

Python

Python is a programming language, which is an easy to learn, powerful programming language. It has effective and efficient high-level data structure. Also, Python is high level, interpreted, multi-paradigm and general purpose programming language.

History of Python

- Creator of Python is Guido Van Rossom. In 1991 python code was released (version 0.9.0) at CWI Netherland.
- In 1994 Python1.0 was released with new features like lambda, map, filter and reduce.
- After this Python2.x added some more features like list comprehensions, garbage collection system.
- December 03, 2008, Python 3.x was released and in this version, all fundamental flow of python programming was changed.

Features of Python

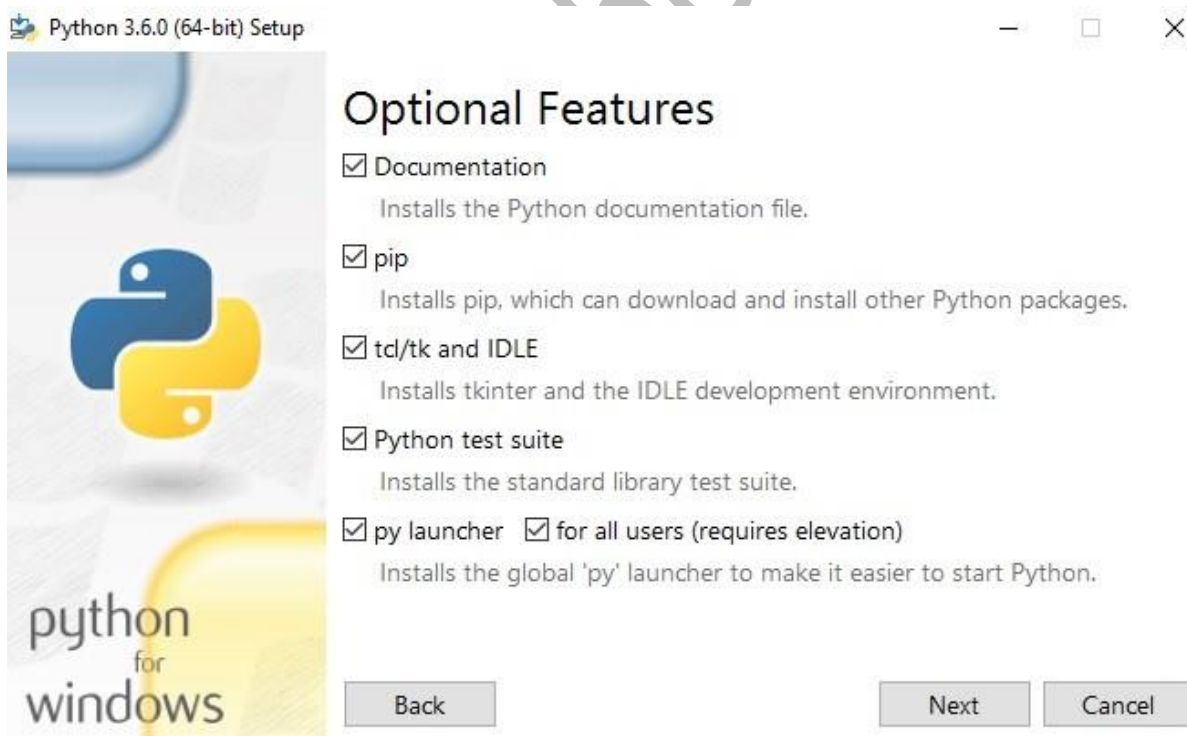
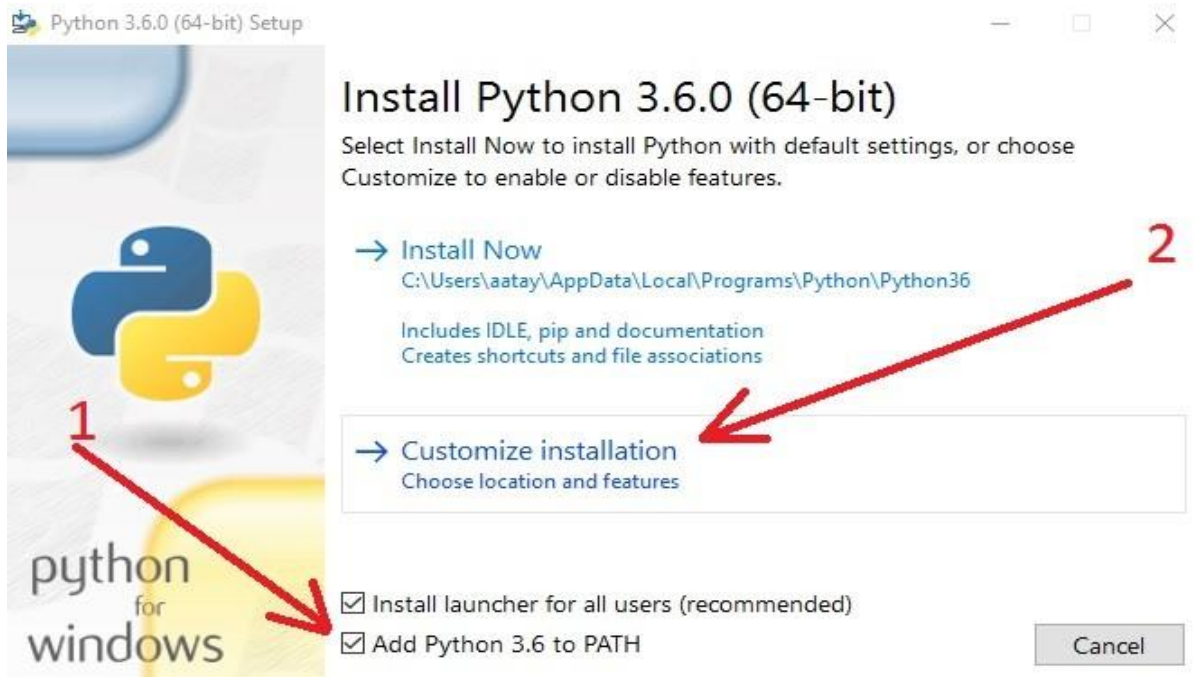
1. Easy to use
2. Expressive language
3. Interpreter Programming Language
4. Support Multi Paradigm Programming Approach
5. Free and Open Source
6. Large Standard Library
7. Cross Platform and Platform Independent
8. Integrated
9. GUI Programming Support

Application of Python

- Web Applications
- Desktop GUI Application
- Software Development
- Scientific and Numeric
- Business Application
- Console Base Application
- Audio and Video Base Application Development
- Image Processing Application
- Data Analysis
- Machine Learning
- Artificial Intelligence

Installation of Python

Download Python3.7 setup from www.python.org/download



Python 3.6.0 (64-bit) Setup



Optional Features

- ☒ Documentation
Installs the Python documentation file.
- ☒ pip
Installs pip, which can download and install other Python packages.
- ☒ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.
- ☒ Python test suite
Installs the standard library test suite.
- ☒ py launcher ☒ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

Back

Next

Cancel

Python 3.6.0 (64-bit) Setup



Advanced Options

- ☒ Install for all users
- ☒ Associate files with Python (requires the py launcher)
- ☒ Create shortcuts for installed applications
- ☒ Add Python to environment variables
- ☒ Precompile standard library
- ☐ Download debugging symbols
- ☐ Download debug binaries (requires VS 2015 or later)

Customize install location

C:\python

Browse

Back

Install

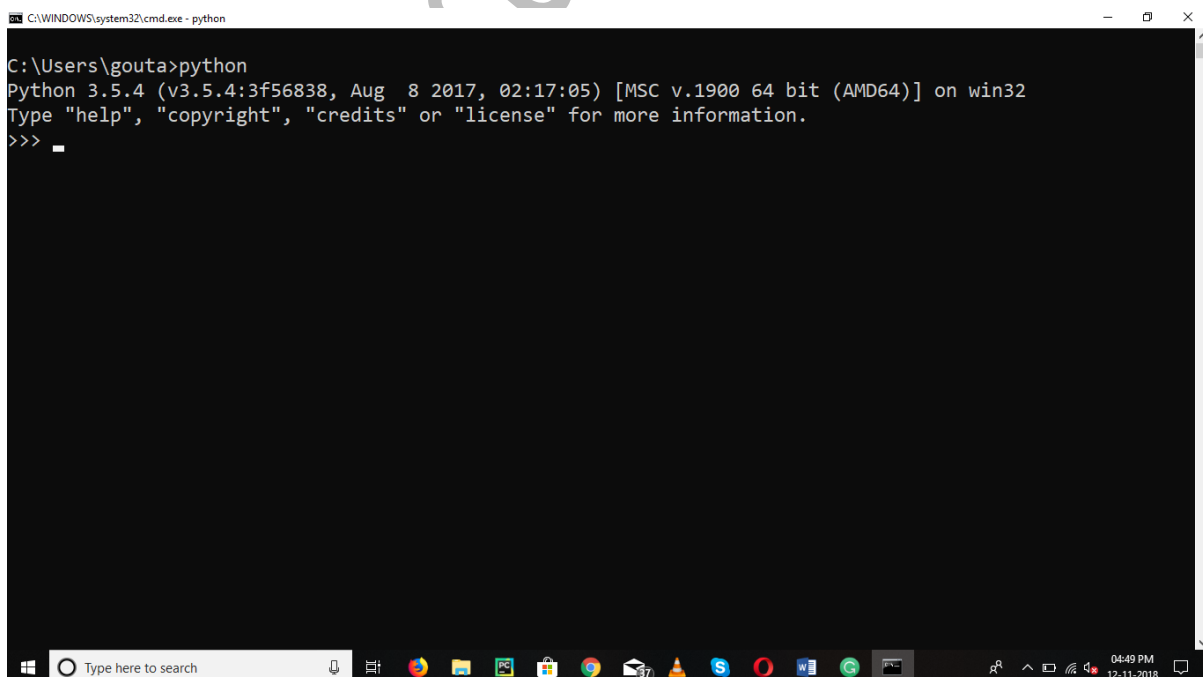
Cancel



**** Python is inbuilt in Linux and Mac Book**

This is a complete process of install Python. Now we need to check that python setup is installed properly or not?

First, open Command Prompt of window or terminal for Linux or Mac Book. Run **python** command on cmd. For Linux and Mac Book write **python3** because in both OS python is inbuilt so both version Python2.x and Python3.x is install already.

The image shows a Windows Command Prompt window. The title bar reads 'C:\WINDOWS\system32\cmd.exe - python'. The command prompt shows the user's location as 'C:\Users\gouta>' and the command 'python' has been entered. The output is: 'Python 3.5.4 (v3.5.4:3f56838, Aug 8 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)] on win32'. It then prompts the user to 'Type "help", "copyright", "credits" or "license" for more information.' and shows the prompt '>>>' with a cursor. The Windows taskbar is visible at the bottom with the search bar and various application icons.

If this message will show then our installation process in complete.

Python Keywords

Keywords are the reserved words in Python, which cannot use as variables name, function name or any other identifiers like class name etc.

Python is a case sensitive programming language. There are total 33 keywords in Python which are

False	Class	finally	is	return
None	Continue	for	lambda	try
True	Def	from	nonlocal	while
And	Del	global	not	with
As	Elif	if	or	yield
Assert	Else	import	pass	
break	Except	in	raise	

Rule of Variables in Python

1. Name of a variable can contain only alpha-numeric characters (a-z, A-Z, 0-9) and underscores (_).
2. Name of a variable must start with a letter (a-z or A-Z) or underscore (_).
3. Variable name cannot start with a number or a special character.
4. As we know that python is a case sensitive programming language so age, Age and AGE all are different variables.

Data Types in Python

There are six data types in python which are:-

1. Number
2. String
3. List
4. Tuple
5. Dictionary
6. Set

Number: -

There are three numeric types in Python:-

1. int :- it store Integer numbers like 1,2,4
2. float :- it store float numbers like 1.2,0.3
3. complex :- it store complex like 4+6j

String:-

Collection of character is called String. In Python String is defined between ' ', " ", "''", and "" ". String are immutable so we cannot change any element or item in string.

Python is a zero index programming.

```
a = "hello"
```

h	e	l	l	o
0	1	2	3	4

```
print(a[2]) l  
print(a[1]) e
```

List:-

The most popular data type of python is list. List is written between square brackets [] and every element of list is comma separated.

The main advantage of list is we can add any datatype in it means we can add string, number, list, tuple, dictionary in the given list

```
a = [1, 2, 3, "hello", [1, 23]]  
print(a[1]) 2
```

List in list is called nested list.

List are mutable so we can change element of list and also we can change size of list.

Tuple:-

The tuple is a sequence of python objects like list. The difference between Tuple and List are tuple is immutable that cannot change like list and tuple use parentheses (), where list use square brackets []

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

Dictionary:-

Dictionary is a collection of unordered and changeable python objects. Each element has a key and a value. Dictionary are written between curly brackets.

Each Element is separated by comma and every key and its value is separate by colon (:)

`d = {key1: value1, key2: value2}`

Set:-

A Set is the collection of unordered and unindexed datatype and Set is written between curly brackets.

Generally Set is used for performing the mathematical operation like union, difference, intersection.

And we show that List support duplicate items but set cannot support duplicate items.

`a = {1, 2, 3, 4, 9, 4, 6, 7}`

Assignment:-

1. Which is the correct operator for power(xy)?

- a) X^y b) $X**y$ c) $X^{^y}$ d) None of the mentioned

2. Which one of these is floor division?

- a) / b) // c) % d) None of the mentioned

3. What is the order of precedence in python?

- i) Parentheses ii) Exponential iii) Multiplication iv) Division
v) Addition vi) Subtraction

- a) i,ii,iii,iv,v,vi b) ii,i,iii,iv,v,vi c) ii,i,iv,iii,v,vi d) i,ii,iii,iv,vi,v

4. What is the answer to this expression, $22 \% 3$ is?

- a) 7 b) 1 c) 0 d) 5

5. Mathematical operations can be performed on a string. State whether true or false.

- a) True b) False

6. Operators with the same precedence are evaluated in which manner?

- a) Left to Right b) Right to Left c) Can't say d) None of the mentioned

7. What is the output of this expression, $3*1**3$?

- a) 27 b) 9 c) 3 d) 1

8. Which one of the following has the same precedence level?

- a) Addition and Subtraction b) Multiplication, Division and Addition
c) Multiplication, Division, Addition and Subtraction d) Addition and Multiplication

9. The expression `Int(x)` implies that the variable x is converted to integer. State whether true or false.

- a) True b) False

Operators in Python

Operators are those special symbols that perform arithmetic or logical operation. The values use by operator is call operates.

```
>> 1+7
8
```

Where 1 and 7 are operates.

There are seven operators in Python:-

1. Arithmetic Operator
2. Relational Operator
3. Assignment Operator
4. Logical Operator
5. Bitwise Operator
6. Membership Operator
7. Identity Operator

Arithmetic Operator

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

+, -, *, /, **, //, % are arithmetic operators

Comparison operators

Comparison operators are used to compare values. It either returns **True** or **False** according to the condition.

<, >, <=, >=, != are comparison operator.

Logical operators

Logical operators are the AND, OR, NOT operators.

Operator	Meaning	Example
AND	True if both the operands are true	x AND y
OR	True if either of the operands is true	x OR y
NOT	True if operand is false	NOT x

Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

Operator	Meaning	Example
&	Bitwise AND	x & y
	Bitwise OR	x y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise Right Shift	x>>2
<<	Bitwise Left Shift	x<<2

Assignment operators

Assignment operators are used in Python to assign values to variables
Example of Assignment operators are =, +=, -=, *=, /= etc.

Identity operators

`is` and `is not` are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Membership operators

`in` and `not in` are the membership operators in Python. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

www.wscubetech.com

Conditional Statements

When we need to decision making required in our programme then we use a conditional statement, in the conditional statement a particular code or statement execute when given condition or a certain condition is True.

There will be various situations while writing a program when you will have to take care of different possible conditions that might arise while execution of the program.

There are three types of conditional statement

1. If conditional statement
2. If-else conditional statement
3. If-elif-else conditional statement

Python supports the usual logical conditions from mathematics:

- Equals: $a == b$
- Not Equals: $a != b$
- Less than: $a < b$
- Less than or equal to: $a <= b$
- Greater than: $a > b$
- Greater than or equal to: $a >= b$

If conditional statement

If we have only one condition to check then we use if statement. Suppose if I want to compare two numbers then we will use if conditional statement.

Syntax:-

```
if (condition):  
    "Body of if statement"
```

NOTE

1. In python, there are no curly brackets. So instead of curly brackets, we use colon (:)
2. Because of the colon, now if we want to write code in the block of if-statement then we need to indent our code. It may be four space indentation or one tab indentation. This indentation makes our code beautiful. And if we want to come out from if-statement then we remove this indentation.

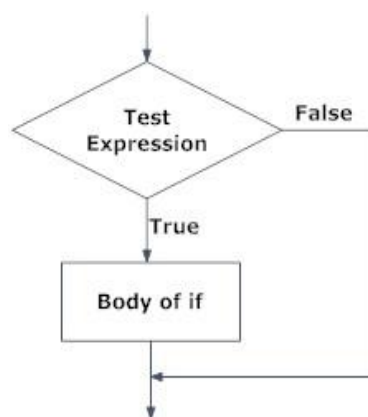


Fig: Operation of if statement

If-else statement

The if-else statement evaluates test expression and will execute body of if only when test condition is **True**.

If the condition is False, body of else is executed. Indentation is used to separate the blocks.

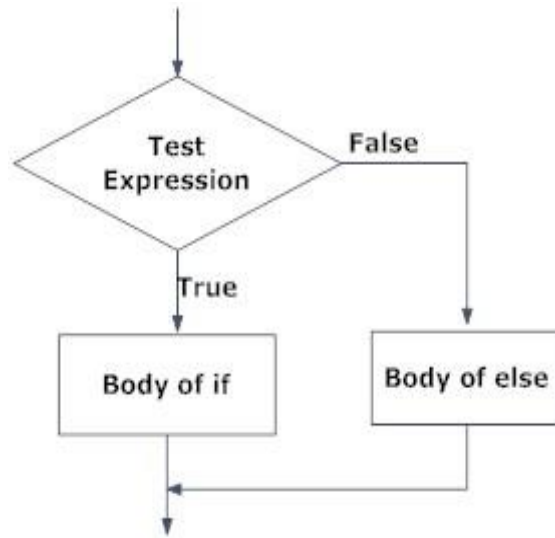


Fig: Operation of if...else statement

Syntax:-

```

if (condition):
    "Body of if statement"
else:
    "Body of else statement"
  
```

Generally we use if-else statements when we have two conditional statement.

if-elif-else statement

The elif is short for else if. It allows us to check for multiple expressions.

If the condition for if is False, it checks the condition of the next elif block and so on.

If all the conditions are False, body of else is executed.

Only one block among the several if...elif...else blocks is executed according to the condition.

The **if** block can have only one else block. But it can have multiple elif blocks.

Syntax:-

```

if (condition01):
    "Body of if statement"
elif (condition02):
    "Body of elif statement"
elif (condition03):
    "Body of elif statement"
else:
    "Body of else statement"
  
```

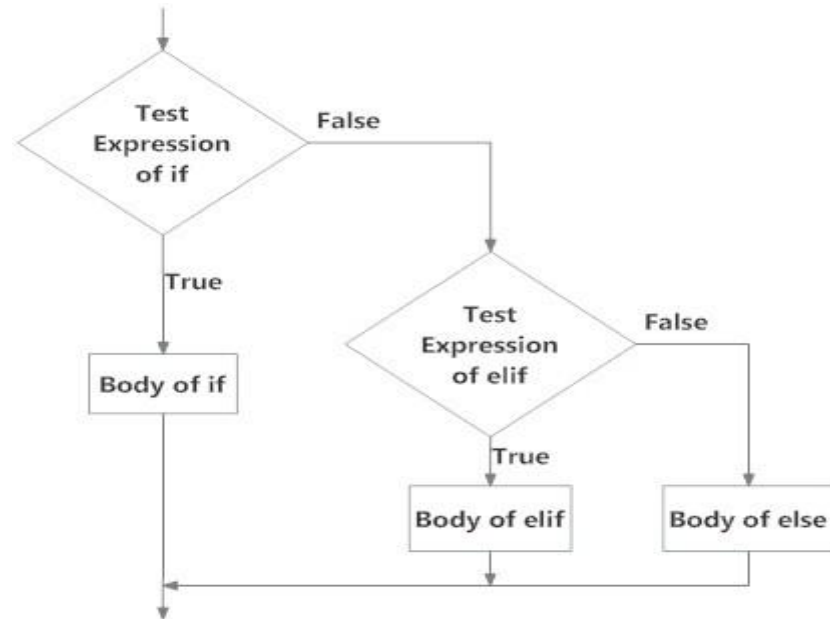


Fig: Operation of if...elif...else statement

Python Nested if statements

We can have if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.

Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

```

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
  
```

Assignment

1. The following syntax is correct for if conditional statement.

```

if condition
    code
end
  
```

a) True b) False

2. If expression.

The expression can be of which type?

- a) True b) Any number c) Any string d) All of the mentioned

3. What error does the if condition gives if not terminated with end statement?

- a) Syntax error b) Unexpected end
c) Expecting keyword end d) All of the mentioned

4. What is the output of the following?

```
if 1<2
```

```
  print "one is less than two"
```

```
end
```

- a) One is less than two b) Syntax error
c) 1<2 d) None of the mentioned

5. If statement inside if statement is called Nested if statements.

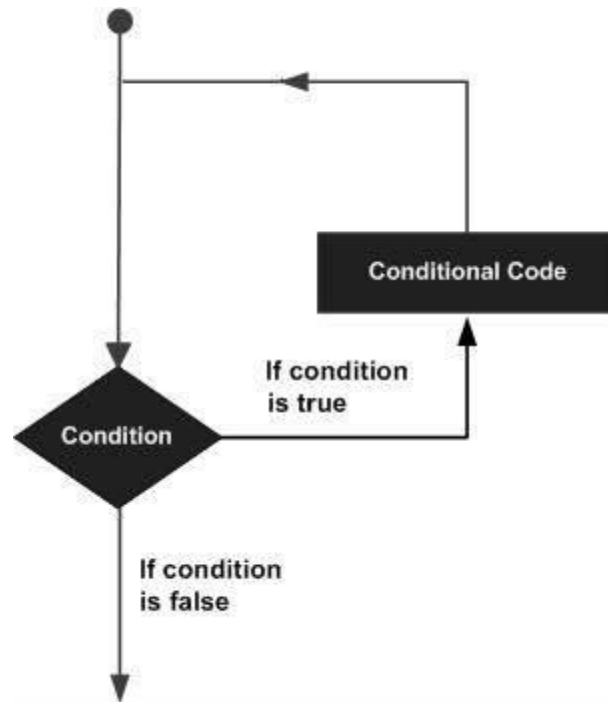
- a) True b) False

Loop Statement

Sometimes we need to execute a particular block of code several times then we use loop statement in loop statement we can execute a block of code multiple times

There are two types of loop statement in python

1. while loop
2. for loop



While loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is True.

We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax:-

```
while condition:  
    Body of while  
    Condition control statement
```

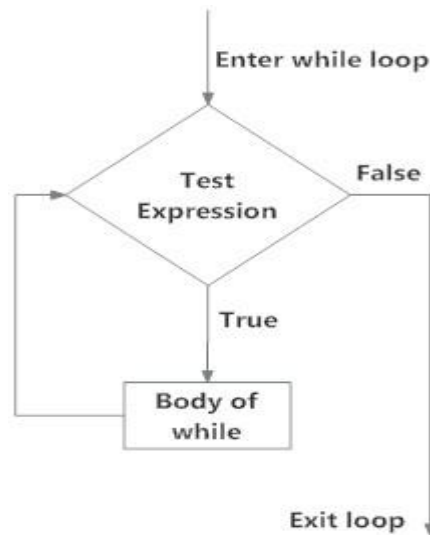



Fig: operation of while loop

```

n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i+1 # update counter

# print the sum
print("The sum is", sum)
  
```

While loop with else

The else part is executed if the condition in the while loop evaluates to False. The while loop can be terminated with a break statement. In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.

```

counter = 0
while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
  
```

For loop:-

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax:-

```
for val in sequence:  
    Body of for
```

Here, val is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

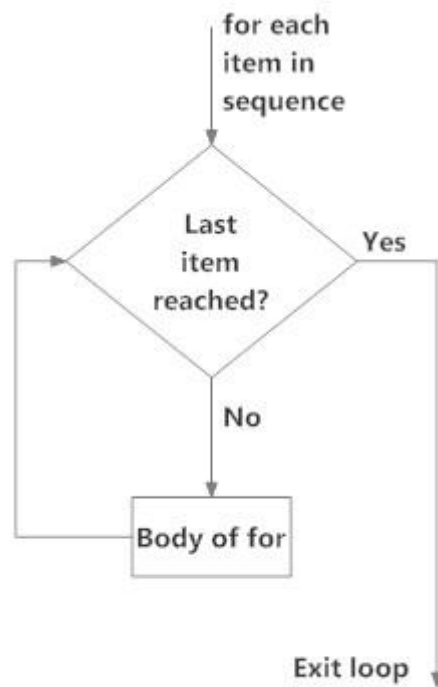


Fig: operation of for loop

The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as range(start, stop, step size). step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

Syntax:-

```
for a in range(10):  
    print(a)
```

for loop with else

A for loop can have an optional else block as well. The else part is executed if the items in the sequence used in for loop exhausts.

break statement can be used to stop a for loop. In such case, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

Nested Loop

If there is loop in a loop is called Nested Loop. It may be for in for or while in for or Vice-Versa.

```
words= ["Apple", "Banana", "Car", "Dolphin" ]
for word in words:
    #This loop is fetching word from the list
    print ("The following lines will print each letters of "+word)
    for letter in word:
        #This loop is fetching letter for the word
        print (letter)
    print("") #This print is used to print a blank line
```

Assignment

1. What is the output of the following?

```
x = ['ab', 'cd']
```

```
for i in x:
```

```
    i.upper()
```

```
print(x)
```

a) ['ab', 'cd']. b) ['AB', 'CD']. c) [None, None]. d) none of the mentioned

2. What is the output of the following?

```
x = ['ab', 'cd']
```

```
for i in x:
```

```
    x.append(i.upper())
```

```
print(x)
```

a) ['AB', 'CD']. b) ['ab', 'cd', 'AB', 'CD']. c) ['ab', 'cd']. d) none of the mentioned

3. What is the output of the following?

```
i = 1
```

```
while True:
```

```
    if i%3 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 1
```

a) 1 2 b) 1 2 3 c) error d) none of the mentioned

4. What is the output of the following?

```
i = 1
```

```
while True:
```

```
    if i%007 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 1
```

a) 1 2 3 4 5 6 b) 1 2 3 4 5 6 7 c) error d) none of the mentioned

5. What is the output of the following?

```
i = 5
```

```
while True:
```

```
    if i%10 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 1
```

a) 5 6 7 8 9 10 b) 5 6 7 8 c) 5 6 d) error

6. What is the output of the following?

```
i = 5
```

```
while True:
```

```
    if i%9 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 1
```

a) 5 6 7 8 b) 5 6 7 8 9 c) 5 6 7 8 9 10 11 12 13 14 15 d) error

7. What is the output of the following?

```
i = 1
```

```
while True:
```

```
    if i%2 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 2
```

a) 1 b) 1 2 c) 1 2 3 4 5 6 d) 1 3 5 7 9 11 ...

8. What is the output of the following?

```
i = 2
```

```
while True:
```

```
    if i%3 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 2
```

a) 2 4 6 8 10 ... b) 2 4 c) 2 3 d) error

9. What is the output of the following?

```
i = 1
```

```
while False:
```

```
    if i%2 == 0:
```

```
        break
```

```
    print(i)
```

```
    i += 2
```

a) 1 b) 1 3 5 7 ... c) 1 2 3 4 ... d) none of the mentioned

10. What is the output of the following?

```
True = False
while True:
    print(True)
    break
```

- a) True b) False c) None d) none of the mentioned

11. What is the output of the following?

```
i = 0
while i < 3:
    print(i)
    i += 1
else:
    print(0)
```

- a) 0 1 2 3 0 b) 0 1 2 0 c) 0 1 2 d) error

12. What is the output of the following?

```
x = "abcdef"
while i in x:
    print(i, end=" ")
```

- a) a b c d e f b) abcdef c) i i i i i ... d) error

13. What is the output of the following?

```
x = "abcdef"
i = "i"
while i in x:
    print(i, end=" ")
```

- a) no output b) i i i i i ... c) a b c d e f d) abcdef

14. What is the output of the following?

```
x = "abcdef"
i = "a"
while i in x:
    print(i, end=" ")
```

- a) no output b) i i i i i ... c) a a a a a ... d) a b c d e f

15. What is the output of the following?

```
x = "abcdef"
i = "a"
while i in x:
    print('i', end=" ")
```

- a) no output b) i i i i i ... c) a a a a a ... d) a b c d e f

16. What is the output of the following?

```
x = "abcdef"
i = "a"
while i in x:
    x = x[:-1]
    print(i, end = " ")
```

- a) i i i i i b) a a a a a a c) a a a a a d) none of the mentioned

17. What is the output of the following?

```
x = "abcdef"
i = "a"
while i in x[:-1]:
    print(i, end = " ")
```

- a) a a a a a b) a a a a a a c) a a a a a a ... d) a

18. What is the output of the following?

```
x = "abcdef"
i = "a"
while i in x:
    x = x[1:]
    print(i, end = " ")
```

- a) a a a a a a b) a c) no output d) error

19. What is the output of the following?

```
x = "abcdef"
i = "a"
while i in x[1:]:
    print(i, end = " ")
```

- a) a a a a a a b) a c) no output d) error

20. What is the output of the following?

```
x = 'abcd'
for i in x:
    print(i)
    x.upper()
```

- a) a B C D b) a b c d c) A B C D d) error

String

Access characters/elements in a string

We can access individual characters using indexing and a range of characters using

slicing. Index starts from 0. Trying to access a character out of index range will raise an `IndexError`. The index must be an integer. We can't use float or other types, this will result into `TypeError`.

0	1	2	3	4	5	6	7	8
p	r	o	g	r	a	M	m	e

```
s = 'programme'
print(s[3])
print(s[2])
```

Change or delete a string

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name. If we try to change and add element in string then it show error.

Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The `+` operator does this in Python. Simply writing two string literals together also concatenates them.

The `*` operator can be used to repeat the string for a given number of times.

```
a = 'python'
b = 'programming'
print(a+b)
print(a*4)
```

Iterating Through String

We can perform iteration operation using **for loop** of string.

```
s = 'python'
for a in s:
    print(a)
```

String Membership Test

We can test if a sub string exists within a string or not, using the keyword `in`.

```
>>> 'a' in 'program'
True
>>> 'at' not in 'battle'
False
```

Python String Formatting

If we want to print a text like -**He said, "What's there?"** - we can neither use single quote or double quotes. This will result into SyntaxError as the text itself contains both single and double quotes.

```
# using triple quotes
print("He said, "What's there?")

# escaping single quotes
print('He said, "What\'s there?")

# escaping double quotes
print("He said, \"What's there?\")
```

Here is a list of all the escape sequence supported by Python

Escape Sequence in Python

Escape Sequence Description

Escape Sequence	Description
<code>\newline</code>	Backslash and newline ignored
<code>\\</code>	Backslash
<code>\'</code>	Single quote
<code>\"</code>	Double quote
<code>\a</code>	ASCII Bell
<code>\b</code>	ASCII Backspace
<code>\n</code>	ASCII Linefeed
<code>\t</code>	ASCII Horizontal Tab
<code>\v</code>	ASCII Vertical Tab

Raw String to ignore escape sequence

Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place **r** or **R** in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored

The format() Method for Formatting Strings

The `format()` method that is available with the string object is very versatile and powerful in formatting strings. Format strings contains curly braces `{}` as placeholders or replacement fields which gets replaced.

```
# default(implicit) order
default_order = "{}, {} and {}".format('John','Bill','Sean')
print("\n--- Default Order ---")
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('John','Bill','Sean')
print("\n--- Positional Order ---")
print(positional_order)
```

String Methods

Method	Description
capitalize()	Converts first character to Capital Letter
count()	returns occurrences of substring in string
endswith()	Checks if String Ends with the Specified Suffix
find()	Returns the index of first occurrence of substring
format()	formats string into nicer output
index()	Returns Index of Substring
isalnum()	Checks Alphanumeric Character
isalpha()	Checks if All Characters are Alphabets
isdigit()	Checks Digit Characters
islower()	Checks if all Alphabets in a String are Lowercase
isspace()	Checks Whitespace Characters
istitle()	Checks for Titlecased String
isupper()	returns if all characters are uppercase characters
ljust()	returns left-justified string of given width
lower()	returns lowercased string
upper()	returns uppercased string
swapcase()	swap uppercase characters to lowercase; vice versa
strip()	Removes Both Leading and Trailing Characters
replace()	Replaces Substring Inside
split()	Splits String from Left
splitlines()	Splits String at Line Boundaries
startswith()	Checks if String Starts with the Specified String
title()	Returns a Title Cased String
ord()	returns Unicode code point for Unicode character

capitalize()

In Python, the `capitalize()` method converts first character of a string to uppercase letter and lowercases all other characters, if any.

Syntax:

```
string.capitalize()
```

```
string = "python is AWesome."
capitalized_string = string.capitalize()
print('Old String: ', string)
print('Capitalized String:', capitalized_string)
```

When you run the program, the output will be:

```
Old String: python is AWesome
Capitalized String: Python is awesome
```

count()

The string `count()` method returns the number of occurrences of a substring in the given string.

In simple words, `count()` method searches the substring in the given string and returns how many times the substring is present in it.

It also takes optional parameters start and end to specify the starting and ending positions in the string respectively.

Syntax:

```
string.count(substring, start=..., end=...)
```

String count() Parameters

count() method only requires a single parameter for execution. However, it also has two optional parameters:

- **substring** - string whose count is to be found.
- **start (Optional)** - starting index within the string where search starts.
- **end (Optional)** - ending index within the string where search ends.

Note: Index in Python starts from 0, not 1.

Example 1: Count number of occurrences of a given substring

```
string = "Python is awesome, isn't it?"
substring = "is"
count = string.count(substring)
# print count
print("The count is:", count)
```

When you run the program, the output will be:

```
The count is: 2
```

Example 2: Count number of occurrences of a given substring using start and end

```
string = "Python is awesome, isn't it?"
substring = "i"
count = string.count(substring, 8, 25)
print("The count is:", count)
```

When you run the program, the output will be:

```
The count is: 1
```

endswith()

The endswith() method returns True if a string ends with the specified suffix. If not, it returns False.

Syntax:

```
str.endswith(suffix[, start[, end]])
```

endswith() Parameters

The endswith() takes three parameters:

- **suffix** - String or tuple of suffixes to be checked
- **start (optional)** - Beginning position where suffix is to be checked within the string.

- **end (optional)** - Ending position where suffix is to be checked within the string.

Return Value from endswith()

The endswith() method returns a Boolean.

- It returns True if strings ends with the specified suffix.
- It returns False if string doesn't end with the specified suffix.

find()

The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

Syntax:

str.find(sub[, start[, end]])

find() Parameters

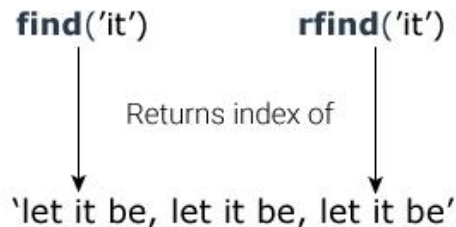
The find() method takes maximum of three parameters:

- **sub** - It's the substring to be searched in the str string.
- **start and end (optional)** - substring is searched within str[start:end]

Return Value from find()

The find() method returns an integer value.

- If substring exists inside the string, it returns the index of first occurrence of the substring.
- If substring doesn't exist inside the string, it returns -1



Example 1: find() With No start and end Argument

```
quote = 'Let it be, let it be, let it be'

result = quote.find('let it')
print("Substring 'let it':", result)

result = quote.find('small')
print("Substring 'small ':", result)
# How to use find()
if (quote.find('be,') != -1):
    print("Contains substring 'be,")
else:
    print("Doesn't contain substring")
```

When you run the program, the output will be:

Substring 'let it': 11

Substring 'small': -1
Contains substring 'be,'

Example 2: find() With start and end Arguments

```
quote = 'Do small things with great love'  
# Substring is searched in 'hings with great love'  
print(quote.find('small things', 10))  
  
# Substring is searched in ' small things with great love'  
print(quote.find('small things', 2))  
  
# Substring is searched in 'hings with great lov'  
print(quote.find('o small ', 10, -1))  
  
# Substring is searched in 'll things with'  
print(quote.find('things ', 6, 20))
```

When you run the program, the output will be:

-1
3
-1
9

index()

The index() method returns the index of a substring inside the string (if found). If the substring is not found, it raises an exception.

Syntax

```
str.index(sub[, start[, end]])
```

index() Parameters

The index() method takes three parameters:

- **sub** - substring to be searched in the string str.
- **start and end(optional)** - substring is searched within str[start:end]

Return Value from index()

If substring exists inside the string, it returns the lowest index in the string where substring is found.

If substring doesn't exist inside the string, it raises a ValueError exception.
The index() method is similar to find() method for strings.

The only difference is that find() method returns -1 if the substring is not found, whereas index() throws an exception.

Example 1: index() With Substring argument Only


```
sentence = 'Python programming is fun.'

result = sentence.index('is fun')
print("Substring 'is fun':", result)

result = sentence.index('Java')
print("Substring 'Java':", result)
```

When you run the program, the output will be:

Substring 'is fun': 19

Traceback (most recent call last):

File "...", line 6, in

result = sentence.index('Java')

ValueError: substring not found

Example 2: index() With start and end Arguments

```
sentence = 'Python programming is fun.'

# Substring is searched in 'gramming is fun.'
print(sentence.index('ing', 10))

# Substring is searched in 'gramming is '
print(sentence.index('g is', 10, -4))

# Substring is searched in 'programming'
print(sentence.index('fun', 7, 18))
```

When you run the program, the output will be:

```
15
17
Traceback (most recent call last)
File "...", line 10, in
print(sentence.index('fun', 7, 18))
ValueError: substring not found
```

isalnum()

The isalnum() method returns True if all characters in the string are alphanumeric (either alphabets or numbers). If not, it returns False.

Syntax:

```
string.isalnum()
```

isalnum() Parameters

The `isalnum()` doesn't take any parameters.

Return Value from `isalnum()`

The `isalnum()` returns:

- True if all characters in the string are alphanumeric
- False if at least one character is not alphanumeric

Example 1: Working of `isalnum()`

```
name = "M234onica"
print(name.isalnum())

name = "M3onica Gell22er "
print(name.isalnum())

name = "Mo3nicaGell22er"
print(name.isalnum())
name = "133"
print(name.isalnum())
```

When you run the program, the output will be:

```
True
False
True
True
```

```
name = "M0n1caG3ll3r"

if name.isalnum() == True:
    print("All characters of string (name) are alphanumeric.")
else:
    print("All characters are not alphanumeric.")
```

When you run the program, the output will be:

```
All characters of string (name) are alphanumeric.
```

`isalpha()`

The `isalpha()` method returns True if all characters in the string are alphabets. If not, it returns False.

Syntax:

```
string.isalpha()
```

`isalpha()` Parameters

The `isalpha()` doesn't take any parameters.

Return Value from `isalpha()`

The `isalpha()` returns:

- True if all characters in the string are alphabets (can be both lowercase and uppercase).
- False if at least one character is not alphabet.

Example 1: Working of isalpha()

```
name = "Monica"
print(name.isalpha())

# contains whitespace
name = "Monica Geller"
print(name.isalpha())

# contains number
name = "Mo3nicaGell22er"
print(name.isalpha())
```

When you run the program, the output will be:

```
True
False
False
```

Example 2:

When you run the program, the output will be:

```
name = "MonicaGeller"

if name.isalpha() == True:
    print("All characters are alphabets")
else:
    print("All characters are not alphabets.")
```

```
All characters are alphabets
```

isdigit()

The isdigit() method returns True if all characters in a string are digits. If not, it returns False.

Syntax:

```
string.isdigit()
```

isdigit() Parameters

The isdigit() doesn't take any parameters.

