $file .= “/“;
$files[] = $file;
}
}
```

- loop through the array again, looking for any directories (where the filename ends in a slash). If it finds a directory, the function calls itself with the directory path (minus the trailing slash) to explore the contents of the directory:

```
foreach ( $files as $file ) {
if ( substr( $file, -1 ) == “/” )
traverseDir( “$dir/” . substr( $file, 0, -1 ) );
}
```

- Finally, close the directory handle

```
closedir( $handle );
```

Introduction to database and SQL

A database engine — commonly known as a Database Management System (DBMS) — to store, retrieve, and modify the data

Database architecture

Two main options: embedded and client - server.

i) Embedded Databases

An embedded database engine, as its name implies, tied up with the application that uses it (PHP in this case). Therefore it always runs and stores its data on the **same machine** as the host application.

The database is not networked, and **only one program can connect to it at any given time.** Moreover, the database can ’ t **be shared between different machines** because each one would simply end up storing and manipulating its own separate version of the data.

On the plus side, embedded databases tend to be faster, easier to configure, and easier to work with. Long - standing examples of embedded database engines include **dBase** and dbm, and PHP supports both these engines in the form of PHP extensions.

ii) Client - Server Databases

Client - server databases are, generally speaking, more powerful and flexible than embedded databases. They are usually designed for use over networks, **enabling many applications in a network to work simultaneously with the same data**. The database engine itself acts as a server, serving up data to its clients (much like Web servers serve pages to Web browsers).

This is the kind of database which are more likely to find in a large company, where large quantities of data need to be shared among many people, where access may be needed from all sorts of different locaions, and where having a single centralized data store

Most relational databases — including **Oracle, DB2, and SQL Server**

Fortunately, alternatives are available, such as PostgreSQL and **MySQL**, which are both open source relational database systems that have proven very popular with **PHP developers** for many years.

Reasons for choosing a Database MySQL

MySQL. offers several advantages:

- It ’ s one of the most popular databases being used on the Web today

- It ' s freely available as a download to install and run on your own machine
- It ' s easy to install on a wide range of operating systems (including UNIX, Windows, and Mac OS X)
- It ' s simple to use and includes some handy administration tools
- It ' s a fast, powerful system that copes well with large, complex databases

MySQL Data Types

MySQL supports three main groups of data types — numeric, date/time, and string —

i) Numeric Data Types

Numeric Data Type	Description	Allowed Range of Values
TINYINT	Very small integer	–128 to 127, or 0 to 255 if UNSIGNED
SMALLINT	Small integer	–32768 to 32767, or 0 to 65535 if UNSIGNED
MEDIUMINT	Medium-sized integer	–8388608 to 8388607, or 0 to 16777215 if UNSIGNED
INT	Normal-sized integer	–2147483648 to 2147483647, or 0 to 4294967295 if UNSIGNED
BIGINT	Large integer	–9223372036854775808 to 9223372036854775807, or 0 to 18446744073709551615 if UNSIGNED
FLOAT	Single-precision floating-point number	Smallest non-zero value: $\pm 1.176 \times 10^{-38}$; largest value: $\pm 3.403 \times 10^{38}$
DOUBLE	Double-precision floating-point number	Smallest non-zero value: $\pm 2.225 \times 10^{-308}$; largest value: $\pm 1.798 \times 10^{308}$
DECIMAL(*precision*, *scale*)	Fixed-point number	Same as DOUBLE, but fixed-point rather than floating-point. *precision* specifies the total number of allowed digits, whereas *scale* specifies how many digits sit to the right of the decimal point.
BIT	0 or 1	0 or 1

ii) Date and Time Data Types

Date/Time Data Type	Description	Allowed Range of Values
DATE	Date	1 Jan 1000 to 31 Dec 9999
DATETIME	Date and time	Midnight, 1 Jan 1000 to 23:59:59, 31 Dec 9999
TIMESTAMP	Timestamp	00:00:01, 1 Jan 1970 to 03:14:07, 9 Jan 2038, UTC (Universal Coordinated Time)
TIME	Time	–838:59:59 to 838:59:59
YEAR	Year	1901 to 2155

To specify a literal DATE , DATETIME , or TIMESTAMP value in MySQL, you can use any of the following formats:

- YYYY - MM - DD / YY - MM - DD
- YYYY - MM - DD HH:MM:SS / YY - MM - DD HH:MM:SS
- YYYYMMDD / YYMMDD
- YYYYMMDDHHMMSS / YYMMDDHHMMSS

iii) String Datatype

String Data Type	Description	Allowed Lengths
CHAR(*n*)	Fixed-length string of *n* characters	0–255 characters
VARCHAR(*n*)	Variable-length string of up to *n* characters	0–65535 characters
BINARY(*n*)	Fixed-length binary string of *n* bytes	0–255 bytes
VARBINARY(*n*)	Variable-length binary string of up to *n* bytes	0–65535 bytes
TINYTEXT	Small text field	0–255 characters
TEXT	Normal-sized text field	0–65535 characters
MEDIUMTEXT	Medium-sized text field	0–16777215 characters
LONGTEXT	Large text field	0–4294967295 characters
TINYBLOB	Small BLOB (Binary Large Object)	0–255 bytes
BLOB	Normal-sized BLOB	0–65535 bytes
MEDIUMBLOB	Medium-sized BLOB	0–16777215 bytes (16MB)
LONGBLOB	Large BLOB	0–4294967295 bytes (4GB)
ENUM	Enumeration	The field can contain one value from a predefined list of up to 65,535 values
SET	A set of values	The field can contain zero or more values from a predefined list of up to 64 values

Introducing SQL Statements

To actually work with databases and tables, you use SQL statements. Common statements include:

CREATE — Creates a database, table or index

SELECT — Retrieves data from one or more tables

INSERT — Inserts data into a table

REPLACE — Replaces data in a table.

UPDATE — Updates data in a table

DELETE — Deletes data from a table

ALTER — Modifies the structure of a table

DROP — Wipes out a database or table

i) Creating a New Database

To create a new database, all you have to do is use the CREATE DATABASE command. Type the following to create a new database called mydatabase :

mysql > CREATE DATABASE mydatabase;

You can see a list of all the databases in the system— including your new database — by typing the command

mysql > SHOW DATABASES

ii) Creating a Table

Type the following at the mysql > prompt:

mysql > CREATE TABLE fruit (

- > id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,

- > name VARCHAR(30) NOT NULL,

- > color VARCHAR(30) NOT NULL,

- > PRIMARY KEY (id)

- >);

[**Note:** Press Enter at the end of each line. Don ' t enter the " -> " arrows; MySQL displays these automatically each time you press Enter, to inform you that your statement is being continued on a new line.]

To see a list of tables in your database, use the SHOW TABLES command:

mysql >SHOW TABLES;

To see the structure of created table by using the EXPLAIN command, as follows:

mysql >EXPLAIN fruit;

iii) Adding Data to a Table

INSERT INTO table VALUES (value1 , value2 , ...);

iv) Retrieving data

mysql > SELECT * from fruit

To retrieve a selected row or rows, introduce a WHERE clause at the end of the SELECT statement. A WHERE clause filters the results according to the condition in the clause.

mysql > SELECT * from fruit WHERE name = 'banana';

v) Updating Data in a Table

change existing data in a table with the UPDATE statement. As with the SELECT statement, a WHERE clause can be used to specify exactly which rows to update. Otherwise, the entire table gets updated.

mysql > UPDATE fruit SET name = 'grapefruit', color = 'yellow' WHERE id = 2;

vi) Deleting Data from a Table

To delete rows, you use the DELETE statement. If you add a WHERE clause, you can choose which row or rows to delete; otherwise all the data in the table are deleted

mysql > DELETE FROM fruit WHERE id = 2;

vii) Deleting Tables and Databases

To delete a table entirely, use the DROP TABLE statement. Similarly, you can delete an entire database with DROP DATABASE .

Connecting to MySQL from PHP

PHP provides two main ways to connect to MySQL databases:

i) mysqli (MySQL improved) — This extension is specifically tied to MySQL, and provides the most complete access to MySQL from PHP. It features both procedural (function - oriented) and object - oriented interfaces.

ii) PDO (PHP Data Objects) — This is an object - oriented extension that sits between the MySQL server and the PHP engine. It gives you a nice, simple, clean set of classes and methods that you can use to work with MySQL databases. Furthermore, you can use the same extension to talk to lots of other database systems

Making a Connection

To make a connection to a MySQL database in your PHP script, all you need to do **is create a new PDO object**. When you create the object, you pass in three arguments:

1. DSN, which describes the database to connect to;

A string that describes attributes of the connection such as the **type of database system, the location of the database, and the database name.**

$dsn = “mysql:host=localhost; dbname=mydatabase”;

If host isn ’ t specified, localhost is assumed.

2. username of the user you want to connect as;
3. user ’ s password.

The returned PDO object serves as your script ’ s connection to the database:

$conn = new PDO($dsn, $username, $password);

Example:

$dsn = “mysql:dbname=mydatabase”;

$username = “root”;

$password = “mypass”;

$conn = new PDO($dsn, $username, $password);

Closing the Connection

To close the connection, just assign null to your connection variable. This effectively destroys the PDO object, and therefore the connection:

$conn = null;

Although the PHP engine usually closes connections when a script finishes, it ’ s a good idea to close the connection explicitly to be on the safe side.

Handling Errors

To set PDO to raise exceptions whenever database errors occur, you use the PDO::SetAttribute method to set your PDO object ’ s error mode, as follows:

$conn = new PDO($dsn, $username, $password);

$conn- > setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

Now you can capture any error that might occur when connecting to the database by using a try ... catch code block. If you were writing a sophisticated application, you ' d probably log the error message to a file, and possibly send an email to the Webmaster informing him of the details of the error. For the sake of these examples, though, you ' ll just display the error message in the Web page:

```
try {
  $conn = new PDO( $dsn, $username, $password );
  $conn->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
} catch ( PDOException $e ) {
  echo "Connection failed: " . $e->getMessage();
}
```

PHP runs the code within the try block. If an exception is raised by PDO, the catch block stores the PDOException object in $e , then displays the error message with $e - > getMessage() . For example, if the $password variable in the script contained an incorrect password, you ' d see a message like this appear when you ran the script:

Connection failed: SQLSTATE[28000] [1045] Access denied for user 'root'@'localhost' (using password: YES)

Reading Data

you can read some data from the database using a SELECT statement. To send SQL statements to the MySQL server, you use the **query method** of the PDO object:

$conn- > query ($sql);

If your SQL statement returns rows of data as a result set, you can capture the data by assigning the

result of $conn - > query to a variable:

$rows = $conn- > query ($sql);

The result returned by $conn - > query is actually another type of object, called a PDOStatement object. You can use this object along with a foreach loop to move through all the rows in the result set. Each row is an associative array containing all the field names and values for that row in the table. For example:

$sql = "SELECT * FROM fruit";

$rows = $conn- > query($sql);

foreach ($rows as $row) {

echo "name = " . $row["name"] . "is". $row["color"] . " < br / > ";

}

UNIT IV

History of Python

Created by Guido van Rossum - Netherland

- Started development in 1989
- Launched in 1991 – first version released 0.9.0
- 1994, Python 1.0

Popular in the decade of 2010

- Van Rossum developed Python as a hobby project
  - 1980s - ABC - Amoeba Operating System.
  - He had seen some issues with ABC.
  - He had taken the syntax of ABC, and some of its good features.

- He wanted a
  - short
  - unique
  - mysterious name for his invention.
- Named after a popular BBC's TV Show
  - comedy TV show "Monty Python's Flying Circus".
- **Benevolent dictator for life** (**BDFL**) is a title given by python community.

Features

- Easy-to-learn and use:
  - Python is clearly defined and easily readable.
  - The structure of the program is very simple.
  - It uses few keywords.
- Interpreted:
  - Python is processed at runtime by the interpreter.
  - So, there is no need to compile a program before executing it.
  - You can simply run the program.
- Cross Platform/Portable:
  - Runs anywhere, including Mac OS X, Windows, Linux, Unix, Android and iOS.
  - Has the same interface on all platforms.
- Extensible:
  - Extended easily by adding new modules implemented in C, C++, java etc.
- Free and Open Source:
  - Doesn't cost anything to download and use Python.
  - Python can also be freely modified and re-distributed.
- High Level Language:
  - When writing programs, programmers no need to worry about the low level details.
- Scalable:

Python provides a better structure and support for large programs than shell scripting.

- Libraries
  - large standard library that supports many common programming tasks.
  - Interactive
  - easy to test short snippets of code.
- Object-oriented Language.
  - Object, class, inheritance, overloading
- Dynamically typed.
  - Data types are dynamically typed
  - Mixing incompatible types (e.g. attempting to add a string and a number) will raise exception
  - So errors are caught sooner.

Creating Python Programs

Interactive Mode

- When you enter an expression or statement, Python evaluates it and displays its result immediately.

  chevron, >>>

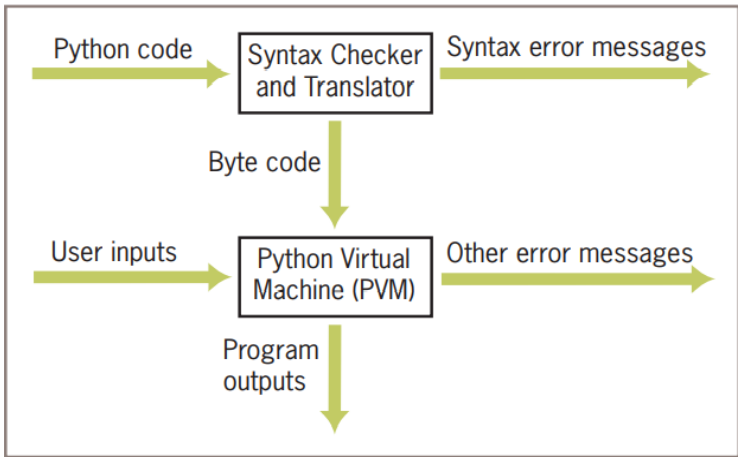
Script Mode

- Type python program in a file called as script
- Python scripts have the extension .py

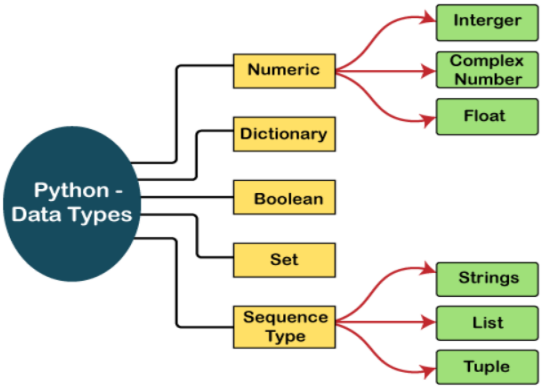
Interactive mode	Script mode
A way of using the Python interpreter by typing commands and expressions at the prompt.	A way of using the Python interpreter to read and execute statements in a script.
Cant save and edit the code	Can save and edit the code
If we want to experiment with the code, we can use interactive mode.	If we are very clear about the code, we can use script mode.
we cannot save the statements for further use and we have to retype all the statements to re-run them.	we can save the statements for further use and we no need to retype all the statements to re-run them.
We can see the results immediately.	We cant see the code immediately.

- Integrated Development Learning Environment (IDLE):
  - Is a graphical user interface which is completely written in Python.
  - It is installed along with the python language .
- Features of IDLE:
  - Multi-window text editor with syntax highlighting
  - Coloring of input, output and error messages
  - Auto completion with smart indentation.

Execution Process



Data Types



- Standard Data Types -5
  - Number
  - String
  - List
  - Tuple
  - Dictionary

- **A floating-point number can be written using either ordinary decimal notation or scientific notation.**
- **Scientific notation is often useful for mentioning very large numbers.**

Decimal Notation	Scientific Notation	Meaning
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e–1	3.78×10^{-1}
0.00378	3.78e–3	3.78×10^{-3}

Write the values of the following floating-point numbers in Python's scientific notation:

a. 355.76 b. 0.007832 c. 4.3212

Arithmetic Expressions

Operator	Meaning	Syntax
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