Return Value from isdigit()

The isdigit() returns:

- True if all characters in the string are digits.
- False if at least one character is not a digit.

Example 1: Working of isdigit()

```
s = "28212"
print(s.isdigit())

# contains alphabets and spaces
s = "Mo3 nicaG el l22er"
print(s.isdigit())
```

islower()

The islower() method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.

Syntax:

```
string.islower()
```

islower() parameters

The islower() method doesn't take any parameters.

Return Value from islower()

The islower() method returns:

- True if all alphabets that exist in the string are lowercase alphabets.
- False if the string contains at least one uppercase alphabet.

Example 1: Return Value from islower()

```
s = 'this is good'
print(s.islower())

s = 'th!s is a1so g00d'
print(s.islower())

s = 'this is Not good'
print(s.islower())
```

When you run the program, the output will be:

```
True
True
False
```

Example 2: How to use islower() in a program?

```
s = 'this is good'
if s.islower() == True:
    print('Does not contain uppercase letter.')
else:
    print('Contains uppercase letter.')

s = 'this is Good'
if s.islower() == True:
    print('Does not contain uppercase letter.')
else:
    print('Contains uppercase letter.')
```

When you run the program, the output will be:

```
Does not contain uppercase letter.
Contains uppercase letter.
```

isspace()

The `isspace()` method returns True if there are only whitespace characters in the string. If not, it returns False.

Characters that are used for spacing are called whitespace characters. For example: tabs, spaces, newline etc.

Syntax:

```
string.isspace()
```

isspace() Parameters

The `isspace()` method doesn't take any parameters.

Return Value from isspace()

The `isspace()` method returns:

- True if all characters in the string are whitespace characters
- False if the string is empty or contains at least one non-printable() character

Example 1: Working of isspace()

```
s = ' \t'
print(s.isspace())

s = ' a '
print(s.isspace())

s = ""
print(s.isspace())
```

When you run the program, the output will be:

True
False
False

Example 2: How to use isspace()?

```
s = '\t \n'
if s.isspace() == True:
    print('All whitespace characters')
else:
    print('Contains non-whitespace characters')

s = '2+2 = 4'

if s.isspace() == True:
    print('All whitespace characters')
else:
    print('Contains non-whitespace characters.')
```

When you run the program, the output will be:

All whitespace characters
Contains non-whitespace characters

istitle()

The istitle() returns True if the string is a titlecased string. If not, it returns False.

Syntax:

string.istitle()

istitle() Parameters

The istitle() method doesn't take any parameters.

Return Value from istitle()

The istitle() method returns:

- True if the string is a titlecased string
- False if the string is not a titlecased string or an empty string

Example 1: Working of istitle()

```
s = 'Python Is Good.'
print(s.istitle())

s = 'Python is good'
print(s.istitle())

s = 'This Is @ Symbol.'
print(s.istitle())
```


When you run the program, the output will be:

```
True
False
True
```

Example 2: How to use istitle()?

```
s = 'I Love Python.'
if s.istitle() == True:
    print('Titlecased String')
else:
    print('Not a Titlecased String')

s = 'PYthon'
if s.istitle() == True:
    print('Titlecased String')
else:
    print('Not a Titlecased String')
```

When you run the program, the output will be:

```
Titlecased String
Not a Titlecased String
```

isupper()

The islower() method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.

Syntax:

```
string.islower()
```

islower() parameters

The islower() method doesn't take any parameters.

Return Value from islower()

The islower() method returns:

- True if all alphabets that exist in the string are lowercase alphabets.
- False if the string contains at least one uppercase alphabet.

Example 1: Return Value from islower()

```
s = 'this is good'
print(s.islower())

s = 'th!s is a1so g00d'
print(s.islower())

s = 'this is Not good'
print(s.islower())
```

When you run the program, the output will be:

True
True
False

Example 2: How to use islower() in a program?

```
s = 'this is good'
if s.islower() == True:
    print('Does not contain uppercase letter.')
else:
    print('Contains uppercase letter.')
s = 'this is Good'
```

```
if s.islower() == True:
    print('Does not contain uppercase letter.')
else:
    print('Contains uppercase letter.')
```

When you run the program, the output will be:

Does not contain uppercase letter.
Contains uppercase letter.

lower():-

The string lower() method converts all uppercase characters in a string into lowercase characters and returns it.

Syntax:

string.lower()

String lower() Parameters()

The lower() method doesn't take any parameters.

Return value from String lower()

The lower() method returns the lowercased string from the given string. It converts all uppercase characters to lowercase.

If no uppercase characters exist, it returns the original string.

Example 1: Convert a string to lowercase

```
# example string
string = "THIS SHOULD BE LOWERCASE!"
print(string.lower())

# string with numbers
# all alphabets should be lowercase
string = "Th!s Sh0uLd B3 L0w3rCas3!"
print(string.lower())
```

When you run the program, the output will be:

```
this should be lowercase!
th!s sh0uld b3 l0w3rcas3!
```

Example 2: How lower() is used in a program?

```
# first string
firstString = "PYTHON IS AWESOME!"

# second string
secondString = "PyThOn Is AwEsOmE!"

if(firstString.lower() == secondString.lower()):
    print("The strings are same.")
else:
    print("The strings are not same.")
```

When you run the program, the output will be:

The strings are same.

Note: If you want to convert to uppercase string, use upper(). You can also use swapcase() to swap between lowercase to uppercase.

upper()

The string upper() method converts all lowercase characters in a string into uppercase characters and returns it.

Syntax:

string.upper()

String upper() Parameters()

The upper() method doesn't take any parameters.

Return value from String upper()

The upper() method returns the uppercased string from the given string. It converts all lowercase characters to uppercase.

If no lowercase characters exist, it returns the original string.

Example 1: Convert a string to uppercase

```
# example string
string = "this should be uppercase!"
print(string.upper())

# string with numbers
# all alphabets should be lowercase
string = "Th!s Sh0uLd B3 uPp3rCas3!"
print(string.upper())
```

When you run the program, the output will be:

THIS SHOULD BE UPPERCASE!
TH!S SH0ULD B3 UPP3RCAS3!

Example 2: How upper() is used in a program?

```
# first string
firstString = "python is awesome!"

# second string
secondString = "PyThOn Is AwEsOmE!"

if(firstString.upper() == secondString.upper()):
    print("The strings are same.")
else:
    print("The strings are not same.")
```

When you run the program, the output will be:

The strings are same.

Note: If you want to convert to lowercase string, use lower(). You can also use swapcase() to swap between lowercase to uppercase.

swapcase()

The string swapcase() method converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string, and returns it.

Syntax:

```
string.swapcase()
```

Note: Not necessarily, `string.swapcase().swapcase() == string`

String swapcase() Parameters()

The `swapcase()` method doesn't take any parameters.

Return value from String swapcase()

The `swapcase()` method returns the string where all uppercase characters are converted to lowercase, and lowercase characters are converted to uppercase.

Example 1: Swap lowercase to uppercase and vice versa using swapcase()

```
string = "ThIs ShOuLd Be MiXeD cAsEd."  
print(string.swapcase())
```

When you run the program, the output will be:

```
tHiS sHoUID bE mlxEd CaSeD.
```

Note: If you want to convert string to lowercase only, use `lower()`. Likewise, if you want to convert string to uppercase only, use `upper()`.

strip()

The `strip()` method returns a copy of the string with both leading and trailing characters removed (based on the string argument passed).

The `strip()` removes characters from both left and right based on the argument (a string specifying the set of characters to be removed).

Syntax:

```
string.strip([chars])
```

strip() Parameters

chars (optional) - a string specifying the set of characters to be removed.

If the `chars` argument is not provided, all leading and trailing whitespaces are removed from the string.

Return Value from strip()

The `strip()` returns a copy of the string with both leading and trailing characters stripped.

When the combination of characters in the `chars` argument mismatches the character of the string in the left, it stops removing the leading characters.

Similarly, when the combination of characters in the `chars` argument mismatches the character of the string in the right, it stops removing the trailing characters.

Example: Working of strip()

```
string = ' xoxo love xoxo '

# Leading whitespace are removed
print(string.strip())

print(string.strip(' xoxoe'))

# Argument doesn't contain space
# No characters are removed.
print(string.strip('sti'))

string = 'android is awesome'
print(string.strip('an'))
```

When you run the program, the output will be:

```
xoxo love xoxo
lov
xoxo love xoxo
droid is awesome
```

split()

The split() method breaks up a string at the specified separator and returns a list of strings.

Syntax:

```
str.split([separator [, maxsplit]])
```

split() Parameters

The split() method takes maximum of 2 parameters:

separator (optional)- The is a delimiter. The string splits at the specified separator.

If the separator is not specified, any whitespace (space, newline etc.) string is a separator.

maxsplit (optional) - The maxsplit defines the maximum number of splits.

The default value of maxsplit is -1, meaning, no limit on the number of splits.

Return Value from split()

The split() breaks the string at the separator and returns a list of strings.

Example 1: How split() works in Python?

```
text= 'Love thy neighbor'

# splits at space
print(text.split())

grocery = 'Milk, Chicken, Bread'

# splits at ','
print(grocery.split(', '))

# Splitting at ':'
print(grocery.split(':'))
```

When you run the program, the output will be:

```
['Love', 'thy', 'neighbor']
['Milk', 'Chicken', 'Bread']
['Milk, Chicken, Bread']
```

Example 2: How split() works when maxsplit is specified?

When you run the program, the output will be:

```
grocery = 'Milk, Chicken, Bread, Butter'

# maxsplit: 2
print(grocery.split(', ', 2))

# maxsplit: 1
print(grocery.split(', ', 1))
# maxsplit: 5
print(grocery.split(', ', 5))
# maxsplit: 0
print(grocery.split(', ', 0))
```

```
['Milk', 'Chicken', 'Bread, Butter']
['Milk', 'Chicken, Bread, Butter']
['Milk', 'Chicken', 'Bread', 'Butter']
['Milk, Chicken, Bread, Butter']
```

If **maxsplit** is specified, the list will have the maximum of maxsplit+1 items.

splitlines()

The `splitlines()` method splits the string at line breaks and returns a list of lines in the string.

Syntax:

```
str.splitlines([keepends])
```

splitlines() Parameters

The `splitlines()` takes maximum of 1 parameter.

keepends (optional) - If `keepends` is provided and `True`, line breaks are also included in items of the list.

By default, the line breaks are not included.

Return Value from splitlines()

The `splitlines()` returns a list of lines in the string.

- If there are not line break characters, it returns a list with single item (a single line).

The `splitlines()` splits on the following line boundaries:

Representation	Description
<code>\n</code>	Line Feed
<code>\r</code>	Carriage Return
<code>\r\n</code>	Carriage Return + Line Feed
<code>\t</code> or <code>\x0b</code>	Line Tabulation
<code>\f</code> or <code>\x0c</code>	Form Feed
<code>\x1c</code>	File Separator
<code>\x1d</code>	Group Separator
<code>\x1e</code>	Record Separator
<code>\x85</code>	Next Line (C1 Control Code)
<code>\u2028</code>	Line Separator
<code>\u2029</code>	Paragraph Separator


```
grocery = 'Milk\nChicken\r\nBread\rButter'
print(grocery.splitlines())
print(grocery.splitlines(True))

grocery = 'Milk Chicken Bread Butter'
print(grocery.splitlines())
```

When you run the program, the output will be:

```
['Milk', 'Chicken', 'Bread', 'Butter']
['Milk\n', 'Chicken\r\n', 'Bread\r', 'Butter']
['Milk Chicken Bread Butter']
```

startswith()

The startswith() method returns True if a string starts with the specified prefix(string). If not, it returns False.

Syntax:

```
str.startswith(prefix[, start[, end]])
```

startswith() Parameters

The startswith() method takes maximum of three parameters:

- **prefix** - String or tuple of strings to be checked
- **start (optional)** - Beginning position where prefix is to be checked within the string.
- **end (optional)** - Ending position where prefix is to be checked within the string.

Return Value from startswith()

The startswith() method returns a boolean.

- It returns True if the string starts with the specified prefix.
- It returns False if the string doesn't start with the specified prefix.

startswith() With start and end Parameters

```
text = "Python programming is easy."
result = text.startswith('programming is', 7)
print(result)

result = text.startswith('programming is', 7, 18)
print(result)

result = text.startswith('program', 7, 18)
print(result)
text = "Python is easy to learn."
```

When you run the program, the output will be:

```
True
False
True
```

Passing Tuple to startswith()

It's possible to pass a tuple of prefixes to the startswith() method in Python.

If the string starts with any item of the tuple, startswith() returns True. If not, it returns False

Example 3: startswith() With Tuple Prefix

```
text = "programming is easy"
result = text.startswith(('python', 'programming'))

print(result)

result = text.startswith(('is', 'easy', 'java'))
print(result)

result = text.startswith(('programming', 'easy'), 12, 19)

# prints False
print(result)
```

When you run the program, the output will be:

```
True
False
False
```

title()

The title() method returns a string with first letter of each word capitalized; a title cased string.

Syntax:

```
str.title()
```

title() Parameters

The title() method doesn't take any parameters.

Return Value from title()

The title() method returns a title cased version of the string. Meaning, the first character of the each word is capitalized (if the first character is a letter).

Example 1: How Python title() works?

```
text = 'My favorite number is 25.'  
print(text.title())
```

```
text = '234 k3l2 *43 fun'  
print(text.title())
```

When you run the program, the output will be:

```
My Favorite Number Is 25.  
234 K3L2 *43 Fun
```

Example 2: title() with apostrophes

```
text = "He's an engineer, isn't he?"  
print(text.title())
```

When you run the program, the output will be:

```
He'S An Engineer, Isn'T He?
```

The title() capitalizes the first letter after apostrophes as well.

ord(c)

The ord() method returns an integer representing Unicode code point for the given Unicode character.

Syntax:

```
ord(c)
```

The ord() method is the inverse of chr().

ord() Parameters

The ord() method takes a single parameter:

c - character string of length 1 whose Unicode code point is to be found

Return value from ord()

The ord() method returns an integer representing the Unicode code point of the given Unicode character.

Example 1: How ord() works in Python?

```
# code point of integer
print(ord('5'))

# code point of alphabet
print(ord('A'))

# code point of character
print(ord('$'))
```

When you run the program, the output will be:

```
53
65
36
```

Assignment

1. What is the output when following statement is executed ?

```
>>>"a"+"bc"
```

- a) a b) bc c) bca d) abc

2. What is the output when following statement is executed ?

```
>>>"abcd"[2:]
```

- a) a b) ab c) cd d) dc

3. The output of executing string.ascii_letters can also be achieved by:

- a) string.ascii_lowercase_string.digits
b) string.ascii_lowercase+string.ascii_uppercase
c) string.letters
d) string.lowercase_string.uppercase

4. What is the output when following code is executed ?

```
>>> str1 = 'hello'
```

```
>>> str2 = ','
```

```
>>> str3 = 'world'
```

```
>>> str1[-1:]
```

- a) olleh b) hello c) h d) o

5. What arithmetic operators cannot be used with strings ?

- a) + b) * c) – d) All of the mentioned

6. What is the output when following code is executed ?

```
>>>print (r"\nhello")
```

- a) a new line and hello b) \nhello
c) the letter r and then hello d) error

7. What is the output when following statement is executed ?

```
>>>print('new' 'line')
```

- a) Error b) Output equivalent to print 'new\nline'
c) newline d) new line

8. What is the output when following statement is executed ?

```
>>> print('x\97\x98')
```

- a) Error b) 9798
c) x\97 d) \x97\x98

9. What is the output when following code is executed ?

```
>>>str1="helloworld"
```

```
>>>str1[::-1]
```

- a) dlrowolleh b) hello c) world d) helloworld

10. print(0xA + 0xB + 0xC) :

- a) 0xA0xB0xC b) Error c) 0x22 d) 33

List

List is a collection which is ordered and changeable. Allows duplicate members. Lists are just like the arrays, declared in other languages. Lists need not be homogeneous always which makes it a most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are also very useful for implementing stacks and queues. Lists are mutable, and hence, they can be altered even after their creation.

List Index

We can use the index operator `[]` to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

Trying to access an element other than this will raise an `IndexError`. The index must be an integer. We can't use float or other types, this will result into `TypeError`.

Nested lists are accessed using nested indexing.

How to slice lists in Python?

We can access a range of items in a list by using the slicing operator (colon).

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need two indexes that will slice that portion from the list.

How to change or add elements to a list?

Lists are mutable, meaning, their elements can be changed unlike string or tuple.

We can use assignment operator (`=`) to change an item or a range of items.

```
# mistake values
odd = [2, 4, 6, 8]
odd[0] = 1
print(odd)

odd[1:4] = [3, 5, 7]
print(odd)
```

How to delete or remove elements from a list?

We can delete one or more items from a list using the keyword `del`. It can even delete the list entirely.

```
my_list = ['p','r','o','b','l','e','m']

del my_list[2]
print(my_list)
del my_list[1:5]
print(my_list)
```

We can use `remove()` method to remove the given item or `pop()` method to remove an item at the given index.

The `pop()` method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

We can also use the `clear()` method to empty a list.

Python List Methods

Methods that are available with list object in Python programming are tabulated below. They are accessed as **list.method()**. Some of the methods have already been used above.

Python Methods	Description
append()	Add an element to the end of the list
extend()	Add all elements of a list to the another list
insert()	Insert an item at the defined index
remove()	Removes an item from the list
pop()	Removes and returns an element at the given index
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of number of items passed as an argument
sort()	Sort items in a list in ascending order
reverse()	Reverse the order of items in the list
copy()	Returns a shallow copy of the list

append()

The `append()` method adds an item to the end of the list.

The `append()` method adds a single item to the existing list. It doesn't return a new list; rather it modifies the original list.

Syntax:

```
list.append(item)
```

append() Parameters

The `append()` method takes a single item and adds it to the end of the list.

The item can be numbers, strings, another list, dictionary etc.

Return Value from append()

As mentioned, the `append()` method only modifies the original list. It doesn't return any value.

Example 1: Adding Element to a List

When you run the program, the output will be:

Updated animal list: ['cat', 'dog', 'rabbit', 'guinea pig']

Example 2: Adding List to a List

```
# animal list
animal = ['cat', 'dog', 'rabbit']

# an element is added
animal.append('guinea pig')

#Updated Animal List
print('Updated animal list: ', animal)
# animal list
animal = ['cat', 'dog', 'rabbit']

# another list of wild animals
wild_animal = ['tiger', 'fox']

# adding wild_animal list to animal list
animal.append(wild_animal)

#Updated List
print('Updated animal list: ', animal)
```

When you run the program, the output will be:

Updated animal list: ['cat', 'dog', 'rabbit', ['tiger', 'fox']]

It's important to notice that, a single item (wild_animal list) is added to the animal list in the above program.

If you need to add items of a list to the another list (rather than the list itself), extend() method is used.

extend()

The extend() extends the list by adding all items of a list (passed as an argument) to the end.

Syntax:

```
list1.extend(list2)
```

Here, the elements of list2 are added to the end of list1.

extend() Parameters

As mentioned, the extend() method takes a single argument (a list) and adds it to the end.

If you need to add elements of other native datatypes (like tuple and set) to the list, you can simply use:

```
# add elements of a tuple to list
list.extend(list(tuple_type))
or even easier
list.extend(tuple_type)
```

Return Value from extend()

The extend() method only modifies the original list. It doesn't return any value.

Example 1: Using extend() Method

```
# language list
language = ['French', 'English', 'German']

# another list of language
language1 = ['Spanish', 'Portuguese']

language.extend(language1)

# Extended List
print('Language List: ', language)
```

When you run the program, the output will be:

```
Language List: ['French', 'English', 'German', 'Spanish', 'Portuguese']
```

Example 2: Add Elements of Tuple and Set to List

```
# language list
language = ['French', 'English', 'German']

# language tuple
language_tuple = ('Spanish', 'Portuguese')

# language set
language_set = {'Chinese', 'Japanese'}
language.extend(language_tuple)

print('New Language List: ', language)
language.extend(language_set)

print('Newest Language List: ', language)
```

When you run the program, the output will be:

```
New Language List: ['French', 'English', 'German', 'Spanish', 'Portuguese']
Newest Language List: ['French', 'English', 'German', 'Spanish', 'Portuguese',
'Japanese', 'Chinese']
```

insert()

The insert() method inserts the element to the list at the given index.