```
>>> 3//4
0
>>> 3/4
0.75
>>> 3%4
3
>>> 4%3
1
```

Precedence rules for evaluating arithmetic expressions in Python:

- Exponentiation
- Unary negation
- Multiplication, both types of division, and remainder
- Addition and subtraction.

Two exceptions

- Operations of equal precedence are left associative, so they are evaluated from left to right.
- Exponentiation and assignment operations are right associative, so consecutive instances of these are evaluated from right to left.

• You can use parentheses to change the order of evaluation.

Expression	Evaluation	Value
5 + 3 * 2	5 + 6	11
(5 + 3) * 2	8 * 2	16
6 % 2	0	0
2 * 3 ** 2	2 * 9	18
-3 ** 2	-(3 ** 2)	-9
(3) ** 2	9	9
2 ** 3 ** 2	2 ** 9	512
(2 ** 3) ** 2	8 ** 2	64
45 / 0	Error: cannot divide by 0	
45 % 0	Error: cannot divide by 0	

Example:

a=9-12/3+3*2-1 a=? a=9-4+3*2-1 a=9-4+6-1 a=5+6-1 a=11-1 **a=10**	A=2*3+4%5-3/2+6 A=6+4%5-3/2+6 A=6+4-3/2+6 A=6+4-1+6 A=10-1+6 A=9+6 **A=15**	find m=? m=-43||8&&0||-2 m=-43||0||-2 m=1||-2 **m=1**
a=2,b=12,c=1 d=a<b>c d=2<12>1 d=1>1 **d=0**	a=2,b=12,c=1 d=a<b>c-1 d=2<12>1-1 d=2<12>0 d=1>0 **d=1**	a=2*3+4%5-3//2+6 a=6+4-1+6 a=10-1+6 **a=15**

-
- When an expression becomes long or complex, you can move to a new line by placing a backslash character \ at the end of the current line.

```
>>> 3 + 4 * \
2 ** 5
131
```

- Type Conversion

Conversion Function	Example Use	Value Returned
int(<a number or a string>)	int(3.77)	3
	int("33")	33
float(<a number or a string>)	float(22)	22.0
str(<any value>)	str(99)	'99'

```
>>> int(6.75)
6
>>> round(6.75)
```

2. Strings

- Sequence of characters (alphabets, numbers, special Characters) .
- Various ways to define a string.
  - ➢ single quotes (' ')

  Eg. 'Welcome'
  - ➢ double quotes (" ")

  Eg. "Welcome"
  - ➢ triple quotes(""" """) or (''' ''') - (either single or double)

  Eg. """ This is a paragraph. It is made up of multiple lines and sentences."""

  or

  ''' This is a paragraph. It is made up of multiple lines and sentences.'''
- Strings are immutable i.e. the contents of the string cannot be changed after it is created.

String in the Python shell with and without the print function

- >>>'Welcome' >>>"Welcome"

  'Welcome'
- >>>print('Welcome') >>>print("Welcome")

  Welcome
- When you evaluate a string in the Python shell without the print function, you can see the literal for the newline character, \n, embedded in the result,

```
>>> print("""This very long sentence extends
all the way to the next line.""")
This very long sentence extends
all the way to the next line.
```

Escape characters can also be used to embed a quote mark within a quoted string

Tell me “what’s your name?”
She replied: “ I didn’t mean to do it!”

```
>>> print(" She replied: “ I didn’t mean to do it! ")
 She replied: “ I didn’t mean to do it!
>>> print(" She replied: I didn’t mean to do it! ")
 She replied: I didn’t mean to do it!
>>> print("Tell me “what’s your name?”")
Tell me “what’s your name?”
>>> "Tell me “what’s your name?”"
'Tell me “what’s your name?”'
```

```
>>> print("Tell me \"what\'s your name?\"")
Tell me "what's your name?"
```

Operations

1. Indexing 2. Slicing 3. Concatenation 4. Repetitions 5. Membership 6. Comparison

1. Indexing:

- Characters can be accessed using indexing operations
  - Positive indexing helps in accessing the string from the beginning.
  - Negative indexing helps in accessing the string from the end.

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

  - Subscript 0 or –ve n(where n is length of the string) displays the first element.

Example: A[0] or A[-5] will display “H”

  - Subscript 1 or –ve (n-1) displays the second element.

Example: A[1] or A[-4] will display “E”

2. Slicing

To extract part of a string.
Specify 2 values separated by :
Eg. 2:5

2 – first position
5 - Ending position -1

```
>>> s="good morning"
```

>>> print(s[2:]) od morning	- Displaying items from 2nd till last.
>>> print(s[:4]) Good	- Displaying items from 1st position till 3rd .

- mystr = ‘language’

```
>>>Print(mystr[0])
 l
>>>Print(mystr[7])
e
>>>Print(mystr[-1])
e
>>>Print(mystr[1:5])
angu
>>>Print(mystr[4:-2])
  ua
```

Concatenation	**>>>print(s+"friends")** good morningfriends	-Adding and printing the characters of two strings.
Repetition	**>>>print(s*2)** good morninggood morning	Creates new strings, concatenating multiple copies of the same string
in, not in (membership operator)	>>> s="good morning" **>>>"m" in s True >>> "a" not in s True**	Using membership operators to check a particular character is in string or not. Returns true if present.

Escape sequences in string

Escape Sequence	Description	example
\n	new line	>>> print("hai \nhello") hai hello
\\	prints Backslash (\)	>>> print("hai\\hello") hai\hello
\'	prints Single quote (')	>>> print("'") '
\"	prints Double quote (")	>>>print("\"") "
\t	prints tab sapace	>>>print("hai\thello") hai hello
\a	ASCII Bell (BEL)	>>>print("\a")

Immutability:

- Python strings are "immutable" as they cannot be changed after they are created.
- Therefore [ ] operator cannot be used on the left side of an assignment.

operations	Example	output
element assignment	a="PYTHON" a[0]='x'	TypeError: 'str' object does not support element assignment
element deletion	a="PYTHON" del a[0]	TypeError: 'str' object doesn't support element deletion

```
>>> a ="program"
>>> a
'program'
>>> del a
>>> a
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    a
NameError: name 'a' is not defined
```

- mystr = 'language'

mystr[3] =' x'

Error

mystr = 'programming'

This is allowed

string built in functions and methods:

A **method** is a function that "belongs to" an object.

Syntax to access the method

Stringname.method()

a="happy birthday"

here, a is the string name.

	syntax	example	description
1	a.capitalize()	>>> a.capitalize() ' Happy birthday'	capitalize only the first letter in a string
2	a.upper()	>>> a.upper() 'HAPPY BIRTHDAY'	change string to upper case
3	a.lower()	>>> a.lower() ' happy birthday'	change string to lower case
4	a.title()	>>> a.title() ' Happy Birthday '	change string to title case i.e. first characters of all the words are capitalized.
5	a.swapcase()	>>> a.swapcase() 'HAPPY BIRTHDAY'	change lowercase characters to uppercase and vice versa
6	a.split()	>>> a.split() ['happy', 'birthday']	returns a list of words separated by space
7	a.center(width,"fillchar")	>>>a.center(19,"*") '***happy birthday***'	pads the string with the specified "fillchar" till the length is equal to "width"
8	a.count(substring)	>>> a.count('happy') 1	returns the number of occurences of substring
9	a.replace(old,new)	>>>a.replace('happy', 'wishyou happy') 'wishyou happy birthday'	replace all old substrings with new substrings

10	a.join(b)	>>> b="happy"
		>>> a="-"
		>>> a.join(b)
		'h-a-p-p-y'
			returns a string concatenated with the elements of an iterable. (Here "a" is the iterable)

11	a.isupper()	>>> a.isupper() False	checks whether all the case-based characters (letters) of the string are uppercase.
12	a.islower()	>>> a.islower() True	checks whether all the case-based characters (letters) of the string are lowercase.
13	a.isalpha()	>>> a.isalpha() False	checks whether the string consists of alphabetic characters only.

14	a.isalnum()	>>> a.isalnum() False	checks whether the string consists of alphanumeric characters.
15	a.isdigit()	>>> a.isdigit() False	checks whether the string consists of digits only.
16	a.isspace()	>>> a.isspace() False	checks whether the string consists of whitespace only.
17	a.istitle()	>>> a.istitle() False	checks whether string is title cased.
18	a.startswith(substring)	>>> a.startswith("h") True	checks whether string starts with substring
19	a.endswith(substring)	>>> a.endswith("y") True	checks whether the string ends with the substring
20	a.find(substring)	>>> a.find("happy") 0	returns index of substring, if it is found. Otherwise -1 is returned.
21	len(a)	>>>len(a) >>>14	Return the length of the string
22	min(a)	>>>min(a) >>>' '	Return the minimum character in the string
23	max(a)	max(a) >>>'y'	Return the maximum character in the string

- Printing without the Newline When you use the print function, it automatically prints a linefeed (\n) to cause the output to advance to the next line. If you don't want this to happen after the print function is finished, you can invoke the print function by passing a special argument end =

For example, the following code

```
1  print("AAA", end = ' ')
2  print("BBB", end = '')
3  print("CCC", end = '***')
4  print("DDD", end = '***')
```

displays

```
AAA BBBCCC***DDD***
```

Formatting Floating-Point Numbers

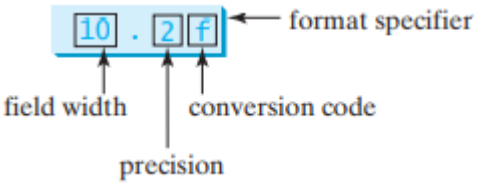
If the item is a float value, you can use the specifier to give the width and precision of the format in the form of **width.precisionf**.

Here, width specifies the width of the resulting string,
precision specifies the number of digits after the decimal point,
and f is called the conversion code, which sets the formatting for floating point numbers.

For example,

```
print(format(57.467657, "10.2f"))
print(format(12345678.923, "10.2f"))
print(format(57.4, "10.2f"))
print(format(57, "10.2f"))
```

where a square box ( ) denotes a blank space. Note that the decimal point is counted as one space.



displays

```
|←— 10 —→|
□□□□ 57.47
 123456782.92
□□□□□ 57.40
□□□□ 57.00
```

Formatting in Scientific Notation

If you change the conversion code from f to e, the number will be formatted in scientific notation. For example,

```
print(format(57.467657, "10.2e"))
print(format(0.0033923, "10.2e"))
print(format(57.4, "10.2e"))
print(format(57, "10.2e"))
```

displays

```
|←— 10 —→|
□□ 5.75e+01
□□ 3.39e-03
□□ 5.74e+01
□□ 5.70e+01
```

Formatting as a Percentage

You can use the conversion code % to format a number as a percentage. For example

```
print(format(0.53457, "10.2%"))
print(format(0.0033923, "10.2%"))
print(format(7.4, "10.2%"))
print(format(57, "10.2%"))
```

displays

```
|←— 10 —→|
□□□ 53.46%
□□□□□ 0.34%
□□□ 740.00%
□□ 5700.00%
```

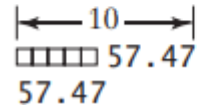
The format 10.2% causes the number to be multiplied by 100 and displayed with a % sign following it. The total width includes the % sign counted as one space.

Justifying Format

By default, the format of a number is right justified. You can put the symbol < in the format specifier to specify that the item be left-justified in the resulting format within the specified width. For example,

```
print(format(57.467657, "10.2f"))
print(format(57.467657, "<10.2f"))
```

displays

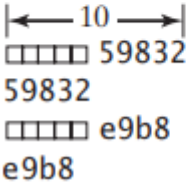


Formatting Integers

The conversion codes d, x, o, and b can be used to format an integer in decimal, hexadecimal, octal, or binary. You can specify a width for the conversion. For example

```
print(format(59832, "10d"))
print(format(59832, "<10d"))
print(format(59832, "10x"))
print(format(59832, "<10x"))
```

displays

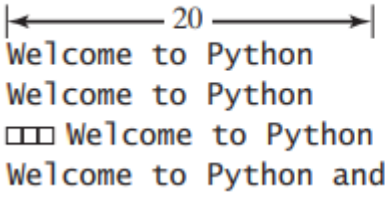


Formatting Strings

You can use the conversion code s to format a string with a specified width. For example

```
print(format("Welcome to Python", "20s"))
print(format("Welcome to Python", "<20s"))
print(format("Welcome to Python", ">20s"))
print(format("Welcome to Python and Java", ">20s"))
```

displays



The format specifier 20s specifies that the string is formatted within a width of 20. By default, a string is left justified. To right-justify it, put the symbol > in the format specifier. If the string is longer than the specified width, the width is automatically increased to fit the string.

Below table summarizes the format specifiers

TABLE 3.4 Frequently Used Specifiers

Specifier	*Format*
`"10.2f"`	Format the float item with width 10 and precision 2.
`"10.2e"`	Format the float item in scientific notation with width 10 and precision 2.
`"5d"`	Format the integer item in decimal with width 5.
`"5x"`	Format the integer item in hexadecimal with width 5.
`"5o"`	Format the integer item in octal with width 5.
`"5b"`	Format the integer item in binary with width 5.
`"10.2%"`	Format the number in decimal.
`"50s"`	Format the string item with width 50.
`"<10.2f"`	Left-justify the formatted item.
`">10.2f"`	Right-justify the formatted item.

LIST

- List is an ordered sequence of comma-separated items (values|elements) between square brackets[ ].
  - [5.87, 34, 'python', 'language']
- Items in the lists can be of different data types.
- Lists are mutable.

Operations on list:

1. Indexing 2. Slicing 3. Concatenation 4. Repetitions 5.Updation 6. Membership 7.Comparison

Creating a list	**>>>list1=["python", 7.79, 101, "hello"]** **>>>list2=["god",6.78,9]**	Creating the list with elements of different data types.
Indexing	**>>>print(list1[0])** python **>>> list1[2]** 101	❖ Accessing the item in the position 0 ❖ Accessing the item in the position 2
Slicing(ending position -1) Slice operator is used to extract part of a string, or some part of a list Python	**>>> print(list1[1:3])** [7.79, 101] **>>>print(list1[1:])** [7.79, 101, 'hello']	- Displaying items from 1st till 2nd. - Displaying items from 1st position till last.
Concatenation	**>>>print(list1+list2)** ['python', 7.79, 101, 'hello', 'god', 6.78, 9]	-Adding and printing the items of two lists.
Repetition	**>>> list2*3** **['god', 6.78, 9, 'god', 6.78, 9, 'god', 6.78, 9]**	Creates new strings, concatenating multiple copies of the same string
Updating the list	**>>> list1[2]=45** >>>print(list1) ['python', 7.79, 45, 'hello']	Updating the list using index value

Membership	>>> a=[2,3,4,5,6,7,8,9,10]
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in list. Otherwise returns false.

Comparison	>>> a=[2,3,4,5,6,7,8,9,10]
	>>>b=[2,3,4]
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

```
>>> a=[9,8,7,6,5,4]
```

- What is a[0:3] a[:4] a[1:] a[:] a[0:6:2] a[::-1] a[2:2] a[2,-2]

>>> a[2:2] [ ]	print an empty slice
>>> a[0:6:2] [9, 7, 5]	Slicing list values with step size 2.
>>> a[::-1] [4, 5, 6, 7, 8, 9]	Returns reverse of given list values