Syntax:

```
list.insert(index, element)
```

insert() Parameters

The insert() function takes two parameters:

index - position where element needs to be inserted

element - this is the element to be inserted in the list

Return Value from insert()

The insert() method only inserts the element to the list. It doesn't return any value.

Example 1: Inserting Element to List

```
# vowel list
vowel = ['a', 'e', 'i', 'u']

# inserting element to list at 4th position
vowel.insert(3, 'o')

print('Updated List: ', vowel)
```

When you run the program, the output will be:

```
Updated List: ['a', 'e', 'i', 'o', 'u']
```

Example 2: Inserting a Tuple (as an Element) to the List

```
mixed_list = [{1, 2}, [5, 6, 7]]

# number tuple
number_tuple = (3, 4)

# inserting tuple to the list
mixed_list.insert(1, number_tuple)
print('Updated List: ', mixed_list)
```

When you run the program, the output will be:

```
Updated List: [{1, 2}, (3, 4), [5, 6, 7]]
```

It is important to note that the index in Python starts from 0 not 1.

If you have to insert element in 4th place, you have to pass 3 as an index. Similarly, if you have to insert element in 2nd place, you have to use 1 as an index.

remove()

The remove() method searches for the given element in the list and removes the first matching element.

Syntax:

```
list.remove(element)
```

remove() Parameters

The remove() method takes a single element as an argument and removes it from the list.

If the element(argument) passed to the remove() method doesn't exist, ValueError exception is thrown.

Return Value from remove()

The remove() method only removes the given element from the list. It doesn't return any value.

Example 1: Remove Element From The List

```
# animal list
animal = ['cat', 'dog', 'rabbit', 'guinea pig']

# 'rabbit' element is removed
animal.remove('rabbit')

#Updated Animal List
print('Updated animal list: ', animal)
```

When you run the program, the output will be:

```
Updated animal list: ['cat', 'dog', 'guinea pig']
```

Example 2: remove() Method on a List having Duplicate Elements

```
# If a list contains duplicate elements
# the remove() method removes only the first instance

# animal list
animal = ['cat', 'dog', 'dog', 'guinea pig', 'dog']

# 'dog' element is removed
animal.remove('dog')

#Updated Animal List
print('Updated animal list: ', animal)
```

When you run the program, the output will be:

```
Updated animal list: ['cat', 'dog', 'guinea pig', 'dog']
Here, only the first occurrence of element dog is removed from the list.
```

Example 3: Trying to Delete Element That Doesn't Exist

```
# animal list
animal = ['cat', 'dog', 'rabbit', 'guinea pig']

# Deleting 'fish' element
animal.remove('fish')

# Updated Animal List
print('Updated animal list: ', animal)
```

When you run the program, you will get the following error:

```
Traceback (most recent call last):  
  File "... ..", line 5, in <module>  
    animal.remove('fish')  
ValueError: list.remove(x): x not in list
```

It's because the element fish doesn't exist in the animal list.

The remove() method removes the element which is passed as an argument.

However, if you need to delete elements based on index (like fourth element or last element), you need to use either pop() method or del operator.

pop()

The pop() method removes and returns the element at the given index (passed as an argument) from the list.

Syntax:

```
list.pop(index)
```

pop() parameter

The pop() method takes a single argument (index) and removes the element present at that index from the list.

If the index passed to the pop() method is not in the range, it throws IndexError: pop index out of range exception.

The parameter passed to the pop() method is optional. If no parameter is passed, the default index -1 is passed as an argument which returns the last element.

Return Value from pop()

The pop() method returns the element present at the given index.

Also, the pop() method removes the element at the given index and updates the list.

Example 1: Print Element Present at the Given Index from the List

```
# programming language list  
language = ['Python', 'Java', 'C++', 'French', 'C']  
return_value = language.pop(3)  
print('Return Value: ', return_value)  
  
# Updated List  
print('Updated List: ', language)
```

When you run the program, the output will be:

```
Return Value: French  
Updated List: ['Python', 'Java', 'C++', 'C']
```

It is important to note that the index in Python starts from 0 not 1.

So, if you need to pop 4th element, you need to pass 3 to the pop() method.

Example 2: pop() when index is not passed and for negative indices

```
# programming language list
language = ['Python', 'Java', 'C++', 'Ruby', 'C']

# When index is not passed
print('When index is not passed:')
print('Return Value: ', language.pop())
print('Updated List: ', language)
```

When you run the program, the output will be:

When index is not passed:

```
Return Value: C
Updated List: ['Python', 'Java', 'C++', 'Ruby']
```

The pop() method returns and removes the element at the given index.

clear()

The clear() method removes all items from the list.

Syntax:

```
list.clear()
```

clear() Parameters

The clear() method doesn't take any parameters.

Return Value from clear()

The clear() method only empties the given list. It doesn't return any value.

Example 1: Working of clear() method

```
# Defining a list
list = [{1, 2}, ('a'), ['1.1', '2.2']]
a
# clearing the list
list.clear()

print('List:', list)
```

When you run the program, the output will be:

List: []

Note: If you are using Python 2 or Python 3.2 and below, you cannot use the clear() method. You can use the del operator instead.

index()

The `index()` method searches an element in the list and returns its index.
In simple terms, `index()` method finds the given element in a list and returns its position.

However, if the same element is present more than once, `index()` method returns its smallest/first position.

Note: Index in Python starts from 0 not 1.

Syntax:

```
list.index(element)
```

index() Parameters

The index method takes a single argument:

element - element that is to be searched.

Return value from index()

The `index()` method returns the index of the element in the list.

If not found, it raises a `ValueError` exception indicating the element is not in the list.

Example 1: Find position of element in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
index = vowels.index('e')
print('The index of e:', index)

# element 'i' is searched
index = vowels.index('i')
print('The index of i:', index)
```

When you run the program, the output will be:

```
The index of e: 1
The index of i: 2
```

count()

The `count()` method returns the number of occurrences of an element in a list.

In simple terms, `count()` method counts how many times an element has occurred in a list and returns it.

Syntax:

```
list.count(element)
```

count() Parameters

The `count()` method takes a single argument:

element - element whose count is to be found.

Return value from count()

The count() method returns the number of occurrences of an element in a list.

Example 1: Count the occurrence of an element in the list

```
# vowels list
vowels = ['a', 'e', 'i', 'o', 'i', 'u']

# count element 'i'
count = vowels.count('i')
print('The count of i is:', count) # count element 'p'

count = vowels.count('p')
print('The count of p is:', count)
```

When you run the program, the output will be:

```
The count of i is: 2
The count of p is: 0
```

sort()

The sort() method sorts the elements of a given list.

The sort() method sorts the elements of a given list in a specific order - Ascending or Descending.

Syntax:

```
list.sort(key=..., reverse=...)
```

Alternatively, you can also use Python's in-built function sorted() for the same purpose.

```
sorted(list, key=..., reverse=...)
```

Note: Simplest difference between sort() and sorted() is: sort() doesn't return any value while, sorted() returns an iterable list.

sort() Parameters

By default, sort() doesn't require any extra parameters. However, it has two optional parameters:

reverse - If true, the sorted list is reversed (or sorted in Descending order)

key - function that serves as a key for the sort comparison

Return value from sort()

sort() method doesn't return any value. Rather, it changes the original list.

If you want the original list, use sorted().

Example 1: Sort a given list

```
vowels = ['e', 'a', 'u', 'o', 'i']
vowels.sort()
print('Sorted list:', vowels)
```

When you run the program, the output will be:

```
Sorted list: ['a', 'e', 'i', 'o', 'u']
```

How to sort in Descending order?

sort() method accepts a reverse parameter as an optional argument.

Setting reverse=True sorts the list in the descending order.

```
list.sort(reverse=True)
```

Alternately for sorted(), you can use the following code.

```
sorted(list, reverse=True)
```

Example 2: Sort the list in Descending order

```
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']

# sort the vowels
vowels.sort(reverse=True)

# print vowels
print('Sorted list (in Descending):', vowels)
```

When you run the program, the output will be:

```
Sorted list (in Descending): ['u', 'o', 'i', 'e', 'a']
```

How to sort using your own function with key parameter?

If you want your own implementation for sorting, sort() also accepts a key function as an optional parameter.

Based on the results of the key function, you can sort the given list.

```
list.sort(key=len)
```

Alternatively for sorted

```
sorted(list, key=len)
```

Here, len is the Python's in-built function to count the length of an element.

The list is sorted based on the length of its each element, from lowest count to highest.

Example 3: Sort the list using key


```
# take second element for sort
def takeSecond(elem):
    return elem[1]

# random list
random = [(2, 2), (3, 4), (4, 1), (1, 3)]

# sort list with key
random.sort(key=takeSecond)

# print list
print('Sorted list:', random)
```

When you run the program, the output will be:

```
Sorted list: [(4, 1), (2, 2), (1, 3), (3, 4)]
```

reverse()

The reverse() method reverses the elements of a given list.

Syntax:

```
list.reverse()
```

reverse() parameter

The reverse() function doesn't take any argument.

Return Value from reverse()

The reverse() function doesn't return any value. It only reverses the elements and updates the list.

Example 1: Reverse a List

```
# Operating System List
os = ['Windows', 'macOS', 'Linux']
print('Original List:', os)

# List Reverse
os.reverse()

# updated list
print('Updated List:', os)
```

When you run the program, the output will be:

```
Original List: ['Windows', 'macOS', 'Linux']
Updated List: ['Linux', 'macOS', 'Windows']
```

There are other several ways to reverse a list.

Example 2: Reverse a List Using Slicing Operator

```
# Operating System List
os = ['Windows', 'macOS', 'Linux']
print('Original List:', os)

# Reversing a list
#Syntax: reversed_list = os[start:stop:step]
reversed_list = os[::-1]

# updated list
print('Updated List:', reversed_list)
```

When you run the program, the output will be:

```
Original List: ['Windows', 'macOS', 'Linux']
Updated List: ['Linux', 'macOS', 'Windows']
```

Example 3: Accessing Individual Elements in Reversed Order

If you need to access individual elements of a list in reverse order, it's better to use reversed() method.

```
# Operating System List
os = ['Windows', 'macOS', 'Linux']

# Printing Elements in Reversed Order
for o in reversed(os):
    print(o)
```

When you run the program, the output will be:

```
Linux
macOS
Windows
```

Assignment

This set of Python Programming Questions & Answers focuses on “Lists”.

1. Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after list1.reverse() ?
a) [3, 4, 5, 20, 5, 25, 1, 3]. b) [1, 3, 3, 4, 5, 5, 20, 25].
c) [25, 20, 5, 5, 4, 3, 3, 1]. d) [3, 1, 25, 5, 20, 5, 4, 3].
2. Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.extend([34, 5]) ?
a) [3, 4, 5, 20, 5, 25, 1, 3, 34, 5]. b) [1, 3, 3, 4, 5, 5, 20, 25, 34, 5].
c) [25, 20, 5, 5, 4, 3, 3, 1, 34, 5]. d) [1, 3, 4, 5, 20, 5, 25, 3, 34, 5].

3. Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.pop(1) ?

- a) [3, 4, 5, 20, 5, 25, 1, 3]. b) [1, 3, 3, 4, 5, 5, 20, 25].
c) [3, 5, 20, 5, 25, 1, 3]. d) [1, 3, 4, 5, 20, 5, 25].

4. Suppose listExample is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after listExample.pop()?

- a) [3, 4, 5, 20, 5, 25, 1]. b) [1, 3, 3, 4, 5, 5, 20, 25].
c) [3, 5, 20, 5, 25, 1, 3]. d) [1, 3, 4, 5, 20, 5, 25].

5. What is the output when the following code is executed ?

```
>>>"Welcome to Python".split()
```

- a) ["Welcome", "to", "Python"]. b) ("Welcome", "to", "Python")
c) {"Welcome", "to", "Python"} d) "Welcome", "to", "Python"

6. What is the output when following code is executed ?

```
>>>list("a#b#c#d".split('#'))
```

- a) ['a', 'b', 'c', 'd']. b) ['a b c d'].
c) ['a#b#c#d']. d) ['abcd'].

7. What is the output when following code is executed ?

```
myList = [1, 5, 5, 5, 5, 1]  
max = myList[0]  
indexOfMax = 0  
for i in range(1, len(myList)):  
    if myList[i] > max:  
        max = myList[i]  
        indexOfMax = i
```

```
>>>print(indexOfMax)
```

- a) 1 b) 2 c) 3 d) 4

8. What is the output when following code is executed ?

```
myList = [1, 2, 3, 4, 5, 6]  
for i in range(1, 6):  
    myList[i - 1] = myList[i]
```

```
for i in range(0, 6):  
    print(myList[i], end = " ")
```

- a) 2 3 4 5 6 1 b) 6 1 2 3 4 5 c) 2 3 4 5 6 6 d) 1 1 2 3 4 5

9. What is the output when following code is executed ?

```
>>>list1 = [1, 3]  
>>>list2 = list1  
>>>list1[0] = 4  
>>>print(list2)
```

- a) [1, 3]. b) [4, 3]. c) [1, 4]. d) [1, 3, 4].

10. What is the output when following code is executed ?

```
def f(values):  
    values[0] = 44
```

```
v = [1, 2, 3]
```

f(v)

print(v)

- a) [1, 44]. b) [1, 2, 3, 44]. c) [44, 2, 3]. d) [1, 2, 3].

11. What will be the output?

```
def f(i, values = []):  
    values.append(i)  
    return values
```

f(1)

f(2)

v = f(3)

print(v)

- a) [1] [2] [3]. b) [1] [1, 2] [1, 2, 3]. c) [1, 2, 3]. d) 1 2 3

12. What will be the output?

```
names1 = ['Amir', 'Bala', 'Chales']
```

```
if 'amir' in names1:
```

```
    print(1)
```

```
else:
```

```
    print(2)
```

- a) None b) 1 c) 2 d) Error

13. What will be the output?

```
names1 = ['Amir', 'Bala', 'Charlie']
```

```
names2 = [name.lower() for name in names1]
```

```
print(names2[2][0])
```

- a) None b) a c) b d) c 4. What will be the output?

Tuple

Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

Indexing

We can use the index operator `[]` to access an item in a tuple where the index starts from 0. So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other than (6, 7,...) will raise an `IndexError`.

The index must be an integer, so we cannot use float or other types. This will result into `TypeError`.

Slicing

We can access a range of items in a tuple by using the slicing operator - colon `:`. Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.

Changing a Tuple

Unlike lists, tuples are immutable.

This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.

We can also assign a tuple to different values (reassignment).

We can use `+` operator to combine two tuples. This is also called concatenation.

We can also repeat the elements in a tuple for a given number of times using the `*` operator.

Both `+` and `*` operations result into a new tuple.

Deleting a Tuple

As discussed above, we cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.

But deleting a tuple entirely is possible using the keyword **del**.

Python Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available.

Method	Description
count(x)	Return the number of items that is equal to x
index(x)	Return index of first item that is equal to x

Assignment

1. Which of the following is a Python tuple?

- a) `[1, 2, 3]` b) `(1, 2, 3)` c) `{1, 2, 3}` d) `{}`

2. Suppose `t = (1, 2, 4, 3)`, which of the following is incorrect?

- a) `print(t[3])` b) `t[3] = 45` c) `print(max(t))` d) `print(len(t))`

3. Suppose $t = (1, 2, 4, 3)$, which of the following is incorrect?

- a) `print(t[3])` b) `t[3] = 45` c) `print(max(t))` d) `print(len(t))`

4. What will be the output?

```
>>>t=(1,2,4,3)
```

```
>>>t[1:3]
```

- a) (1, 2) b) (1, 2, 4) c) (2, 4) d) (2, 4, 3)

5. What will be the output?

```
>>>t=(1,2,4,3)
```

```
>>>t[1:-1]
```

- a) (1, 2) b) (1, 2, 4) c) (2, 4) d) (2, 4, 3)

6. What will be the output?

```
>>>my_tuple = (1, 2, 3, 4)
```

```
>>>my_tuple.append( (5, 6, 7) )
```

```
>>>print len(my_tuple)
```

- a) 1 b) 2 c) 5 d) Error

7. What will be the output?

```
numberGames = {}
```

```
numberGames[(1,2,4)] = 8
```

```
numberGames[(4,2,1)] = 10
```

```
numberGames[(1,2)] = 12
```

```
sum = 0
```

```
for k in numberGames:
```

```
    sum += numberGames[k]
```

```
print len(numberGames) + sum
```

- a) 30 b) 24 c) 33 d) 12

8. What is the output of the following code?

```
>>> a=(2,3,4)
```

```
>>> sum(a,3)
```

- a) Too many arguments for sum() method
- b) The method sum() doesn't exist for tuples
- c) 12
- d) 9

9. What type of data is: a=[(1,1),(2,4),(3,9)]?

- a) Array of tuples
- b) List of tuples
- c) Tuples of lists
- d) Invalid type

www.wscubetech.com

Dictionary

How to access elements from a dictionary?

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method.

The difference while using get() is that it returns None instead of KeyError, if the key is not found.

```
my_dict = {'name': 'Jack', 'age': 26}
```

```
# Output: Jack
```

```
print(my_dict['name'])
```

```
# Output: 26
```

```
print(my_dict.get('age'))
```

```
# Trying to access keys which doesn't exist throws error
```

```
# my_dict.get('address')
```

```
# my_dict['address']
```

When you run the program, the output will be:

```
Jack  
26
```

How to change or add elements in a dictionary?

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.

If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

```
my_dict = {'name': 'Jack', 'age': 26}
```

```
my_dict['age'] = 27
```

```
print(my_dict)
```

```
my_dict['address'] = 'Downtown'
```

```
print(my_dict)
```

When you run the program, the output will be:

```
{'name': 'Jack', 'age': 27}  
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```


How to delete or remove elements from a dictionary?

We can remove a particular item in a dictionary by using the method `pop()`. This method removes an item with the provided key and returns the value.

The method, `popitem()` can be used to remove and return an arbitrary item (key, value) from the dictionary. All the items can be removed at once using the `clear()` method.

We can also use the `del` keyword to remove individual items or the entire dictionary itself.

```
# create a dictionary
squares = {1:1, 2:4, 3:9, 4:16, 5:25}
```

```
# remove a particular item
```

```
# Output: 16
```

```
print(squares.pop(4))
```

```
# Output: {1: 1, 2: 4, 3: 9, 5: 25}
```

```
print(squares)
```

```
# remove an arbitrary item
```

```
# Output: (1, 1)
```

```
print(squares.popitem())
```

```
# Output: {2: 4, 3: 9, 5: 25}
```

```
print(squares)
```

```
# delete a particular item
```

```
del squares[5]
```

```
# Output: {2: 4, 3: 9}
```

```
print(squares)
```

```
# remove all items
```

```
squares.clear()
```

```
# Output: {}
```

```
print(squares)
```

```
# delete the dictionary itself
```

```
del squares
```

```
# Throws Error
```

```
# print(squares)
```

When you run the program, the output will be:

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(1, 1)
{2: 4, 3: 9, 5: 25}
{2: 4, 3: 9}
{}
```

Python Dictionary Methods

Methods that are available with dictionary are tabulated below. Some of them have already been used in the above examples.

Method	Description
clear()	Remove all items form the dictionary.
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitrary item (key, value). Raises KeyError if the dictionary is empty.
values()	Return a new view of the dictionary's values

clear()

The clear() method removes all items from the dictionary.

Syntax:

```
dict.clear()
```

clear() Parameters

The clear() method doesn't take any parameters.

Return Value from clear()

The clear() method doesn't return any value (returns None).

Example 1: How clear() method works for dictionaries?

```
d = {1: "one", 2: "two"}

d.clear()
print('d =', d)
```

When you run the program, the output will be:

```
d = {}
```

You can also remove all elements from the dictionary by assigning empty dictionary {}.

get()

The get() method returns the value for the specified key if key is in dictionary.

Syntax:

```
dict.get(key[, value])
```

get() Parameters

The get() method takes maximum of two parameters:

key - key to be searched in the dictionary

value (optional) - Value to be returned if the key is not found. The default value is None.

Return Value from get()

The get() method returns:

- the value for the specified key if key is in dictionary.
- None if the key is not found and value is not specified.
- value if the key is not found and value is specified.

Example 1: How get() works for dictionaries?

```
person = {'name': 'Phill', 'age': 22}

print('Name: ', person.get('name'))
print('Age: ', person.get('age'))

# value is not provided
print('Salary: ', person.get('salary'))

# value is provided
print('Salary: ', person.get('salary', 0.0))
```

When you run the program, the output will be:

```
Name: Phill
Age: 22
Salary: None
Salary: 0.0
```

Python get() method Vs dict[key] to Access Elements

The get() method returns a default value if the key is missing.

However, if the key is not found when you use dict[key], KeyError exception is raised.

```
print('Salary: ', person.get('salary'))

print(person['salary'])
```

When you run the program, the output will be:

```
Traceback (most recent call last):  
  File "...", line 1, in <module>  
    print('Salary: ', person.get('salary'))  
NameError: name 'person' is not defined
```

items()

The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.

Syntax:

```
dictionary.items()
```

The items() method is similar to dictionary's viewitems() method in Python 2.7

items() Parameters

The items() method doesn't take any parameters.

Return value from items()

The items() method returns a view object that displays a list of a given dictionary's (key, value) tuple pair.

Example 1: Get all items of a dictionary with items()

```
# random sales dictionary  
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }  
  
print(sales.items())
```

When you run the program, the output will be:

```
dict_items([('apple', 2), ('orange', 3), ('grapes', 4)])
```

Example 2: How items() works when a dictionary is modified?

```
# random sales dictionary  
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }  
  
items = sales.items()  
  
print('Original items:', items)  
  
# delete an item from dictionary  
del[sales['apple']]  
print('Updated items:', items)
```

When you run the program, the output will be:

```
Original items: dict_items([('apple', 2), ('orange', 3), ('grapes', 4)])  
Updated items: dict_items([('orange', 3), ('grapes', 4)])
```

The view object items doesn't itself return a list of sales items but it returns a view of sales's (key, value) pair.

keys()

The keys() method returns a view object that displays a list of all the keys in the dictionary

Syntax:

```
dict.keys()
```

keys() Parameters

The keys() doesn't take any parameters.

Return Value from keys()

The keys() returns a view object that displays a list of all the keys.

When the dictionary is changed, the view object also reflect these changes.

Example 1: How keys() works?

```
person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
print(person.keys())

empty_dict = {}
print(empty_dict.keys())
```

When you run the program, the output will be:

```
dict_keys(['name', 'salary', 'age'])
dict_keys([])
```

Example 2: How keys() works when dictionary is updated?

```
person = {'name': 'Phill', 'age': 22, }

print('Before dictionary is updated')
keys = person.keys()
print(keys)

# adding an element to the dictionary
person.update({'salary': 3500.0})
print("\nAfter dictionary is updated")
print(keys)
```

When you run the program, the output will be:

Before dictionary is updated

```
dict_keys(['name', 'age'])
```

After dictionary is updated

```
dict_keys(['name', 'age', 'salary'])
```

Here, when the dictionary is updated, keys is also automatically updated to reflect changes.

pop()

The pop() method removes and returns an element from a dictionary having the given key.

Syntax:

```
dictionary.pop(key[, default])
```

pop() Parameters

The pop() method takes two parameters:

key - key which is to be searched for removal

default - value which is to be returned when the key is not in the dictionary

Return value from pop()

The pop() method returns:

- If key is found - removed/popped element from the dictionary
- If key is not found - value specified as the second argument (default)
- If key is not found and default argument is not specified - KeyError exception is raised

Example 1: Pop an element from the dictionary

```
# random sales dictionary
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }

element = sales.pop('apple')
print('The popped element is:', element)
print('The dictionary is:', sales)
```

When you run the program, the output will be:

```
The popped element is: 2
The dictionary is: {'orange': 3, 'grapes': 4}
```

popitem()

The popitem() returns and removes an arbitrary element (key, value) pair from the dictionary.

Syntax:

```
dict.popitem()
```

popitem() Parameters

The popitem() doesn't take any parameters.

Return Value from popitem()

The popitem() returns an arbitrary element (key, value) pair from the dictionary removes an arbitrary element (the same element which is returned) from the dictionary.

Note: Arbitrary elements and random elements are not same. The popitem() doesn't return a random element.

Example: How popitem() works?

```
person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
result = person.popitem()
print('person = ',person)
print('Return Value = ',result)
```

When you run the program, the output will be:

```
person = {'name': 'Phill', 'salary': 3500.0}
result = ('age', 22)
```

The popitem() raises a KeyError error if the dictionary is empty.

values()

The values() method returns a view object that displays a list of all the values in the dictionary.

Syntax:

```
dictionary.values()
```

values() Parameters

The values() method doesn't take any parameters.

Return value from values()

The values() method returns a view object that displays a list of all values in a given dictionary.

Example 1: Get all values from the dictionary

```
# random sales dictionary
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }

print(sales.values())
```

When you run the program, the output will be:

```
dict_values([2, 4, 3])
```

Example 2: How values() works when dictionary is modified?

```
# random sales dictionary
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }

values = sales.values()
print('Original items:', values)

# delete an item from dictionary
del[sales['apple']]
print('Updated items:', values)
```

When you run the program, the output will be:

```
Original items: dict_values([2, 4, 3])
Updated items: dict_values([4, 3])
```

The view object values doesn't itself return a list of sales item values but it returns a view of all values of the dictionary.

If the list is updated at any time, the changes are reflected on to the view object itself, as shown in the above program.

1. Which of the following statements create a dictionary?

- a) d = {}
- b) d = {"john":40, "peter":45}
- c) d = {40:"john", 45:"peter"}
- d) All of the mentioned

2. Read the code shown below carefully and pick out the keys?

```
d = {"john":40, "peter":45}
```

- a) "john", 40, 45, and "peter"
- b) "john" and "peter"
- c) 40 and 45
- d) d = (40:"john", 45:"peter")

3. What will be the output?

```
d = {"john":40, "peter":45}
```

```
"john" in d
```

- a) True
- b) False
- c) None
- d) Error

4. What will be the output?

```
d1 = {"john":40, "peter":45}
```

```
d2 = {"john":466, "peter":45}
```

```
d1 == d2
```

- a) True
- b) False
- c) None
- d) Error

5. What will be the output?

```
d1 = {"john":40, "peter":45}
```



```
d2 = {"john":466, "peter":45}
```

```
d1 > d2
```

- a) True b) False c) Error d) None

6. What is the output?

```
d = {"john":40, "peter":45}
```

```
d["john"]
```

- a) 40 b) 45 c) "john" d) "peter"

7. Suppose d = {"john":40, "peter":45}, to delete the entry for "john" what command do we use

- a) d.delete("john":40) b) d.delete("john")
c) del d["john"]. d) del d("john":40)

8. Suppose d = {"john":40, "peter":45}. To obtain the number of entries in dictionary which command do we use?

- a) d.size() b) len(d) c) size(d) d) d.len()

9. What will be the output?

```
d = {"john":40, "peter":45}
```

```
print(list(d.keys()))
```

- a) ["john", "peter"]. b) [{"john":40, "peter":45}].
c) ("john", "peter") d) ("john":40, "peter":45)

10. Suppose d = {"john":40, "peter":45}, what happens when we try to retrieve a value using the expression d["susan"]?

- a) Since "susan" is not a value in the set, Python raises a KeyError exception
b) It is executed fine and no exception is raised, and it returns None
c) Since "susan" is not a key in the set, Python raises a KeyError exception
d) Since "susan" is not a key in the set, Python raises a syntax error

Set

What is a set in Python?

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

However, the set itself is mutable. We can add or remove items from it.

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

How to create a set?

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

How to remove elements from a set?

A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

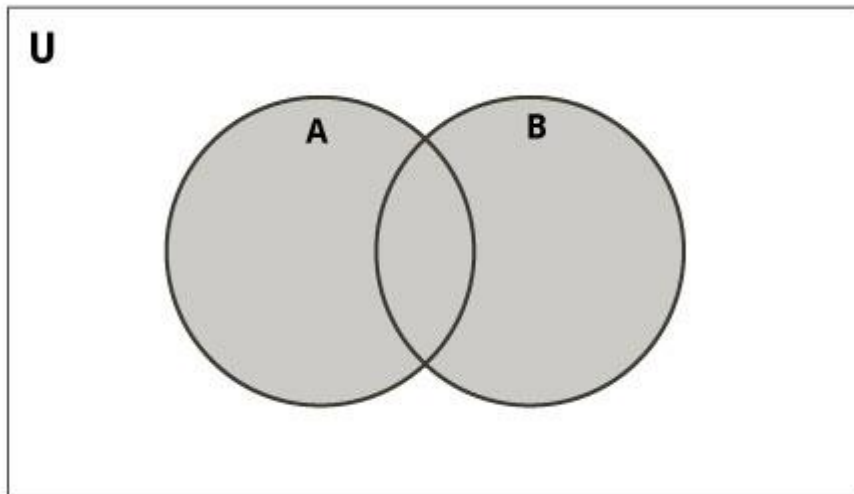
Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Let us consider the following two sets for the following operations.

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
```

Set Union



Union of A and B is a set of all elements from both sets.

Union is performed using | operator. Same can be accomplished using the method union().

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)
```

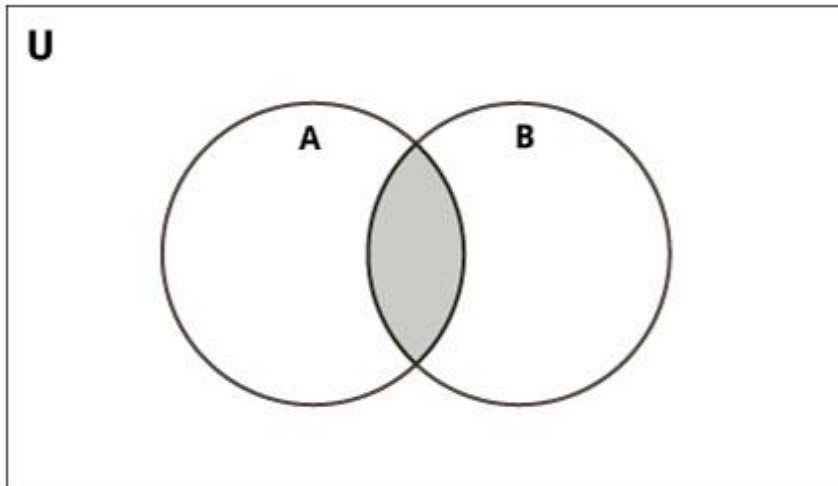
Output:-

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Set Intersection

Intersection of A and B is a set of elements that are common in both sets.

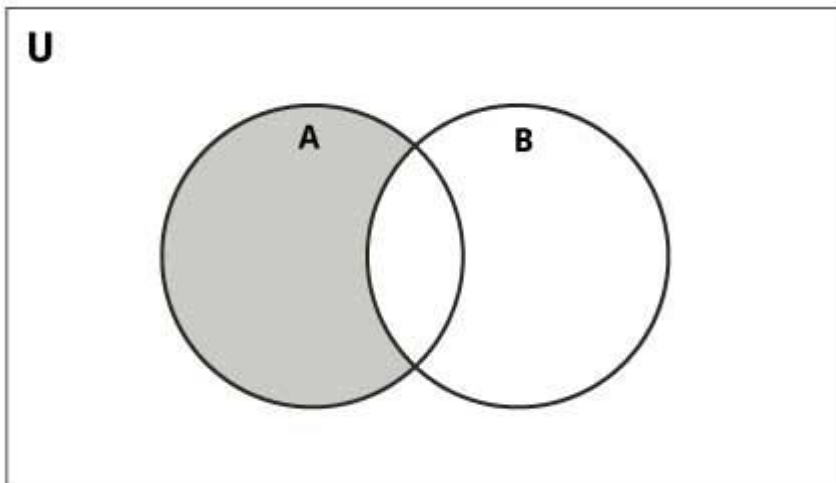
Intersection is performed using & operator. Same can be accomplished using the method intersection ().



```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)
```

Set Difference



Difference of A and B ($A - B$) is a set of elements that are only in A but not in B. Similarly, $B - A$ is a set of element in B but not in A.

Difference is performed using - operator. Same can be accomplished using the method `difference()`.

```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)
```

Python Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
difference()	Returns the difference of two or more sets as a new set
intersection()	Returns the intersection of two sets as a new set
pop()	Removes and returns an arbitrary set element. Raise KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raise a KeyError
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

set()

The set add() method adds a given element to a set. If the element is already present, it doesn't add any element.

Syntax:

```
set.add(elem)
```

The add() method doesn't add an element to the set if it's already present in it.

Also, you don't get back a set if you use add() method when creating a set object.

```
noneValue = set().add(elem)
```

The above statement doesn't return a reference to the set but 'None', because the statement returns the return type of add which is 'None',

Set add() Parameters

The add() method takes a single parameter:

elem - the element that is added to the set Return Value from Set add()

The add() method doesn't return any value and returns a 'None'.

Example 1: Add an element to a set

```
# set of vowels
vowels = {'a', 'e', 'i', 'u'}

# adding 'o'
vowels.add('o')
print('Vowels are:', vowels)

# adding 'a' again
vowels.add('a')
print('Vowels are:', vowels)
```

When you run the program, the output will be:

```
Vowels are: {'a', 'i', 'o', 'u', 'e'}
Vowels are: {'a', 'i', 'o', 'u', 'e'}
```

Note: Order of the vowels can be different.

Example 2: Add tuple to a set

```
# set of vowels
vowels = {'a', 'e', 'u'}

# a tuple ('i', 'o')
tup = ('i', 'o')
# adding tuple
vowels.add(tup)
print('Vowels are:', vowels)

# adding same tuple again
vowels.add(tup)
print('Vowels are:', vowels)
```

When you run the program, the output will be:

```
Vowels are: {'i', 'o', 'e', 'u', 'a'}
Vowels are: {'i', 'o', 'e', 'u', 'a'}
```

clear()

The clear() method removes all elements from the set.

Syntax:

```
set.clear()
```

Set clear() Parameters

The clear() method doesn't take any parameters.

Return value from Set clear()

The clear() method doesn't return any value and returns a 'None'.

Example 1: Remove all elements from a Python set using clear()

```
# set of vowels
vowels = {'a', 'e', 'i', 'o', 'u'}
print('Vowels (before clear):', vowels)

# clearing vowels
vowels.clear()
print('Vowels (after clear):', vowels)
```

When you run the program, the output will be:

```
Vowels (before clear): {'e', 'a', 'o', 'u', 'i'}
Vowels (after clear): set()
```

pop()

The pop() method removes an arbitrary element from the set and returns the element removed.

Syntax:

```
set.pop()
```

pop() Parameters

The pop() method doesn't take any arguments.

Return Value from pop()

The pop() method returns an arbitrary (random) element from the set. Also, the set is updated and will not contain the element (which is returned).

If the set is empty, TypeError exception is raised.

Example: How pop() works for Python Sets?

```
A = {'a', 'b', 'c', 'd'}

print('Return Value is', A.pop())
print('A = ', A)
```

When you run the program, we got the follow output

```
Return Value is d
A = {'a', 'b', 'c'}
```

Note: You may get difference output as pop() returns and removes a random element.

remove()

The `remove()` method searches for the given element in the set and removes it.

Syntax:

```
set.remove(element)
```

remove() Parameters

The `remove()` method takes a single element as an argument and removes it from the set.

If the element(argument) passed to the `remove()` method doesn't exist, `keyError` exception is thrown.

Return Value from remove()

The `remove()` method only removes the given element from the set. It doesn't return any value.

Example 1: Remove Element From The Set

```
# language set
language = {'English', 'French', 'German'}

# 'German' element is removed
language.remove('German')

# Updated language set
print('Updated language set: ', language)
```

When you run the program, the output will be:

```
Updated language set: {'English', 'French'}
```

Example 2: Trying to Delete Element That Doesn't Exist

```
# animal set
animal = {'cat', 'dog', 'rabbit', 'guinea pig'}

# Deleting 'fish' element
animal.remove('fish')

# Updated animal
print('Updated animal set: ', animal)
```

When you run the program, you will get the following error:

```
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
    animal.remove('fish')
KeyError: 'fish'
```


It's because the element fish doesn't exist in the animal set.

You can use `discard()` method if you do not want this error. The set remains unchanged if the element passed to `discard()` method doesn't exist.

A set is an unordered collection of elements. If you need to remove arbitrary element from the set, you can use `pop()` method.

update()

The `update()` adds elements from a set (passed as an argument) to the set (calling the `update()` method).

Syntax:

```
A.update(B)
```

Here, A and B are two sets. The elements of set B are added to the set A.

update() Parameters

The `update()` method takes a single argument (a set).

If you need to add elements of other native datatypes (like tuple, list, dictionary etc.) to the set, you can simply use:

```
# add list elements to set
set.update(set(list))
or even easier
```

```
set.update(list)
```

Return Value from update()

The `A.update(B)` adds elements from the set B to A.

This method returns `None` (meaning, absence of a return value).

Example 1: How update() works in Python?

```
A = {'a', 'b'}
B = {1, 2, 3}

result = A.update(B)
print('A =',A)
print('B =',B)
print('result =',result)
```

When you run the program, the output will be:

```
A = {'a', 1, 2, 3, 'b'}
B = {1, 2, 3}
result = None
```

Example 2: Adding Elements of String and Dictionary to Set

```
# Update With String
string_alphabet = 'abc'
numbers_set = {1, 2}

numbers_set.update(string_alphabet)

print('numbers_set =', numbers_set)
print('string_alphabet =', string_alphabet)

# Update With Dictionary
info_dictionary = {'key': 1, 2 : 'lock'}
numbers_set = {'a', 'b'}

numbers_set.update(info_dictionary)
print('numbers_set =', numbers_set)
```

When you run the program, the output will be:

```
numbers_set = {'c', 1, 2, 'b', 'a'}
string_alphabet = abc
numbers_set = {'key', 'b', 2, 'a'}
```

Assignment

- Which of these about a set is not true?
 - Mutable data type
 - Allows duplicate values
 - Data type with unordered values
 - Immutable data type
- Which of the following is not the correct syntax for creating a set?
 - set([1,2],[3,4])
 - set([1,2,2,3,4])
 - set((1,2,3,4))
 - {1,2,3,4}
- What is the output of the following code?

```
nums = set([1,1,2,3,3,3,4,4])
print(len(nums))
```

 - 7
 - Error, invalid syntax for formation of set
 - 4
 - 8
- What is the output of the following piece of code?

```
a = [5,5,6,7,7,7]
b = set(a)
def test(lst):
    if lst in b:
        return 1
```

```
else:
    return 0
for i in filter(test, a):
    print(i,end=" ")
a) 5 5 6          b) 5 6 7          c) 5 5 6 7 7 7          d) 5 6 7 7 7
```

5. Which of the following statements is used to create an empty set?

- a) { } b) set() c) []. d) ()

6. What is the output of the following piece of code when executed in the python shell?

```
>>> a={5,4}
>>> b={1,2,4,5}
>>> a<b
a) {1,2}      b) True      c) False      d) Invalid operation
```

7. If a={5,6,7,8}, which of the following statements is false?

- a) print(len(a)) b) print(min(a)) c) a.remove(5) d) a[2]=45

8. If a={5,6,7}, what happens when a.add(5) is executed?

- a) a={5,5,6,7}
b) a={5,6,7}
c) Error as there is no add function for set data type
d) Error as 5 already exists in the set

9. What is the output of the following code?

```
>>> a={4,5,6}
>>> b={2,8,6}
>>> a+b
a) {4,5,6,2,8}
b) {4,5,6,2,8,6}
c) Error as unsupported operand type for sets
d) Error as the duplicate item 6 is present in both sets
```

10. What is the output of the following code?

```
>>> a={4,5,6}
>>> b={2,8,6}
>>> a-b
a) {4,5}
b) {6}
c) Error as unsupported operand type for set data type
d) Error as the duplicate item 6 is present in both sets
```

11. What is the output of the following piece of code?

```
>>> a={5,6,7,8}
>>> b={7,8,10,11}
>>> a^b
a) {5,6,7,8,10,11}
b) {7,8}
c) Error as unsupported operand type of set data type
d) {5,6,10,11}
```

12. What is the output of the following code?