```
>>> a=[9,8,7,6,5,4]
>>> a[0:3]
[9, 8, 7]
>>> a[:4]
[9, 8, 7, 6]
>>> a[1:]
[8, 7, 6, 5, 4]
>>> a[:]
[9, 8, 7, 6, 5, 4]
```

- List Methods

syntax:

list name.method name(element/index/list)

	syntax	example	description

1	a.append(element)	>>> a=[1,2,3,4,5]
		>>> a.append(6)
		>>> print(a)
		[1, 2, 3, 4, 5, 6]
			Add an element to the end of the list

2	a.insert(index,element)	>>> a.insert(0,0)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6]
			Insert an item at the defined index

3	a.extend(b)	>>> b=[7,8,9]
		>>> a.extend(b)
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8,9]
			Add all elements of a list to the another list

4	a.index(element)	>>> a.index(8) 8	Returns the index of the first matched item

5	a.sort()	>>> a.sort()
		>>> print(a)
		[0, 1, 2, 3, 4, 5, 6, 7, 8]
			Sort items in a list in ascending order

6	a.reverse()	>>> a.reverse()
		>>> print(a)
		[8, 7, 6, 5, 4, 3, 2, 1, 0]
			Reverse the order of items in the list

7	a.pop()	>>> a.pop() 0	Removes and returns an element at the last element
8	a.pop(index)	>>> a.pop(0) 8	Remove the particular element and return it.

9	a.remove(element)	>>> a.remove(1)
		>>> print(a)
		[7, 6, 5, 4, 3, 2]
			Removes an item from the list

10	a.count(element)	>>> a.count(6) 1	Returns the count of number of items passed as an argument

11	a.copy()	>>> b=a.copy()
		>>> print(b)
		[7, 6, 5, 4, 3, 2]
			Returns a shallow copy of the list

12	len(list)	>>> len(a) 6	return the length of the length
13	min(list)	>>> min(a) 2	return the minimum element in a list
14	max(list)	>>> max(a) 7	return the maximum element in a list.

15	a.clear()	>>> a.clear()
		>>> print(a)
		[ ]
			Removes all items from the list.

16	del(a)	>>> del(a)
		>>> print(a)
		Error: name 'a' is not defined
			delete the entire list.

- Mutability - It is the ability to change after creating .

Example	description

>>> a=[1,2,3,4,5]
>>> a[0]=100
>>> print(a)
[100, 2, 3, 4, 5]
	changing single element

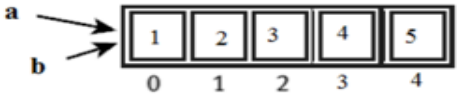
>>> a=[1,2,3,4,5]
>>> a[0:3]=[100,100,100]
>>> print(a)
[100, 100, 100, 4, 5]
	changing multiple element

>>> a=[1,2,3,4,5]
>>> a[0:3]=[ ]
>>> print(a)
[4, 5]
	The elements from a list can also be removed by assigning the empty list to them.

>>> a=[1,2,3,4,5]
>>> a[0:0]=[20,30,45]
>>> print(a)
[20,30,45,1, 2, 3, 4, 5]
	The elements can be inserted into a list by squeezing them into an empty slice at the desired location.

Aliasing(copying):

- *Creating a copy of a list is called aliasing.*
- *When you create a copy both list will be having same memory location.*
- *Changes in one list will affect another list.*



Example	Output:
a= [1, 2, 3 ,4 ,5] b=a print (b) a is b a[0]=100 print(a) print(b)	 [1, 2, 3, 4, 5] True [100,2,3,4,5] [100,2,3,4,5]

Cloning:

- To avoid the disadvantages of copying we are using cloning.
- Creating a copy of list of elements with two different memory locations is called cloning.
- Changes in one list will not affect another list.

clonning using Slicing
>>>a=[1,2,3,4,5] >>>b=a[:] >>>print(b) [1,2,3,4,5] >>>a is b False

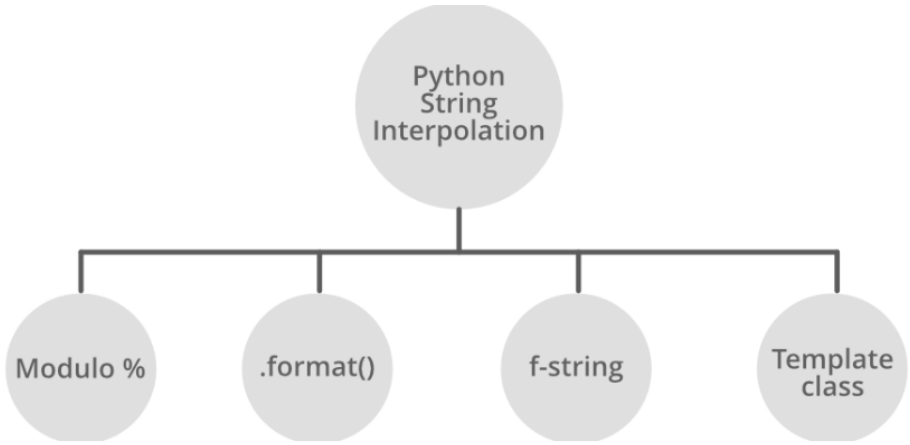
clonning using List( ) method
>>>a=[1,2,3,4,5] >>>b=list(a) >>>print(b) [1,2,3,4,5] >>>a is b false >>>a[0]=100 >>>print(a) >>>a=[100,2,3,4,5] >>>print(b) >>>b=[1,2,3,4,5]

clonning using copy() method

a=[1,2,3,4,5]
>>>b=a.copy()
>>> print(b)
[1, 2, 3, 4, 5]
>>> a is b
False

String Interpolation

String Interpolation is the process of inserting values of variables into placeholders in a string.
To print “hello <name> welcome to geeksforgeeks” where the <name> is the placeholder for the name of the user.



1. **% – Formatting**
   This is similar to printf style function in C
   n1 = 'Arun'
   n2 = 'OSP class'
   print("Hello % s Welcome to % s." % (n1, n2))

2. Str.format()
   In this method, instead of providing a % sign as a placeholder, we define the placeholder using curly braces { }. The values that we want to replace in the placeholder are passed as arguments to the format function.

```
print("Hello {} Welcome to  {} ".format(n1,n2))
```

3. f strings
   strings are perhaps the easiest way of performing string interpolation in python. f strings were introduced after python 3.6
   you do not have to define the placeholder and their values separately.
   you would write f before quotes

Using f strings you can directly interpolate values into the string by providing the values in curly brace inside the string.

```
print(f'Hello {n1} Welcome to  {n2} ")
```

4. Template
   A template is a class inside the string module.

create a template by using the $ sign as a placeholder and then use the substitute function inside the template class to replace it with placeholder values.

```
from string import Template

t = Template("Hello $n3 Welcome to $n4")

print(t.substitute(n3 = n1,n4 = n2))
```

---

4. Tuple

- A tuple is same as list, except that the elements are enclosed in parentheses instead of square brackets.

```
>>> t = (1, 2, 3, 2, 1)
```

- A tuple is an immutable list or a fixed list.
- Once a tuple has been created, you cannot add, delete, or replace elements in a tuple
- A list can be changed into a tuple, and vice versa.

```
>>>list(t)
[1, 2, 3, 2, 1]
```

```
>>> list1 = [1,2,3,4,5]
>>> tuple(list1)
(1, 2, 3, 4, 5)
```

```
>>> tuple(['a', 'b', 'c'])
('a', 'b', 'c')
```

Operations.

1. Indexing 2. Slicing 3. Concatenation 4. Repetitions
5. Membership 6.Comparison

Operations	examples	description
Creating a tuple	>>>a=(20,40,60,"apple","ball")	Creating the tuple with elements of different data types.
Indexing	>>>print(a[0]) 20 >>> a[2] 60	Accessing the item in the position 0 Accessing the item in the position 2
Slicing	>>>print(a[1:3]) (40,60)	Displaying items from 1st till 2nd.
Concatenation	>>> b=(2,4) >>>print(a+b) >>>(20,40,60,"apple","ball",2,4)	Adding tuple elements at the end of another tuple elements
Repetition	>>>print(b*2) >>>(2,4,2,4)	repeating the tuple in n no of times

Membership	>>> a=(2,3,4,5,6,7,8,9,10)
	>>> 5 in a
	True
	>>> 100 in a
	False
	>>> 2 not in a
	False
		Returns True if element is present in tuple. Otherwise returns false.

Comparison	>>> a=(2,3,4,5,6,7,8,9,10)
	>>>b=(2,3,4)
	>>> a==b
	False
	>>> a!=b
	True
		Returns True if all elements in both elements are same. Otherwise returns false

Methods

methods	example	description

a.index(tuple)	>>> a=(1,2,3,4,5)
	>>> a.index(5)
	4
		Returns the index of the first matched item.

a.count(tuple)	>>>a=(1,2,3,4,5) >>> a.count(3) 1	Returns the count of the given element.
len(tuple)	>>> len(a)	return the length of the
min(tuple)	>>> min(a) 1	return the minimum element in a tuple
max(tuple)	>>> max(a) 5	return the maximum element in a tuple
del(tuple)	>>> del(a)	Delete the entire tuple.

sum(tuple)

Tuple Assignment

- Python has a very powerful tuple assignment feature
- It allows a tuple of variables on the left of an assignment to be assigned values from a tuple on the right of the assignment.