```
>>> s={5,6}
```

```
>>> s*3
```

- a) Error as unsupported operand type for set data type
- b) {5,6,5,6,5,6}
- c) {5,6}
- d) Error as multiplication creates duplicate elements which isn't allowed

13. What is the output of the following piece of code?

```
>>> a={5,6,7,8}
```

```
>>> b={7,5,6,8}
```

```
>>> a==b
```

- a) True
- b) False

14. What is the output of the following piece of code?

```
>>> a={3,4,5}
```

```
>>> b={5,6,7}
```

```
>>> a|b
```

- a) Invalid operation
- b) {3, 4, 5, 6, 7}
- c) {5}
- d) {3,4,6,7}

Function

In Python, function is a group of related statements that perform a specific task. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. Furthermore, it avoids repetition and makes code reusable.

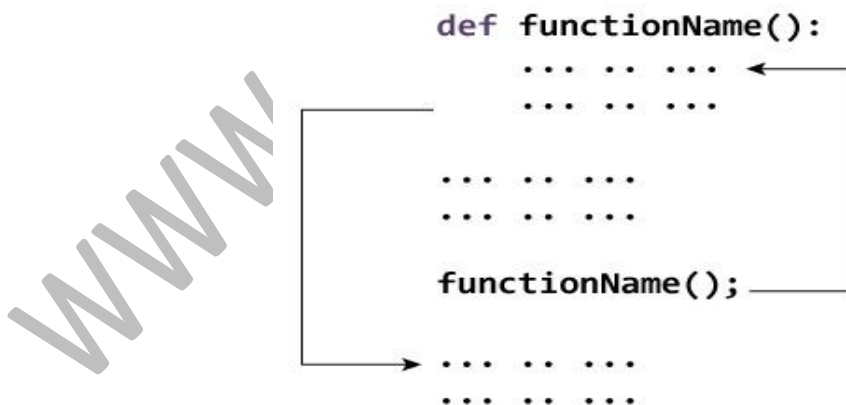
Syntax of Function

```
def function_name(parameters):  
    """docstring"""  
    statement(s)
```

Above shown is a function definition which consists of following components.

1. Keyword **def** marks the start of function header.
2. A function name to uniquely identify it. Function naming follows the same rules of writing identifiers in Python.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

How Function works in Python?



Types of Functions

Basically, we can divide functions into the following two types:

1. **Built-in functions** - Functions that are built into Python.
2. **User-defined functions** - Functions defined by the users themselves.

Docstring:-

The first string after the function header is called the docstring and is short for documentation string. It is used to explain in brief, what a function does.

Although optional, documentation is a good programming practice. Unless you can remember what you had for dinner last week, always document your code.

In the above example, we have a docstring immediately below the function header. We generally use triple quotes so that docstring can extend up to multiple lines. This string is available to us as `__doc__` attribute of the function.

Arguments

In user-defined function topic, we learned about defining a function and calling it. Otherwise, the function call will result into an error.

Variable Function Argument

Up until now functions had fixed number of arguments. In Python there are other ways to define a function which can take variable number of arguments.

Three different forms of this type are described below.

Python Default Arguments

Function arguments can have default values in Python.

We can provide a default value to an argument by using the assignment operator (=).

Python Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position

Python Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function. Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument

```
def greet(*names):  
    """This function greets all  
    the person in the names tuple."""  
  
    # names is a tuple with arguments  
    for name in names:  
        print("Hello",name)  
  
greet("Monica","Luke","Steve","John")
```

What is recursion in Python?

Recursion is the process of defining something in terms of itself.
A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

Python Recursive Function

We know that in Python, a function can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

Following is an example of recursive function to find the factorial of an integer.
Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is $1*2*3*4*5*6 = 720$.

```
def calc_factorial(x):  
    """This is a recursive function  
    to find the factorial of an integer"""  
  
    if x == 1:  
        return 1  
    else:  
        return (x * calc_factorial(x-1))  
num = 4  
print("The factorial of", num, "is", calc_factorial(num))
```

Advantages of Recursion

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

What are lambda functions in Python?

In Python, anonymous function is a function that is defined without a name.
While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.
Hence, anonymous functions are also called lambda functions.

How to use lambda Functions in Python?

A lambda function in python has the following syntax.

Syntax of Lambda Function in python

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

Example of Lambda Function in python

Here is an example of lambda function that doubles the input value.

```
# Program to show the use of lambda functions

double = lambda x: x * 2

# Output: 10
print(double(5))
```

In the above program, `lambda x: x * 2` is the lambda function. Here `x` is the argument and `x * 2` is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier `double`. We can now call it as a normal function. The statement

```
double = lambda x: x * 2
```

is nearly the same as

```
def double(x):
    return x * 2
```

Use of Lambda Function in python

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like `filter()`, `map()` etc.

Example use with filter()

The `filter()` function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to `True`.

Here is an example use of `filter()` function to filter out only even numbers from a list.


```
# Program to filter out only the even items from a list

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(filter(lambda x: (x%2 == 0) , my_list))

# Output: [4, 6, 8, 12]
print(new_list)
```

Example use with map()

The map() function in Python takes in a function and a list.

The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Here is an example use of map() function to double all the items in a list.

```
# Program to double each item in a list using map()

my_list = [1, 5, 4, 6, 8, 11, 3, 12]

new_list = list(map(lambda x: x * 2 , my_list))

# Output: [2, 10, 8, 12, 16, 22, 6, 24]
print(new_list)
```

What are modules in Python?

Modules refer to a file containing Python statements and definitions.

A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.

We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

How to import modules in Python?

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

```
>>> import example
```

What are packages?

We don't usually store all of our files in our computer in the same location. We use a well-organized hierarchy of directories for easier access.

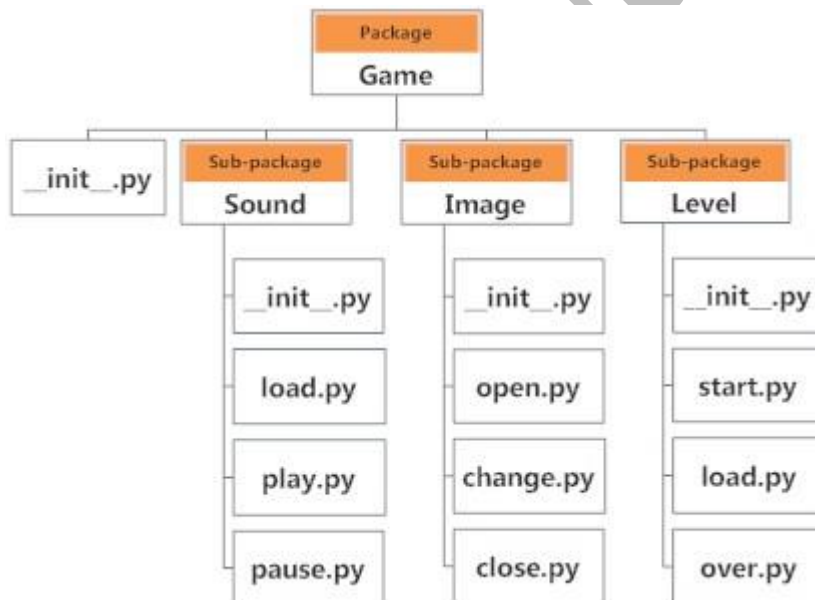
Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files.

As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.

A directory must contain a file named `__init__.py` in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.



Assignment

1. Which of the following is the use of function in python?
- a) Functions are reusable pieces of programs
 - b) Functions don't provide better modularity for your application
 - c) you can't also create your own functions
 - d) All of the mentioned

2. Which keyword is use for function?
- a) Fun
 - b) Define
 - c) Def
 - d) Function

3. What is the output of the below program?

```
def sayHello():  
    print('Hello World!')  
sayHello()  
sayHello()
```

- a) Hello World!
Hello World!
- b) 'Hello World!'
'Hello World!'
- c) Hello
Hello
- d) None of the mentioned

4. What is the output of the below program?

```
def printMax(a, b):  
    if a > b:  
        print(a, 'is maximum')  
    elif a == b:  
        print(a, 'is equal to', b)  
    else:  
        print(b, 'is maximum')  
printMax(3, 4)
```

- a) 3
- b) 4
- c) 4 is maximum
- d) None of the mentioned

5. What is the output of the below program ?

```
x = 50  
def func(x):  
    print('x is', x)  
    x = 2  
    print('Changed local x to', x)  
func(x)  
print('x is now', x)
```

- a) x is now 50
- b) x is now 2
- c) x is now 100
- d) None of the mentioned

6. What is the output of the below program?

```
x = 50
def func():
    global x
    print('x is', x)
    x = 2
    print('Changed global x to', x)
func()
print('Value of x is', x)
```

a) x is 50
Changed global x to 2
Value of x is 50
b) x is 50
Changed global x to 2
Value of x is 2
c) x is 50
Changed global x to 50
Value of x is 50
d) None of the mentioned

7. What is the output of below program?

```
def say(message, times = 1):
    print(message * times)
say('Hello')
say('World', 5)
```

a) Hello
WorldWorldWorldWorldWorld
b) Hello
World 5
c) Hello
World,World,World,World,World
d) Hello
HelloHelloHelloHelloHello

8. What is the output of the below program?

```
def func(a, b=5, c=10):
    print('a is', a, 'and b is', b, 'and c is', c)
```

func(3, 7)
func(25, c = 24)
func(c = 50, a = 100)

a) a is 7 and b is 3 and c is 10
a is 25 and b is 5 and c is 24
a is 5 and b is 100 and c is 50
b) a is 3 and b is 7 and c is 10
a is 5 and b is 25 and c is 24
a is 50 and b is 100 and c is 5

- c) a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
- d) None of the mentioned

9. What is the output of below program?

```
def maximum(x, y):  
    if x > y:  
        return x  
    elif x == y:  
        return 'The numbers are equal'  
    else:  
        return y
```

print(maximum(2, 3))

- a) 2
- b) 3
- c) The numbers are equal
- d) None of the mentioned

10. Which of the following is a feature of DocString?

- a) Provide a convenient way of associating documentation with Python modules, functions, classes, and methods
- b) All functions should have a docstring
- c) Docstrings can be accessed by the `__doc__` attribute on objects
- d) All of the mentioned

Date Time module in Python

In Python, date, time and datetime classes provides a number of function to deal with dates, times and time intervals. Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps. Whenever you manipulate dates or time, you need to import datetime function.

The datetime classes in Python are categorized into main 5 classes.

date – Manipulate just date (Month, day, year)

time – Time independent of the day (Hour, minute, second, microsecond)

datetime – Combination of time and date (Month, day, year, hour, second, microsecond)

timedelta— A duration of time used for manipulating dates

Get Current Date and Time

```
import datetime
datetime_object = datetime.datetime.now()
print(datetime_object)
```

When you run the program, the output will be something like:

2018-12-19 09:26:03.478039

Get Current Date

```
import datetime
nowdate = datetime.date.today()
print(nowdate)
```

When you run the program, the output will be something like:

2018-12-19

Commonly used classes in the datetime module are:

- date Class
- time Class
- datetime Class
- timedelta Class

datetime.timedelta

A timedelta object represents the difference between two dates or times.

```
from datetime import datetime, date
t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)
t3 = t1 - t2
print("t3 =", t3)
t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9, second = 33)
t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55, second = 13)
t6 = t4 - t5
print("t6 =", t6)
print("type of t3 =", type(t3))
print("type of t6 =", type(t6))
```

www.wscubetech.com

What is a file?

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

- Open a file
- Read or write (perform operation)
- Close the file

Mode	Description
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

How to open a file?

Python has a built-in function `open()` to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file and is done using Python **`close()`** method. Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

How to read files in Python?

To read a file in Python, we must open the file in reading mode.

There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

Python Directory and Files Management

If there are a large number of files to handle in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and sub directories. Python has the os module, which provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the getcwd() method.

This method returns the current working directory in the form of a string. We can also use the getcwdb() method to get it as bytes object.

```
>>> import os
>>> os.getcwd()
'C:\\Program Files\\PyScripter'
>>> os.getcwdb()
b'C:\\Program Files\\PyScripter'
```

The extra backslash implies escape sequence. The print() function will render this properly.

```
>>> print(os.getcwd())
C:\Program Files\PyScripter
```

Changing Directory

We can change the current working directory using the chdir() method.

The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements.

It is safer to use escape sequence when using the backward slash.

```
>>> os.chdir('C:\\Python33')
>>> print(os.getcwd())
C:\Python33
```

List Directories and Files

All files and sub directories inside a directory can be known using the `listdir()` method. This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

```
>>> print(os.getcwd())
```

```
C:\Python33
```

```
>>> os.listdir()
```

```
['DLLs',  
'Doc',  
'include',  
'Lib',  
'libs',  
'LICENSE.txt',  
'NEWS.txt',  
'python.exe',  
'pythonw.exe',  
'README.txt',  
'Scripts',  
'tcl',  
'Tools']
```

```
>>> os.listdir('G:\')
```

```
['$RECYCLE.BIN',  
'Movies',  
'Music',  
'Photos',  
'Series',  
'System Volume Information']
```

Making a New Directory

We can make a new directory using the `mkdir()` method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

```
>>> os.mkdir('test')
```

```
>>> os.listdir()
```

```
['test']
```

Renaming a Directory or a File

The `rename()` method can rename a directory or a file. The first argument is the old name and the new name must be supplied as the second argument.

```
>>> os.listdir()
```

```
['test']
```

```
>>> os.rename('test','new_one')
```

```
>>> os.listdir()
```

```
['new_one']
```

Removing Directory or File

A file can be removed (deleted) using the `remove()` method. Similarly, the `rmdir()` method removes an empty directory.

```
>>> os.listdir()
['new_one', 'old.txt']
```

```
>>> os.remove('old.txt')
>>> os.listdir()
['new_one']
```

```
>>> os.rmdir('new_one')
>>> os.listdir()
[]
```

However, note that `rmdir()` method can only remove empty directories. In order to remove a non-empty directory we can use the `rmtree()` method inside the `shutil` module.

```
>>> os.listdir()
['test']
```

```
>>> os.rmdir('test')
Traceback (most recent call last):
...
OSError: [WinError 145] The directory is not empty: 'test'
```

```
>>> import shutil
```

```
>>> shutil.rmtree('test')
>>> os.listdir()
[]
```

Python Errors and Built-in Exceptions

When writing a program, we, more often than not, will encounter errors. Error caused by not following the proper structure (syntax) of the language is called syntax error or parsing error.

```
>>> if a < 3
File "<interactive input>", line 1
  if a < 3
    ^
SyntaxError: invalid syntax
```

We can notice here that a colon is missing in the if statement.

Errors can also occur at runtime and these are called exceptions. They occur, for example, when a file we try to open does not exist (FileNotFoundError), dividing a number by zero (ZeroDivisionError), module we try to import is not found (ImportError) etc.

Whenever these type of runtime error occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Python Built-in Exceptions

Illegal operations can raise exceptions. There are plenty of built-in exceptions in Python that are raised when corresponding errors occur. We can view all the built-in exceptions using the local() built-in functions as follows.

```
>>> locals()['__builtins__']
```

This will return us a dictionary of built-in exceptions, functions and attributes.

Some of the common built-in exceptions in Python programming along with the error that cause then are tabulated below.

Python Built-in Exceptions

Exception	Base class for all exceptions
ArithmeticError	Raised when numeric calculations fails
FloatingPointError	Raised when a floating point calculation fails
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types
AssertionError	Raised when Assert statement fails
ImportError	Raised when the imported module is not found
IndexError	Raised when index of a sequence is out of range
IndentationError	Raised when there is incorrect indentation
SyntaxError	Raised by parser when syntax error is encountered
KeyError	Raised when the specified key is not found in the dictionary
NameError	Raised when an identifier is not found in the local or global namespace
TypeError	Raised when a function or operation is applied to an object of incorrect type
ValueError	Raised when a function gets argument of correct type but improper value
IOError	Raised when an input/ output operation fails
RuntimeError	Raised when a generated error does not fall into any category

Introduction to OOPs in Python

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

1. attributes
2. behaviour

Let's take an example:

Parrot is an object,

name, age, color are attributes
singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

Inheritance	A process of using details from a new class without modifying existing class.
Encapsulation	Hiding the private details of a class from other objects.
Polymorphism	A concept of using common operation in different ways for different data input.

Class

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

The example for class of parrot can be:

```
class Parrot:  
    pass
```

Here, we use class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, obj is object of class Parrot.

Suppose we have details of parrot. Now, we are going to show how to build the class and objects of parrot.

Example 1: Creating Class and Object in Python

```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

When we run program, the output will be:

```
Blu sings 'Happy'
Blu is now dancing
```

In the above program, we define two methods i.e sing() and dance(). These are called instance method because they are called on an instance object i.e blu.

Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

Example 3: Use of Inheritance in Python

```
# parent class
class Bird:

    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")

    def swim(self):
        print("Swim faster")

# child class
class Penguin(Bird):

    def __init__(self):
        # call super() function
        super().__init__()
        print("Penguin is ready")

    def whoisThis(self):
        print("Penguin")

    def run(self):
        print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

When we run this program, the output will be:

```
Bird is ready
Penguin is ready
Penguin
Swim faster
Run faster
```

In the above program, we created two classes i.e. Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from swim() method. Again, the child class modified the behavior of parent class. We can see this from whoisThis() method. Furthermore, we extend the functions of parent class, by creating a new run() method.

Additionally, we use `super()` function before `__init__()` method. This is because we want to pull the content of `__init__()` method from the parent class into the child class.

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “_” or double “__”.

Example 4: Data Encapsulation in Python

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

When we run this program, the output will be:

Selling Price: 900

Selling Price: 900

Selling Price: 1000

In the above program, we defined a class `Computer`. We use `__init__()` method to store the maximum selling price of computer. We tried to modify the price. However, we can't change it because Python treats the `__maxprice` as private attributes. To change the value, we used a setter function i.e `setMaxPrice()` which takes price as parameter.

Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

Example 5: Using Polymorphism in Python

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

When we run above program, the output will be:

Parrot can fly
Penguin can't fly

In the above program, we defined two classes Parrot and Penguin. Each of them have common method fly() method. However, their functions are different. To allow polymorphism, we created common interface i.e flying_test() function that can take any object. Then, we passed the objects blu and peggy in the flying_test() function, it ran effectively.

Key Points to Remember:

- The programming gets easy and efficient.
- The class is sharable, so codes can be reused.
- The productivity of programmers increases
- Data is safe and secure with data abstraction.

Python Objects and Class

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stress on objects.

Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called instantiation.

Defining a Class in Python

Like function definitions begin with the keyword `def`, in Python, we define a class using the keyword `class`. The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended. Here is a simple class definition.

```
class MyNewClass:  
    """This is a docstring. I have created a new class"""  
    pass
```

A class creates a new local namespace where all its attributes are defined. Attributes may be data or functions.

There are also special attributes in it that begins with double underscores (`__`). For example, `__doc__` gives us the docstring of that class.

As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

```
class MyClass:  
    """This is my second class"  
    a = 10  
    def func(self):  
        print('Hello')  
  
# Output: 10  
print(MyClass.a)  
  
# Output: <function MyClass.func at 0x0000000003079BF8>  
print(MyClass.func)  
  
# Output: 'This is my second class'  
print(MyClass.__doc__)
```

When you run the program, the output will be:

```
10
<function 0x7feaa932eae8="" at="" myclass.func="">
This is my second class
```

Creating an Object in Python

We saw that the class object could be used to access different attributes.

It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a function call.

```
>>> ob = MyClass()
```

This will create a new instance object named ob. We can access attributes of objects using the object name prefix.

Attributes may be data or method. Method of an object are corresponding functions of that class. Any function object that is a class attribute defines a method for objects of that class.

This means to say, since MyClass.func is a function object (attribute of class), ob.func will be a method object.

```
class MyClass:
    "This is my second class"
    a = 10
    def func(self):
        print('Hello')

# create a new MyClass
ob = MyClass()

# Output: <function MyClass.func at 0x000000000335B0D0>
print(MyClass.func)

# Output: <bound method MyClass.func of <__main__.MyClass object at
0x000000000332DEF0>>
print(ob.func)

# Calling function func()
# Output: Hello
ob.func()
```

Constructors in Python

Class functions that begins with double underscore (__) are called special functions as they have special meaning.

Of one particular interest is the `__init__()` function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

```
class ComplexNumber:
    def __init__(self,r = 0,i = 0):
        self.real = r
        self.imag = i

    def getData(self):
        print("{0}+{1}j".format(self.real,self.imag))

# Create a new ComplexNumber object
c1 = ComplexNumber(2,3)

# Call getData() function
# Output: 2+3j
c1.getData()

# Create another ComplexNumber object
# and create a new attribute 'attr'
c2 = ComplexNumber(5)
c2.attr = 10

# Output: (5, 0, 10)
print((c2.real, c2.imag, c2.attr))

# but c1 object doesn't have attribute 'attr'
# AttributeError: 'ComplexNumber' object has no attribute 'attr'
c1.attr
```

In the above example, we define a new class to represent complex numbers. It has two functions, `__init__()` to initialize the variables (defaults to zero) and `getData()` to display the number properly.

An interesting thing to note in the above step is that attributes of an object can be created on the fly. We created a new attribute `attr` for object `c2` and we read it as well. But this did not create that attribute for object `c1`.

Deleting Attributes and Objects

Any attribute of an object can be deleted anytime, using the `del` statement. Try the following on the Python shell to see the output.

```
>>> c1 = ComplexNumber(2,3)
```

```
>>> del c1.imag
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'imag'

>>> del ComplexNumber.getData
>>> c1.getData()
Traceback (most recent call last):
...
AttributeError: 'ComplexNumber' object has no attribute 'getData'
We can even delete the object itself, using the del statement.

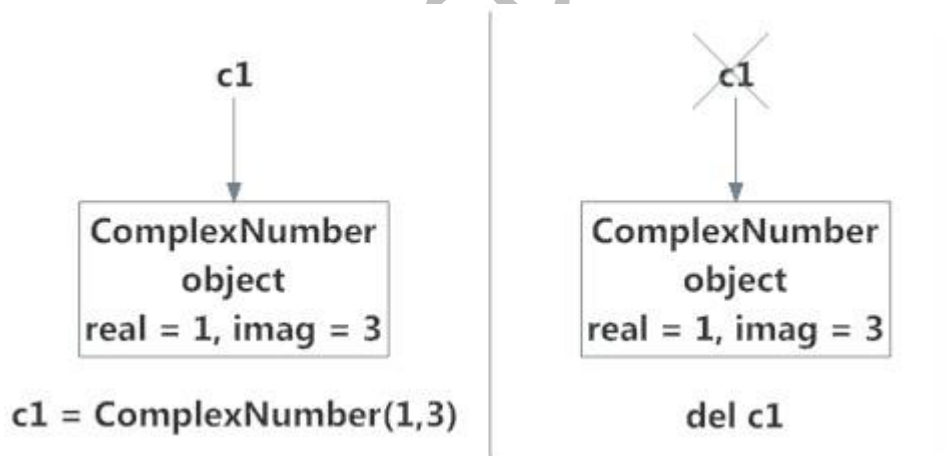
>>> c1 = ComplexNumber(1,3)
>>> del c1
>>> c1
```

Traceback (most recent call last):

```
...
NameError: name 'c1' is not defined
Actually, it is more complicated than that. When we do c1 = ComplexNumber(1,3), a new
instance object is created in memory and the name c1 binds with it.
```

On the command `del c1`, this binding is removed and the name `c1` is deleted from the corresponding namespace. The object however continues to exist in memory and if no other name is bound to it, it is later automatically destroyed.

This automatic destruction of unreferenced objects in Python is also called garbage collection.



Multiple Inheritance in Python

Like C++, a class can be derived from more than one base classes in Python. This is called multiple inheritance.

In multiple inheritance, the features of all the base classes are inherited into the derived class. The syntax for multiple inheritance is similar to single inheritance.

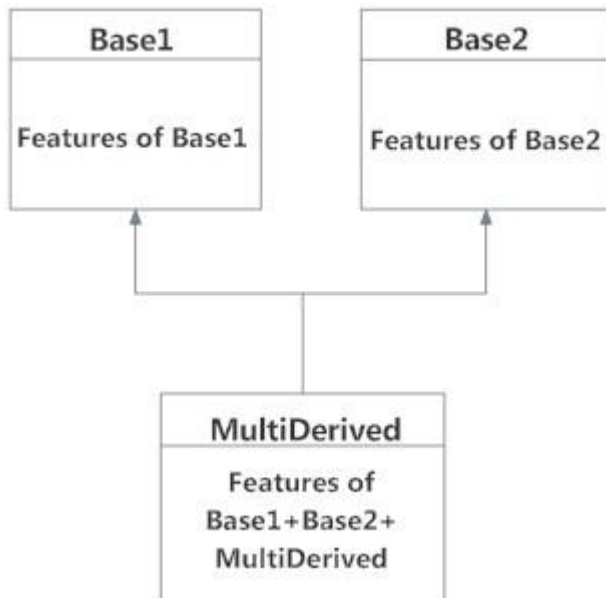
Example

```
class Base1:  
    pass
```

```
class Base2:  
    pass
```

```
class MultiDerived(Base1, Base2):  
    pass
```

Here, MultiDerived is derived from classes Base1 and Base2.



The class MultiDerived inherits from both Base1 and Base2.

Multilevel Inheritance in Python

On the other hand, we can also inherit from a derived class. This is called multilevel inheritance. It can be of any depth in Python.

In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

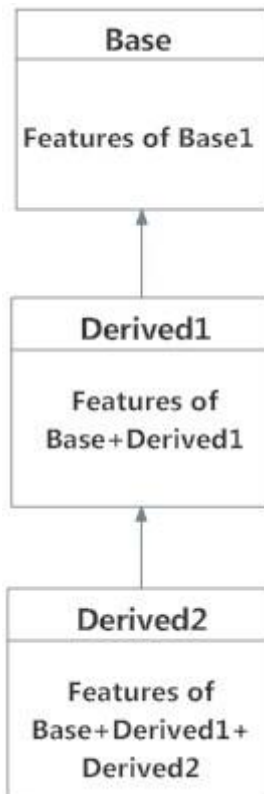
An example with corresponding visualization is given below.

```
class Base:  
    pass
```

```
class Derived1(Base):  
    pass
```

```
class Derived2(Derived1):  
    pass
```

Here, Derived1 is derived from Base, and Derived2 is derived from Derived1



Regular Expression

A regular expression in a programming language is a special text string used for describing a search pattern. It is extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.

While using the regular expression the first thing is to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters also referred as string. Ascii or latin letters are those that are on your keyboards and Unicode is used to match the foreign text. It includes digits and punctuation and all special characters like \$#@!%, etc.

For instance, a regular expression could tell a program to search for specific text from the string and then to print out the result accordingly. Expression can include

Text matching

Repetition

Branching

Pattern-composition etc.

In Python, a regular expression is denoted as RE (REs, regexes or regex pattern) are imported through re module. Python supports regular expression through libraries. In Python regular expression supports various things like Modifiers, Identifiers, and White space characters.

Identifiers	Modifiers	White space characters	Escape required
\d= any number (a digit)	\d represents a digit.Ex: \d{1,5} it will declare digit between 1,5 like 424,444,545 etc.	\n = new line	. + * ? [] \$ ^ () { } \
\D= anything but a number (a non-digit)	+ = matches 1 or more	\s= space	
\s = space (tab,space,newline etc.)	? = matches 0 or 1	\t =tab	

\S= anything but a space	* = 0 or more	\e = escape
\w = letters (Match alphanumeric character, including "_")	\$ match end of a string	\r = carriage return
\W =anything but letters (Matches a non-alphanumeric character excluding "_")	^ match start of a string	\f= form feed
. = anything but letters (periods)	matches either or x/y	-----
\b = any character except for new line	[] = range or "variance"	-----
\.	{x} = this amount of preceding code	-----

Regular Expression Syntax

- ```
import re
```
- "re" module included with Python primarily used for string searching and manipulation
  - Also used frequently for web page "Scraping" (extract large amount of data from websites)

We will begin the expression tutorial with this simple exercise by using the expressions (w+) and (^).

### Example of w+ and ^ Expression

"^": This expression matches the start of a string

"w+": This expression matches the alphanumeric character in the string

Here we will see an example of how we can use w+ and ^ expression in our code. We cover re.findall function later in this tutorial but for a while we simply focus on \w+ and \^ expression.

```
import re
xx = "guru99,education is fun"
r1 = re.findall(r"^\w+",xx)
print(r1)
```

Remember, if you remove +sign from the w+, the output will change, and it will only give the first character of the first letter, i.e., [g]

### Example of \s expression in re.split function

- "s": This expression is used for creating a space in the string

To understand how this regular expression works in Python, we begin with a simple example of a split function. In the example, we have split each word using the "re.split" function and at the same time we have used expression \s that allows to parse each word in the string separately.

When you execute this code it will give you the output ['we', 'are', 'splitting', 'the', 'words'].

Now, let see what happens if you remove "\" from s. There is no 's' alphabet in the output, this is because we have removed '\' from the string, and it evaluates "s" as a regular character and thus split the words wherever it finds "s" in the string.

Similarly, there are series of other regular expressions in Python that you can use in various ways in Python like \d,\D,\$,\.,\b, etc.

Here is the complete code

```
import re
xx = "guru99,education is fun"
r1 = re.findall(r"^\w+", xx)
print((re.split(r'\s','we are splitting the words')))
print((re.split(r's','split the words')))
```

Using regular expression methods

The "re" package provides several methods to actually perform queries on an input string. The method we going to see are

- re.match()
- re.search()
- re.findall()

Note: Based on the regular expressions, Python offers two different primitive operations. The match method checks for a match only at the beginning of the string while search checks for a match anywhere in the string.

### Using re.match()

The match function is used to match the RE pattern to string with optional flags. In this method, the expression "w+" and "\W" will match the words starting with letter 'g' and thereafter, anything which is not started with 'g' is not identified. To check match for each element in the list or string, we run the forloop.

www.wscubetech.com

## Database

### What is Data?

In simple words data can be facts related to any object in consideration.

For example your name, age, height, weight, etc are some data related to you.

A picture , image , file , pdf etc can also be considered data.

### What is a Database?

Database is a systematic collection of data. Databases support storage and manipulation of data. Databases make data management easy. Let's discuss few examples.

An online telephone directory would definitely use database to store data pertaining to people, phone numbers, other contact details, etc.

Your electricity service provider is obviously using a database to manage billing, client related issues, to handle fault data, etc.

Let's also consider the Facebook. It needs to store, manipulate and present data related to members, their friends, member activities, messages, advertisements and lot more.

### What is a Database Management System (DBMS)?

Database Management System (DBMS) is a collection of programs which enables its users to access database, manipulate data, reporting / representation of data .

It also helps to control access to the database.

Database Management Systems are not a new concept and as such had been first implemented in 1960s.

Charles Bachmen's Integrated Data Store (IDS) is said to be the first DBMS in history.

With time database technologies evolved a lot while usage and expected functionalities of databases have been increased immensely.

### What is MySQL?

MySQL is an open source relational database.

MySQL is cross platform which means it runs on a number of different platforms such as Windows, Linux, and Mac OS etc.

### Why use MySQL?

There are a number of relational database management systems on the market.

Examples of relational databases include Microsoft SQL Server, Microsoft Access, Oracle, DB2 etc.

One may ask why we would choose MySQL over the other database management systems.

The answer to this question depends on a number of factors.

### **Let's look at the strengths of MySQL compared to over relational databases such as SQL Server-**

- MySQL supports multiple storage engines each with its own specifications while other systems like SQL server only support a single storage engine. In order to appreciate this statement, let's look at two of the storage engines supported by MySQL.
  - InnoDB: - its default storage engine provided with MySQL as of version 5.5. InnoDB supports foreign keys for referential integrity and also supports ACID-standard transactions.
  - MyISAM: - it was the default storage engine for MySQL prior to version 5.5. MyISAM lacks support for transactions. Its advantages over InnoDB include simplicity and high performance.
- MySQL has high performance compared to other relation database systems. This is due to its simplicity in design and support for multiple-storage engines.
- Cost effective, it's relatively cheaper in terms of cost when compared to other relational databases. In fact, the community edition is free. The commercial edition has a licensing fee which is also cost effective compared to licensing fees for products such as Microsoft SQL Server.
- Cross platform - MySQL works on many platforms which means it can be deployed on most machines. Other systems such as MS SQL Server only run on the windows platform.

In order to interact with MySQL, you will need a server access tool that can communicate with MySQL server. MySQL supports multiple user connections.

### **MySQL Create Database, Tables, Data Types**

Steps for Create Database Mysql

Create Database in two ways

- 1) By executing a simple SQL query
- 2) By using forward engineering in MySQL Workbench

#### **Create Database**

CREATE DATABASE is the SQL command for creating a database.

Imagine you need to create a database with name "movies". You can do it by executing following SQL command.

**CREATE DATABASE movies;**

**Note:** you can also use the command CREATE SCHEMA instead of CREATE DATABASE  
Now let's improve our SQL query adding more parameters and specifications.

#### **IF NOT EXISTS**

A single MySQL server could have multiple databases. If you are not the only one accessing the same MySQL server or if you have to deal with multiple databases there is a probability

of attempting to create a new database with name of an existing database. IF NOT EXISTS let you to instruct MySQL server to check the existence of a database with a similar name prior to creating database.

When IF NOT EXISTS is used database is created only if given name does not conflict with an existing database's name. Without the use of IF NOT EXISTS MySQL throws an error.

### **CREATE DATABASE IF NOT EXISTS movies;**

#### **Collation and Character Set**

Collation is set of rules used in comparison. Many people use MySQL to store data other than English. Data is stored in MySQL using a specific character set. The character set can be defined at different levels viz, server , database , table and columns.

You need to select the rules of collation which in turn depend on the character set chosen.

For instance, the Latine1 character set uses the latin1\_swedish\_ci collation which is the Swedish case insensitive order.

### **CREATE DATABASE IF NOT EXISTS movies CHARACTER SET latin1 COLLATE latin1\_swedish\_ci**

The best practice while using local languages like Arabic , Chinese etc is to select Unicode (utf-8) character set which has several collations or just stick to default collation utf8-general-ci.

You can find the list of all collations and character sets here here

You can see list of existing databases by running following SQL command.

#### **SHOW DATABASES**

#### **Creating Tables MySQL**

Tables can be created using CREATE TABLE statement and it actually has the following syntax.

```
CREATE TABLE [IF NOT EXISTS] `TableName` (`fieldname` dataType
[optional parameters]) ENGINE = storage Engine;
HERE
```

"CREATE TABLE" is the one responsible for the creation of the table in the database.

"[IF NOT EXISTS]" is optional and only create the table if no matching table name is found.

"fieldName" is the name of the field and "data Type" defines the nature of the data to be stored in the field.

"[optional parameters]" additional information about a field such as " AUTO\_INCREMENT" , NOT NULL etc

#### **Create Table Example:-**

```
CREATE TABLE IF NOT EXISTS `MyFlixDB`.`Members` (
 `membership_number` INT AUTOINCREMENT,
 `full_names` VARCHAR(150) NOT NULL,
 `gender` VARCHAR(6),
 `date_of_birth` DATE,
 `physical_address` VARCHAR(255),
 `postal_address` VARCHAR(255),
 `contact_number` VARCHAR(75),
 `email` VARCHAR(255),
 PRIMARY KEY (`membership_number`))
```

Now let's see what the MySQL's data types are. You can use any of them depending on your need. You should always try to not to underestimate or overestimate potential range of data when creating a database.

## DATA TYPES

Data types define the nature of the data that can be stored in a particular column of a table

MySQL has 3 main categories of data types namely

- Numeric,
- Text
- Date/time.

### Numeric Data types

Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

|              |                                                       |
|--------------|-------------------------------------------------------|
| TINYINT( )   | -128 to 127 normal<br>0 to 255 UNSIGNED.              |
| SMALLINT( )  | -32768 to 32767 normal<br>0 to 65535 UNSIGNED.        |
| MEDIUMINT( ) | -8388608 to 8388607 normal<br>0 to 16777215 UNSIGNED. |

|              |                                                                                                       |
|--------------|-------------------------------------------------------------------------------------------------------|
| INT( )       | -2147483648 to 2147483647 normal<br>0 to 4294967295 UNSIGNED.                                         |
| BIGINT( )    | -9223372036854775808 to 9223372036854775807 normal<br>0 to 18446744073709551615 UNSIGNED.             |
| FLOAT        | A small approximate number with a floating decimal point.                                             |
| DOUBLE( , )  | A large number with a floating decimal point.                                                         |
| DECIMAL( , ) | A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values. |

### Text Data Types

As data type category name implies these are used to store text values. Always make sure you length of your textual data do not exceed maximum lengths.

|            |                                                          |
|------------|----------------------------------------------------------|
| CHAR( )    | A fixed section from 0 to 255 characters long.           |
| VARCHAR( ) | A variable section from 0 to 255 characters long.        |
| TINYTEXT   | A string with a maximum length of 255 characters.        |
| TEXT       | A string with a maximum length of 65535 characters.      |
| BLOB       | A string with a maximum length of 65535 characters.      |
| MEDIUMTEXT | A string with a maximum length of 16777215 characters.   |
| MEDIUMBLOB | A string with a maximum length of 16777215 characters.   |
| LONGTEXT   | A string with a maximum length of 4294967295 characters. |



|          |                                                          |
|----------|----------------------------------------------------------|
| LONGBLOB | A string with a maximum length of 4294967295 characters. |
|----------|----------------------------------------------------------|

## Date / Time

|           |                     |
|-----------|---------------------|
| DATE      | YYYY-MM-DD          |
| DATETIME  | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS      |
| TIME      | HH:MM:SS            |

## SQL SELECT statement syntax

It is the most frequently used SQL command and has the following general syntax

```
SELECT [DISTINCT|ALL] { * | [fieldExpression [AS newName]] FROM
tableName [alias] [WHERE condition][GROUP BY fieldName(s)] [HAVING
condition] ORDER BY fieldName(s)
```

- SELECT is the SQL keyword that lets the database know that you want to retrieve data.
- [DISTINCT | ALL] are optional keywords that can be used to fine tune the results returned from the SQL SELECT statement. If nothing is specified then ALL is assumed as the default.
- { \* | [fieldExpression [AS newName]] at least one part must be specified, "\*" selected all the fields from the specified table name, fieldExpression performs some computations on the specified fields such as adding numbers or putting together two string fields into one.
- FROM tableName is mandatory and must contain at least one table, multiple tables must be separated using commas or joined using the JOIN keyword.
- WHERE condition is optional, it can be used to specify criteria in the result set returned from the query.
- GROUP BY is used to put together records that have the same field values.
- HAVING condition is used to specify criteria when working using the GROUP BY keyword.
- ORDER BY is used to specify the sort order of the result set.

The Star symbol is used to select all the columns in table. An example of a simple SELECT statement looks like the one shown below.

```
SELECT * FROM `members`;
```

The above statement selects all the fields from the members table. The semi-colon is a statement terminate. It's not mandatory but is considered a good practice to end your statements like that.

## What is the WHERE Clause?

We looked at how to query data from a database using the SELECT statement in the previous tutorial. The SELECT statement returned all the results from the queried database table.

They are however, times when we want to restrict the query results to a specified condition. The SQL WHERE clause comes in handy in such situations.

## WHERE clause Syntax

The basic syntax for the WHERE clause when used in a SELECT statement is as follows.

```
SELECT * FROM tableName WHERE condition;
```

HERE

- "SELECT \* FROM tableName" is the standard SELECT statement
- "WHERE" is the keyword that restricts our select query result set and "condition" is the filter to be applied on the results. The filter could be a range, single value or sub query.

Let's now look at a practical example.

Suppose we want to get a member's personal details from members table given the membership number 1, we would use the following script to achieve that.

```
SELECT * FROM `members` WHERE `membership_number` = 1;
```

## WHERE clause combined with - AND LOGICAL Operator

The WHERE clause when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.

Let's now look at a practical example - Suppose we want to get a list of all the movies in category 2 that were released in 2008, we would use the script shown below is achieve that.