```
>>> (a, b, c, d) = (1, 2, 3, 4)
```

- The left side is a tuple of variables; the right side is a tuple of values.
- Each value is assigned to its respective variable.
- The number of variables on the left and the number of values on the right have to be the same.

```
>>> (a, b, c, d) = (1, 2, 3)
ValueError: need more than 3 values to unpack
```

Swapping

- It is useful to swap the values of two variables.
- With conventional assignment statements, we have to use a temporary variable.
- For example, to swap a and b:

Swap two numbers
a=2;b=3
print(a,b)
temp = a
a = b
b = temp
print(a,b)

a=3 b=2

```
>>> (a,b)=(2,3)
>>> a
2
>>> b
3
>>> (a,b)=(b,a)
>>> a
3
>>> b
2
```

Tuple packing and unpacking

- In tuple unpacking, the values in a tuple on the right are 'unpacked' into the variables/names on the left:

```
>>> b = ("George", 25, 20000) # tuple packing
>>> (name, age, salary) = b # tuple unpacking
```

- >>> name
  'George'
- >>> age
  25
- >>> salary
  20000

Main function, lambda fun

Special variable __name__

mymath.py

```
1  def add (a , b):
2      return a + b
3
4  print(__name__)
5  if __name__ == "__main__":
6      print(add(10, 16))
```

test.py

```
1  import mymath
2
3  print(mymath.add(7, 6))
```

```
mymath
13

Process finished with exit code 0
```

Lambda function

Consider 3 functions. How to convert into lambda fn

```
def double(x):
    return x * 2
def add(x, y):
    return x + y
def product(x, y, z):
    return x * y * z
```

```
double = lambda x : x * 2
add = lambda x, y : x +y
product = lambda x, y, z : x * y * z
```

Where it is used

To pass fn as argument

To return fn as result

Map function

```
#filter, reduce and map
my_list = [2, 5, 8, 10, 9, 3]

a = map(lambda x : x * 2, my_list)
print(a)
```

```
6000
<map object at 0x00C4B670>
```

```
#filter, reduce and map
my_list = [2, 5, 8, 10, 9, 3]

a = map(lambda x : x * 2, my_list)
print(list(a))
```

```
[4, 10, 16, 20, 18, 6]
```

```
#filter, reduce and map
my_list = [2, 5, 8, 10, 9, 3]
my_list2 = [1, 4, 7, 8, 5, 1]

a = map(lambda x : x * 2, my_list)
print(list(a))
b = map(lambda x, y : x + y, my_list, my_list2)
print(list(b))
```

```
[3, 9, 15, 18, 14, 4]
```

```
d = filter(lambda x : True if x > 5 else False, my_list)
print(list(d))
```

```
e = reduce(lambda x , y : x + y, my_list)
print(e)
```

Combining two dictionaries with update

- Two lists can be combined using catenation, or append. This concept does not make sense for dictionary.
- however a somewhat similar operation is provided by the update method.
- This method takes as argument another dictionary. The values from the argument dictionary are copied into the receiver, possibly overwriting an existing entry.

```
dictone = {'abc':3, 'def':7, 'xyz': 9}
dicttwo = {'def':5, 'pdq': 4}
dictone.update(dicttwo)
print (dictone)
```

```
{'abc': 3, 'def': 5, 'xyz': 9, 'pdq': 4}
```

Making Copies

- Remember that Python uses reference semantics from assignment. If you simply assign one dictionary to a new variable, they end up referring to the same collection. A change to one will end up modifying both:

```
dictone = {'abc': 3, 'def': 7}
dicttwo = dictone
dicttwo['xyz'] = 12
print dictone
```

{' xyz': 12, 'abc': 3, 'def': 7}

- To make an independent copy of a dictionary you can use the method **copy**.

```
dictone = {'abc': 3, 'def': 7}
dicttow = dictone.copy()
dicttwo['xyz'] = 12
print dictone
{'abc': 3, 'def': 7}
```

- When a dictionary is passed as an argument the parameter is simply assigned the argument value. Hence both refer to the same value. A change to the dictionary inside the function will remain after the function returns.

Zip List Initialization

- The function dict takes a list of two-element tuples, and converts them into a dictionary with the first element in each tuple representing the key and the second representing the value:

```
x = dict([('name', 'fred'), ('age', 42), ('weight',175)])
x
{'name':'fred', 'age':42, 'weight':175}
```

- Many times you will find yourself with **a list of keys**, and separately with a **list of values**. Zip makes it easy to **convert this into a list of tuples**, which can then be used to create a dictionary.

```
keys = ['name','age','weight']
values = ['fred',42,175]
x= dict(zip(keys, values))
x
{'name':'fred', 'age':42, 'weight':175
```

- The following syntax shows how to zip together two lists of equal length into one list:

```
#define list a and list b
a = ['a', 'b', 'c']
b = [1, 2, 3]
#zip the two lists together into one list
list(zip(a, b))
[('a', 1), ('b', 2), ('c', 3)]
```

- The following syntax shows how to zip together two lists of equal length into a dictionary:

```
#define list of keys and list of values
keys = ['a', 'b', 'c']
values = [1, 2, 3]
#zip the two lists together into one dictionary
dict(zip(keys, values))
```

{'a': 1, 'b': 2, 'c': 3}

- If your two lists have unequal length, zip() will truncate to the length of the shortest list:

  #define list a and list b

  ```
  a = ['a', 'b', 'c', 'd']

  b = [1, 2, 3]

  #zip the two lists together into one list

  list(zip(a, b))

  [('a', 1), ('b', 2), ('c', 3)]
  ```

  If you'd like to prevent zip() from truncating to the length of the shortest list, you can instead use the zip_longest() function from **the itertools** library.

  By default, this function fills in a value of "None" for missing values:

  ```
  from itertools import zip_longest

  #define list a and list b

  a = ['a', 'b', 'c', 'd']

  b = [1, 2, 3]


  #zip the two lists together without truncating to length of shortest list

  list(zip_longest(a, b))

  [('a', 1), ('b', 2), ('c', 3), ('d', None)]
  ```

  However, you can use the fillvalue argument to specify a different fill value to use:

  ```
  #define list a and list b

  a = ['a', 'b', 'c', 'd']

  b = [1, 2, 3]

  #zip the two lists together, using fill value of '0'

  list(zip_longest(a, b, fillvalue=0))
  ```

  [('a', 1), ('b', 2), ('c', 3), ('d', 0)]

UNIT V

Class objects

```
class student:

    def __init__(self,n,r,m):
        self.name=n
        self.rno=r
        self.mark=m

    def display(self):
        print("NAME: ",self.name)
        print("RNO: ",self.rno)
        print("MARKS: ",self.mark)

st1=student("john",1,90)
st2=student("ravi",2,90)

st1.display()
st2.display()
```

To add 100 students

Multile objects

Using list

Store list of name and rollno

Have a list of objects

```
File Edit Format Run Options Window Help
class student:

    def __init__(self,n,r,m):
        self.name=n
        self.rno=r
        self.mark=m

    def display(self):
        print("NAME: ",self.name)
        print("RNO: ",self.rno)
        print("MARKS: ",self.mark)

nl=["A","B","C","D"]
rl=[1,2,3,4]
ml=[90,100,90,100]
objl=[]

for i in range(0,4):
    objl.append(student(nl[i],rl[i],ml[i]))

for i in range(0,4):
    objl[i].display()

print("\n")
```

t0 give new line

```
Python Shell
# st1=student("john",1,90)
#st1.display()
```

```
class Student(object):
    def __init__(self):
        print("Super Class Student")
        self.RegNo=0
        self.RollNo=0
        self.Name=""
        self.Class=""
        self.Section=""

    def EnterData(self):
        self.RegNo=int(input("Enter Reg No:"))
        self.RollNo=int(input("Enter Roll No:"))
        self.Name=input("Enter Name:")
        self.Class=input("Enter Class:")
        self.Section=input("Enter Section:")

    def Display(self):
        print("Reg No :",self.RegNo)
        print("Roll No:",self.RollNo)
        print("Name   :",self.Name)
        print("Class  :",self.Class)
        print("Section:",self.Section)
```

```

L=[]
for i in range(2):
    a=Student()
    a.EnterData()
    L.append(a)

for i in range(2):
    print("----------------------")
    L[i].Display()
    print("----------------------")
```

Exception Handling

Defn

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's execution.

What is the purpose of exception handling in Python?

To recover the program from an abnormal termination.

Built-in Exceptions in Python

Python has many **built-in exceptions** that enable the program to run without interruption and give the output. These exceptions are given below:

A list of common exceptions that can be thrown from a standard Python program is given below.

1. **ZeroDivisionError:** Occurs when a number is divided by zero.
2. **NameError:** It occurs when a name is not found. It may be local or global.
3. **IndentationError:** If incorrect indentation is given.
4. **IOError:** It occurs when Input Output operation fails.
5. **EOFError:** It occurs when the end of the file is reached, and yet operations are being performed.

Example

```
a = int(input("Enter a:"))
b = int(input("Enter b:"))
c = a/b
print("a/b = %d" %c)

#other code:
print("Hi I am other part of the program")
```

Output:

```
Enter a:10
Enter b:0
Traceback (most recent call last):
  File "exception-test.py", line 3, in <module>
    c = a/b;
ZeroDivisionError: division by zero
```

Handling an exception -- Simple syntax

```
try:
   # suspicious code
except:
    #   code to handle runtime error
```

1. Place the suspicious code(critical code/sensitive code) in a **try:** block.
2. After the **try:** block, include an **except:** block, to handle the problem

```
a = int(input("enter a  "))
b = int(input("enter b  "))
try:
    c = a/b
except:
    print("Can't divide with zero")
```

Output:

```
Enter a:10
Enter b:0
Can't divide with zero
```

except: block without any exception name catches all the exceptions that occur. Using this kind of try-except statement is not considered as a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem.

Other clauses

else: block

code to execute if there is no exception

finally: block

code to execute always.(if exception raises or not)

Example

This example tries to open a file where you do not have write permission, so it raises an exception −

```
try:
   fp = open("testfile", "r")
   fp.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t write, Access mode is read")
else:
   print "Written content in the file successfully"
```

This produces the following result −

Error: can\'t write, Access mode is read

Handling Multiple Exceptions

1**) use the single *except* statement** −

```
try:
    #suspicious code
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

2) Use Multiple except statement

```
try:
    #suspicious code
   ......................
except Exception1:
     execute this block if exception1 raises
except Exception2:
    execute this block if exception2 raises
except Exception3:
      execute this block if exception3 raises

else:
   If there is no exception then execute this block.
```

_____

Example to handle error in List

```
a = [1, 2, 3]
try:
    print ("Second element = ",(a[1]))

    # Throws error since there are only 3 elements in array
    print ("Fourth element = ", (a[3]))

except:
    print ("An error occurred")
```

Example for all the four clauses to handle exception

```
a = int(input("enter the value of a  "))
b = int(input("enter the value of b  "))
try:
    c = a/b
except:
    print("Zero division error")
else:
    print("c = ",c)
finally:
    print("successfully terminated")
```

Output

```
enter the value of a  5
enter the value of b  3
c =  1.6666666666666667
successfully terminated
```

```
enter the value of a  5
enter the value of b  0
Zero division error
successfully terminated
```

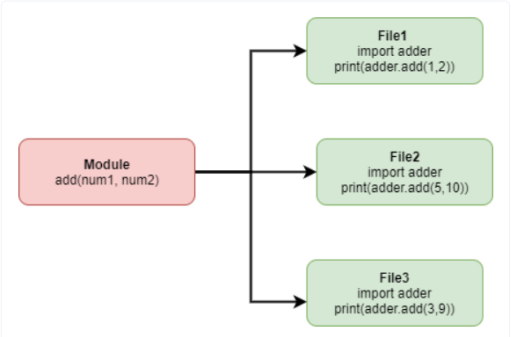
Print the type of error

```
a = int(input("enter a  "))
b = int(input("enter b  "))
try:
    print (c = a/b)
except Exception as e:
    print("Can't divide with zero  :", e)
```

```
enter a  5
enter b  0
Can't divide with zero  : division by zero
```

MODULES

A Python module is a Python file that contains classes, functions, or variables that you'd like to include in your application. A common practice in advanced Python applications is reusable functions or classes combined in a file, and then imported in other parts of the program.



This is referred to as a Python module. This module can then be imported and the variables and functions can be reused multiple times without declaring or creating them every time.

To Create a Python Module?

1. Create a File Containing a Method

We'll first create a basic method that adds two numbers which it accepts as parameters.

```
name = "AskPython"

def add(num1, num2):
    return num1 + num2
```

Save the above code as adder.py and we'll move on to the next step.

2. Create the Main File to Import the Module

Now, this is exactly the same as importing any other module. The only difference is that the module file is locally located instead of being in the system path. Syntax:

import module_name

module_name.function_name(variable)

```
import adder

nums = adder.add(5, 10)

print(nums)
```

3. Importing Only One Function

Suppose in our module, we had multiple functions performing multiple tasks. But in the program that we are importing the module, we needed only one of those functions. Importing the entire module would be unnecessary. In this case, we can make use of from and import together. Python allows us to use the function as if it was nativ-e to the file without having to reference it with the module name.

```
from adder import add

nums = add(5, 10)

print(nums)
```

4. Using Variables From Our Module

You might have noticed the two variables in our module. We added those to demonstrate how variables can be directly imported from the module with their values intact.

```
import adder
```

```
nums = adder.add(5, 10)

print(nums)

print(adder.name)
```

There are **four ways to import a module** in our program, they are

Import: It is simplest and most common way to use modules in our code.
Example:
`import math`
`x=math.pi`
`print("The value of pi is", x)`
Output: The value of pi is 3.141592653589793
	from import : It is used to get a specific function in the code instead of complete file.
	Example:
	`from math import pi`
	`x=pi`
	`print("The value of pi is", x)`
	Output: The value of pi is 3.141592653589793

import with renaming:
We can import a module by renaming the module as our wish.
Example:
`import math as m`
`x=m.pi`
`print("The value of pi is", x)`
Output: The value of pi is 3.141592653589793
	import all:
	We can import all names(definitions) form a module using *
	Example:
	`from math import *`
	`x=pi`
	`print("The value of pi is", x)`
	Output: The value of pi is 3.141592653589793

File Handling

File handling: Need for a data file, Types of file: Text files, Binary files and CSV (Comma separated values) files.

Text File: Basic operations on a text file: Open (filename – absolute or relative path, mode), Close a text file, Reading and Manipulation of data from a text file, Appending data into a text file, standard input / output and error streams, relative and absolute paths.

Binary File: Basic operations on a binary file: Open (filename – absolute or relative path, mode), Close a binary file, Pickle Module – methods load and dump; Read, Write/Create, Search, Append and Update operations in a binary file.

CSV File: Import csv module, functions – Open, Close a csv file, Read from a csv file and Write into a csv file using csv.reader ( ) and csv.writerow( ).

Fundamental operations of all types of files

Open a file

Read or write - Performing operation

Close the file

Opening a file

All files must first be opened before they can be read from or written to. In Python, when a file is (successfully) opened, a file object is created that provides methods for accessing the file.

Python provides an open() function that accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

Syntax:

file object = open(<file-name>, <access-mode>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

Mode	*Description*
"r"	Opens a file for reading.
"w"	Opens a new file for writing. If the file already exists, its old contents are destroyed.
"a"	Opens a file for appending data from the end of the file.
"rb"	Opens a file for reading binary data.
"wb"	Opens a file for writing binary data.

SN Access mode Description

1 r It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.

2 rb It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.

3 r+ It opens the file to read and write both. The file pointer exists at the beginning of the file.

4 rb+ It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.

5 w It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.

6 wb It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file.

7 w+ It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.

8 wb+ It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.

9 a It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.

10 ab It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.

11 a+ It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.

12 ab+ It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

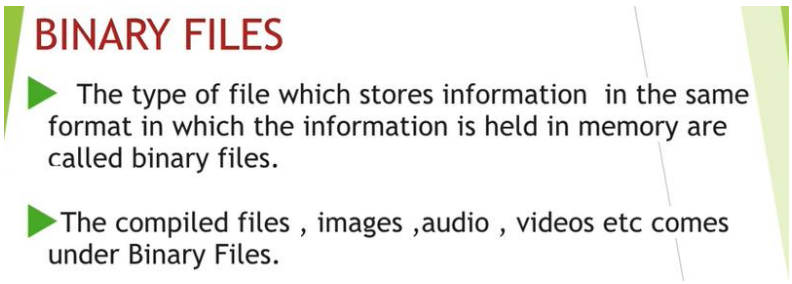
Example

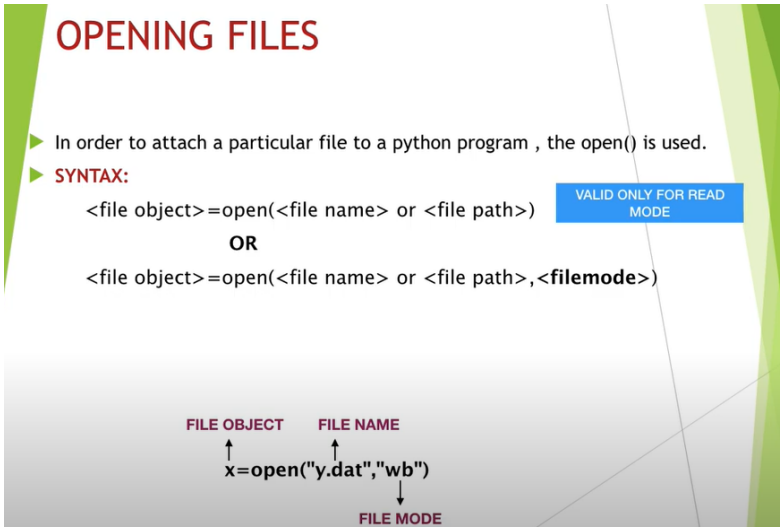
```
#opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
   print("file is opened successfully")
```

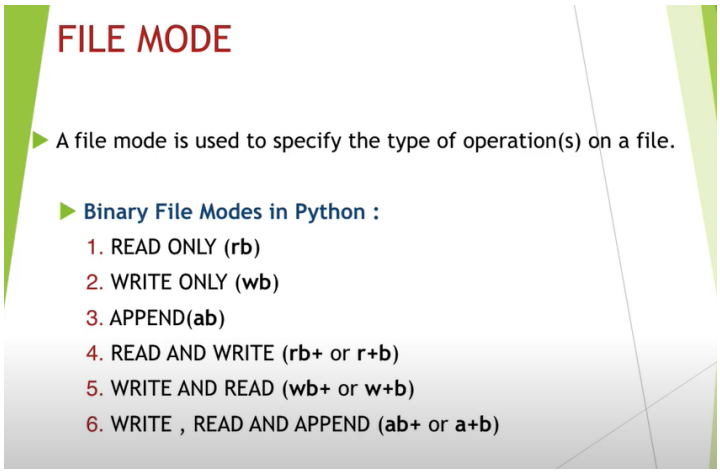
Output:

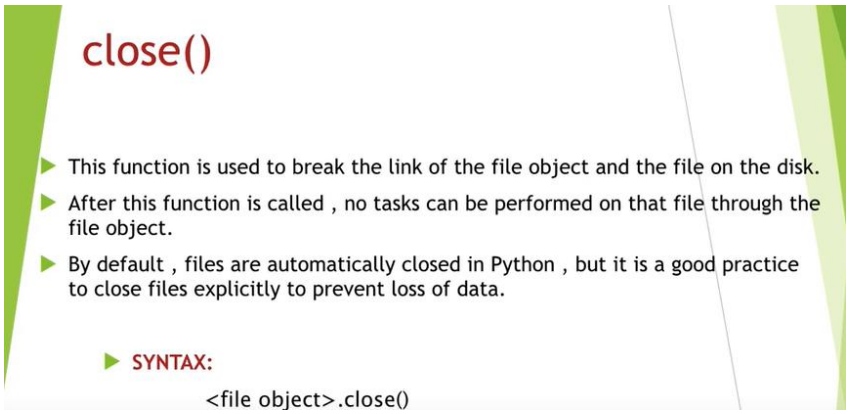
```
<class '_io.TextIOWrapper'>
file is opened successfully
```

In the above code, we have passed **filename** as a first argument and opened file in read mode as we mentioned **r** as the second argument. The **fileptr** holds the file object and if the file is opened successfully, it will execute the print statement

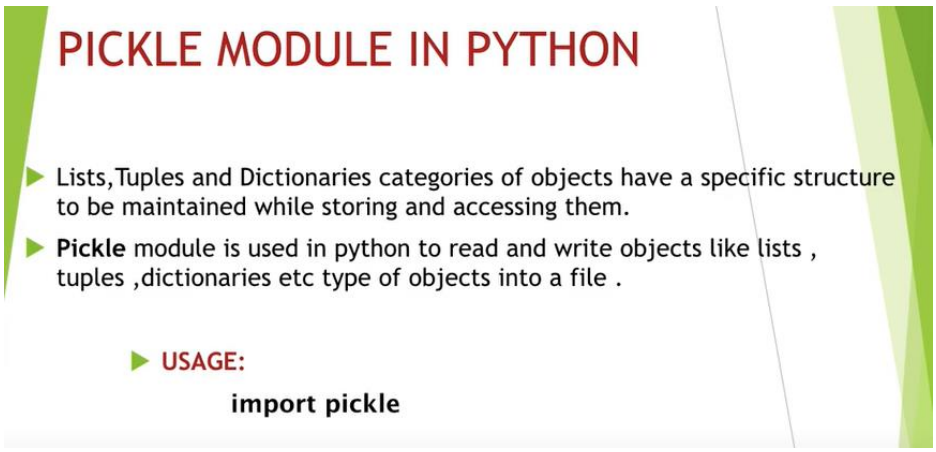


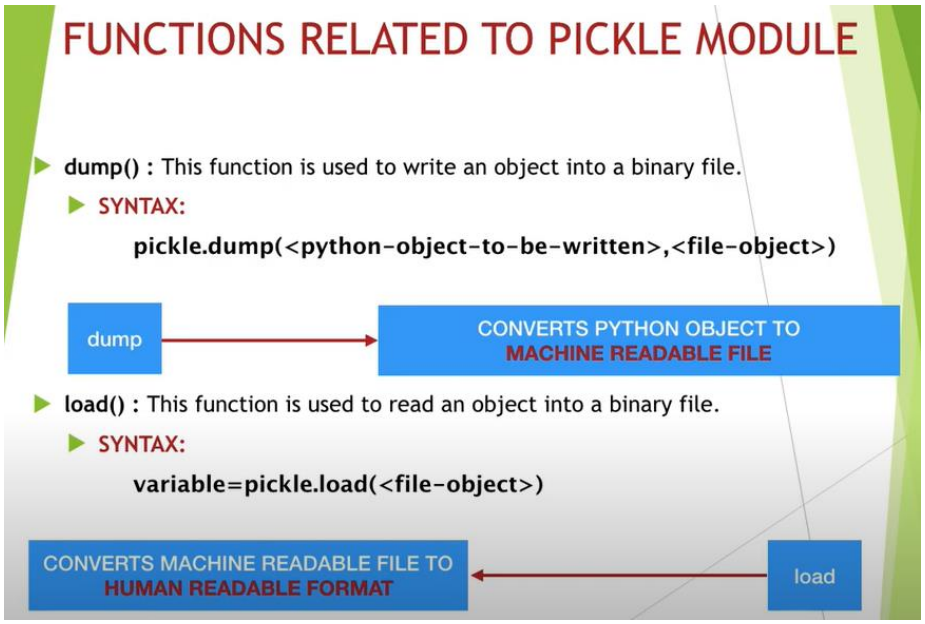






For working with binary files we need a module called as pickle




FUNCTIONS RELATED TO PICKLE MODULE
dump() : This function is used to write an object into a binary file.
SYNTAX:
pickle.dump(<python-object-to-be-written>,<file-object>)
dump
CONVERTS PYTHON OBJECT TO
MACHINE READABLE FILE
load() : This function is used to read an object into a binary file.
SYNTAX:
variable=pickle.load(<file-object>)
CONVERTS MACHINE READABLE FILE TO
HUMAN READABLE FORMAT
load