```
SELECT * FROM `movies` WHERE `category_id` = 2 AND `year_released` = 2008;
```

### **WHERE clause combined with - OR LOGICAL Operator**

The WHERE clause when used together with the OR operator, is only executed if any or the entire specified filter criteria is met.

The following script gets all the movies in either category 1 or category 2

```
SELECT * FROM `movies` WHERE `category_id` = 1 OR `category_id` = 2;
```

### **WHERE clause combined with - IN Keyword**

The WHERE clause when used together with the IN keyword only affects the rows whose values matches the list of values provided in the IN keyword. IN helps reduces number of OR clauses you may have to use

The following query gives rows where membership\_number is either 1 , 2 or 3

```
SELECT * FROM `members` WHERE `membership_number` IN (1,2,3);
```

### **WHERE clause combined with - NOT IN Keyword**

The WHERE clause when used together with the NOT IN keyword DOES NOT affects the rows whose values matches the list of values provided in the NOT IN keyword.

The following query gives rows where membership\_number is NOT 1 , 2 or 3

```
SELECT * FROM `members` WHERE `membership_number` NOT IN (1,2,3);
```

### **WHERE clause combined with - COMPARISON OPERATORS**

The less than (<), equal to (=), not equal to (<>) comparison operators can be used with the Where clause

#### **= Equal To**

The following script gets all the female members from the members table using the equal to comparison operator.

```
SELECT * FROM `members` WHERE `gender` = 'Female';
```

#### **> Greater than**

The following script gets all the payments that are greater than 2,000 from the payments table.

```
SELECT * FROM `payments` WHERE `amount_paid` > 2000;
```

#### **< > Not Equal To**

The following script gets all the movies whose category id is not 1.

```
SELECT * FROM `movies` WHERE `category_id` <> 1;
```

### **What is INSERT INTO?**

The main goal of database systems is to store data in the tables. The data is usually supplied by application programs that run on top of the database. Towards that end, SQL has the INSERT command that is used to store data into a table. The INSERT command creates a new row in the table to store data.

#### Basic syntax

Let's look at the basic syntax of the SQL INSERT command shown below.

**INSERT INTO `table\_name` (column\_1,column\_2,...) VALUES (value\_1,value\_2,...);**  
HERE

- INSERT INTO `table\_name` is the command that tells MySQL server to add new row into a table named `table\_name`.
- (column\_1,column\_2,...) specifies the columns to be updated in the new row
- VALUES (value\_1,value\_2,...) specifies the values to be added into the new row

When supplying the data values to be inserted into the new table, the following should be considered while dealing with different data types.

- String data types - all the string values should be enclosed in single quotes.
- Numeric data types - all numeric values should be supplied directly without enclosing them in single or double quotes.
- Date data types - enclose date values in single quotes in the format 'YYYY-MM-DD'.

## What is the DELETE Keyword?

The SQL DELETE command is used to delete rows that are no longer required from the database tables. It deletes the whole row from the table. Delete command comes in handy to delete temporary or obsolete data from your database. The DELETE command can delete more than one row from a table in a single query. This proves to be advantages when removing large numbers of rows from a database table.

Once a row has been deleted, it cannot be recovered. It is therefore strongly recommended to make database backups before deleting any data from the database. This can allow you to restore the database and view the data later on should it be required.

#### Delete command syntax

The basic syntax of the delete command is as shown below.

**DELETE FROM `table\_name` [WHERE condition];**  
HERE

- DELETE FROM `table\_name` tells MySQL server to remove rows from the table ..
- [WHERE condition] is optional and is used to put a filter that restricts the number of rows affected by the DELETE query.

## ORDER BY in MySQL: DESC & ASC

#### Sorting Results

We looked at how to get data from our tables using the SELECT command. Results were returned in the same order the records were added into the database. This is the default sort order. In this section, we will be looking at how we can sort our query results. Sorting is

simply re-arranging our query results in a specified way. Sorting can be performed on a single column or on more than one column. It can be done on number, strings as well as date data types.

### Order by clause

The order by clause is used to sort the query result sets in either ascending or descending order. It is used in conjunction with the SELECT query. It has the following basic syntax.

**SELECT statement... [WHERE condition | GROUP BY `field\_name(s)` HAVING condition] ORDER BY `field\_name(s)` [ASC | DESC];**

HERE

- "SELECT statement..." is the regular select query
- " | " represents alternatives
- "[WHERE condition | GROUP BY `field\_name(s)` HAVING condition]" is the optional condition used to filter the query result sets.
- "ORDER BY" performs the query result set sorting
- "[ASC | DESC]" is the keyword used to sort result sets in either ascending or descending order. Note ASC is used as the default.

### DESC and ASC syntax

The DESC sort keyword has the following basic syntax.

**SELECT {fieldName(s) | \*} FROM tableName(s) [WHERE condition] ORDER BY filename(s) ASC /DESC [LIMIT N]**

HERE

- SELECT {fieldName(s) | \*} FROM tableName(s) is the statement containing the fields and table(s) from which to get the result set from.
- [WHERE condition] is optional but can be used to filter the data according to the given condition.
- ORDER BY filename(s) is mandatory and is the field on which the sorting is to be performed. The DESC keyword specifies that the sorting is to be in descending order.
- [LIMIT] is optional but can be used to limit the number of results returned from the query result set.

### What is the Group by Clause?

The GROUP BY clause is a SQL command that is used to group rows that have the same values.

The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

That's what it does, summarizing data from the database.

The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

## GROUP BY Syntax

Now that we know what the GROUP BY clause is, let's look at the syntax for a basic group by query.

**SELECT statements... GROUP BY column\_name1[,column\_name2,...] [HAVING condition];**

HERE

- "SELECT statements..." is the standard SQL SELECT command query.
- "GROUP BY column\_name1" is the clause that performs the grouping based on column\_name1.
- "[,column\_name2,...]" is optional; represents other column names when the grouping is done on more than one column.
- "[HAVING condition]" is optional; it is used to restrict the rows affected by the GROUP BY clause. It is similar to the WHERE clause.
- Grouping using a Single Column

In order to help understand the effect of Group By clause, let's execute a simple query that returns all the gender entries from the members table.

**SELECT `gender` FROM `members` ;**

## What are wildcards?

Wildcards are characters that help search data matching complex criteria. Wildcards are used in conjunction with the LIKE comparison operator or the NOT LIKE comparison operator.

### Why use WildCards ?

If you are familiar with using the SQL, you may think that you can search for any complex data using SELECT and WHERE clause . Then why use Wildcards ?

Before we answer that question, let's look at an example. Suppose that the marketing department of Myflix video library carried out marketing promotions in the city of Texas and would like to get some feedback on the number of members

that registered from Texas, you can use the following SELECT statement together with the WHERE clause to get the desired information.

**SELECT \* FROM members WHERE postal\_address = 'Austin , TX' OR postal\_address = Dallas , TX OR postal\_address = lola,TX OR postal\_adress = Houston ,TX';**

As you can see from the above query, the "WHERE clause" becomes complex. Using wildcards however, simplifies the query as we can use something simple like the script shown below.

**SELECT \* FROM members WHERE postal\_address like '% TX';**

In short, wildcards allow us to develop power search engines into our data driven applications.

## Types of wildcards

% the percentage

% the percentage character is used to specify a pattern of zero (0) or more characters. It has the following basic syntax.

**SELECT statements... WHERE fieldname LIKE 'xxx%';**  
**HERE**

- "SELECT statement..." is the standard SQL SELECT command.
- "WHERE" is the key word used to apply the filter.
- "LIKE" is the comparison operator that is used in conjunction with wildcards
- 'xxx' is any specified starting pattern such as a single character or more and "%" matches any number of characters starting from zero (0).

To fully appreciate the above statement, let's look at a practical example

Suppose we want to get all the movies that have the word "code" as part of the title, we would use the percentage wildcard to perform a pattern match on both sides of the word "code". Below is the SQL statement that can be used to achieve the desired results.

**SELECT \* FROM movies WHERE title LIKE '%code%';**  
**\_ underscore wildcard**

The underscore wildcard is used to match exactly one character. Let's suppose that we want to search for all the movies that were released in the years 200x where x is exactly one character that could be any value. We would use the underscore wild card to achieve that. The script below select all the movies that were released in the year "200x"

**SELECT \* FROM movies WHERE year\_released LIKE '200\_';**

## NOT Like

The NOT logical operator can be used together with the wildcards to return rows that do not match the specified pattern.

Suppose we want to get movies that were not released in the year 200x. We would use the NOT logical operator together with the underscore wildcard to get our results. Below is the script that does that.

**SELECT \* FROM movies WHERE year\_released NOT LIKE '200\_';**

## What are functions?

MySQL can do much more than just store and retrieve data. We can also perform manipulations on the data before retrieving or saving it. That's where MySQL Functions come in. Functions are simply pieces of code that perform some operations and then return a result. Some functions accept parameters while other functions do not accept parameters.

Let's briefly look at an example of MySQL function. By default, MySQL saves date data types in the format "YYYY-MM-DD". Suppose we have built an application and our users want the date to be returned in the format "DD-MM-YYYY", we can use MySQL built in function DATE\_FORMAT to achieve this. DATE\_FORMAT is one of the most used functions in MySQL. We will look at it in more details as we unfold the lesson.



## Why use functions?

Based on the example given in the introduction, people with experience in computer programming may be thinking "Why bother MySQL Functions? The same effect can be achieved with scripting/programming language?" It's true we can achieve that by writing some procedures/function in the application program.

Getting back to our DATE example in the introduction, for our users to get the data in the desired format, business layer will have to do necessary processing.

This becomes a problem when the application has to integrate with other systems. When we use MySQL functions such as the DATE\_FORMAT, then we can have that functionality embedded into the database and any application that needs the data gets it in the required format. This reduces re-work in the business logic and reduce data inconsistencies.

Another reason why we should consider using MySQL functions is the fact that it can help reducing network traffic in client/server applications. Business Layer will only need to make call to the stored functions without the need manipulate data .On average, the use of functions can help greatly improve overall system performance.

## Types of functions

### Built-in functions

MySQL comes bundled with a number of built in functions. Built in functions are simply functions come already implemented in the MySQL server. These functions allow us to perform different types of manipulations on the data. The built in functions can be basically categorized into the following most used categories.

- Strings functions - operate on string data types
- Numeric functions - operate on numeric data types
- Date functions - operate on date data types
- Aggregate functions - operate on all of the above data types and produce summarized result sets.
- Other functions - MySQL also supports other types of built in functions but we will limit our lesson to the above named functions only.

Let's now look at each of the functions mentioned above in detail. We will be explaining the most used functions using our "Myflixdb".

### String functions

We already looked at what string functions do. We will look at a practical example that uses them. In our movies table, the movie titles are stored using combinations of lower and upper case letters. Suppose we want to get a query list that returns the movie titles in upper case letters. We can use the "UCASE" function to do that. It takes a string as a parameter and converts all the letters to upper case. The script shown below demonstrates the use of the "UCASE" function.

```
SELECT `movie_id`, `title`, UCASE(`title`) FROM `movies`;
```



HERE

- UCASE(`title`) is the built in function that takes the title as a parameter and returns it in upper case letters with the alias name `upper\_case\_title`.

## Numeric functions

As earlier mentioned, these functions operate on numeric data types. We can perform mathematic computations on numeric data in the SQL statements.

### Arithmetic operators

MySQL supports the following arithmetic operators that can be used to perform computations in the SQL statements.

| Name     | Description      |
|----------|------------------|
| DIV      | Integer division |
| /        | Division         |
| -        | Subtraction      |
| +        | Addition         |
| *        | Multiplication   |
| % or MOD | Modulus          |

#### Integer Division (DIV)

**SELECT 23 DIV 6 ;**

Executing the above script gives us the following results.

3

#### Division operator (/)

Let's now look at the division operator example. We will modify the DIV example.

**SELECT 23 / 6 ;**

Executing the above script gives us the following results.

3.8333

#### Subtraction operator (-)

Let's now look at the subtraction operator example. We will use the same values as in the previous two examples

**SELECT 23 - 6 ;**

Executing the above script gives us 17

### Addition operator (+)

Let's now look at the addition operator example. We will modify the previous example.

**SELECT 23 + 6 ;**

Executing the above script gives us 29

## Aggregate Functions are all about

Performing calculations on multiple rows

Of a single column of a table

And returning a single value.

The ISO standard defines five (5) aggregate functions namely;

- 1) COUNT
- 2) SUM
- 3) AVG
- 4) MIN
- 5) MAX

### Why use aggregate functions.

From a business perspective, different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details.

>Aggregate functions allow us to easily produce summarized data from our database. For instance, from our myflix database , management may require following reports

- Least rented movies.
- Most rented movies.
- Average number that each movie is rented out in a month.

We easily produce above reports using aggregate functions.

Let's look into aggregate functions in detail.

### COUNT Function

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. All aggregate functions by default exclude nulls values before working on the data.

**COUNT (\*)** is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (\*) also considers Nulls and duplicates.

Let's suppose that we want to get the number of times that the movie with id 2 has been rented out

**SELECT COUNT(`movie\_id`) FROM `movierentals` WHERE `movie\_id` = 2;**

Executing the above query in MySQL workbench against myflixdb gives us the following results.

**COUNT('movie\_id')**

3

**MIN function**

The MIN function returns the smallest value in the specified table field.

As an example, let's suppose we want to know the year in which the oldest movie in our library was released, we can use MySQL's MIN function to get the desired information.

The following query helps us achieve that

```
SELECT MIN(`year_released`) FROM `movies`;
```

**MAX function**

Just as the name suggests, the MAX function is the opposite of the MIN function. It returns the largest value from the specified table field.

Let's assume we want to get the year that the latest movie in our database was released. We can easily use the MAX function to achieve that.

The following example returns the latest movie year released.

```
SELECT MAX(`year_released`) FROM `movies`;
```

**AVG function**

MySQL AVG function returns the average of the values in a specified column. Just like the SUM function, it works only on numeric data types.

Suppose we want to find the average amount paid. We can use the following query -

```
SELECT AVG(`amount_paid`) FROM `payments`;
```

**WHAT IS THE ALTER COMMAND?**

As the saying goes Change is the only constant

With time business requirements change as well. As business requirements change, Database designs need changing as well.

MySQL provides the ALTER function that helps us incorporate the changes to the already existing database design.

The alter command is used to modify an existing database, table, view or other database objects that might need to change during the life cycle of a database.

Let's suppose that we have completed our database design and it has been implemented. Our database users are using it and then they realize some of the vital information was left out in the design phase. They don't want to lose the existing data but just want to incorporate the new information. The alter command comes in handy in such situations. We can use the alter command to change the data type of a field from say string to numeric, change the field name to a new name or even add a new column in a table.

**Alter- syntax**

The basic syntax used to add a column to an already existing table is shown below

**ALTER TABLE `table\_name` ADD COLUMN `column\_name` `data\_type`;**

HERE

- "ALTER TABLE `table\_name`" is the command that tells MySQL server to modify the table named `table\_name`.
- "ADD COLUMN `column\_name` `data\_type`" is the command that tells MySQL server to add a new column named `column\_name` with data type `data\_type`.

Let's suppose that Myflix has introduced online billing and payments. Towards that end, we have been asked to add a field for the credit card number in our members table. We can use the ALTER command to do that. Let's first look at the structure of the members table before we make any amendments. The script shown below helps us to do that.

**SHOW COLUMNS FROM `members`;**

### **Multiplication operator (\*)**

Let's now look at the multiplication operator example. We will use the same values as in the previous examples.

**SELECT 23 \* 6 AS `multiplication\_result`;**

Executing the above script gives us the following results.

**multiplication\_result**

138

### **Modulo operator (-)**

The modulo operator divides N by M and gives us the remainder. Let's now look at the modulo operator example. We will use the same values as in the previous examples.

**SELECT 23 % 6 ;**

OR

**SELECT 23 MOD 6 ;**

Executing the above script gives us 5

Let's now look at some of the common numeric functions in MySQL.

Floor - this function removes decimal places from a number and rounds it to the nearest lowest number. The script shown below demonstrates its usage.

**SELECT FLOOR(23 / 6) AS `floor\_result`;**

Executing the above script gives us the following results.

**Floor\_result**

3

Round - this function rounds a number with decimal places to the nearest whole number. The script shown below demonstrates its usage.

**SELECT ROUND(23 / 6) AS `round\_result`;**

Executing the above script gives us the following results.

#### **Round\_result**

4

**Rand** - this function is used to generate a random number, its value changes every time that the function is called. The script shown below demonstrates its usage.

```
SELECT RAND() AS `random_result`;
```

## **GUI Development in Python**

### **What Is A Graphical User Interface(GUI)**

GUI is a desktop app which helps you to interact with the computers. They are used to perform different tasks in the desktops, laptops, other electronic devices, etc..., Here, we mainly talking about the laptops and desktops.

- GUI apps like Text-Editors are used to create, read, update and delete different types of files.
- GUI apps like Sudoku, Chess, Solitaire, etc..., are games which you can play.
- GUI apps like Chrome, Firefox, Microsoft Edge, etc..., are used to surf the Internet.

They are some different types of GUI apps which we daily use on the laptops or desktops. We are going to learn how to create those type of apps.

As this is an Introduction to GUI, we will create a simple Calculator GUI app.

### **What Is Tkinter**

Tkinter is an inbuilt Python module used to create simple GUI apps. It is the most commonly used module for GUI apps in the Python.

You don't need to worry about installation of the Tkinter module as it comes with Python default.

### **Introduction To Tkinter**

Run the following code to create a simple window with the text Hello World!.

#### **Steps:-**

- import the module tkinter.
- Initialize the window manager with the tkinter.Tk() method and assign it to a variable window. This method creates a blank window with close, maximize and minimize buttons.
- Rename the title of the window as you like with the window.title(title\_of\_the\_window).
- Label is used to insert some objects into the window. Here, we are adding a Label with some text.
- pack() attribute of the widget is used to display the widget in a size it requires.
- Finally, the mainloop() method to display the window until you manually close it.

```
import tkinter

window = tkinter.Tk()
to rename the title of the window
window.title("GUI")
pack is used to show the object in the window
label = tkinter.Label(window, text = "Hello World!").pack()
window.mainloop()
```

## Tkinter Widgets

Widgets are something like elements in the HTML. You will find different types of widgets to the different types of elements in the Tkinter.

Let's see the brief introduction to all of these widgets in the Tkinter.

- Button:- Button widget is used to place the buttons in the tkinter.
- Canvas:- Canvas is used to draw shapes in your GUI.
- Checkbutton:- Checkbutton is used to create the check buttons in your application. You can select more than one option at a time.
- Entry:- Entry widget is used to create input fields in the GUI.
- Frame:- Frame is used as containers in the tkinter.
- Label:- Label is used to create a single line widgets like text, images, etc..,
- Menu:- Menu is used to create menus in the GUI.

## Geometry Management

All widgets in the tkinter will have some geometry measurements. These measurements give you to organize the widgets and their parent frames, windows, etc..,

Tkinter has the following three Geometry Manager classes.

- pack():- It organizes the widgets in the block, which mean it occupies the entire available width. It's a standard method to show the widgets in the window
- grid():- It organizes the widgets in table-like structure. You will see details about grid later in this tutorial.
- place():- It's used to place the widgets at a specific position you want.

## Organizing Layout And Widgets

To arrange the layout in the window, we will use Frame, class. Let's create a simple program to see how the Frame works.

### Steps:-

- Frame is used to create the divisions in the window. You can align the frames as you like with side parameter of pack() method.
- Button is used to create a button in the window. It takes several parameters like text(Value of the Button), fg(Color of the text), bg(Background color), etc..,

**Note:-** The parameter of any widget method must be where to place the widget. In the below code, we use to place in the window, top\_frame, bottom\_frame.

```
import tkinter

window = tkinter.Tk()
window.title("GUI")

creating 2 frames TOP and BOTTOM
top_frame = tkinter.Frame(window).pack()
bottom_frame = tkinter.Frame(window).pack(side = "bottom")

now, create some widgets in the top_frame and bottom_frame
btn1 = tkinter.Button(top_frame, text = "Button1", fg = "red").pack()# 'fg - foreground' is used to
color the contents
btn2 = tkinter.Button(top_frame, text = "Button2", fg = "green").pack()# 'text' is used to write
the text on the Button
btn3 = tkinter.Button(bottom_frame, text = "Button2", fg = "purple").pack(side = "left")# 'side' is
used to align the widgets
btn4 = tkinter.Button(bottom_frame, text = "Button2", fg = "orange").pack(side = "left")

window.mainloop()
```

```
import tkinter

window = tkinter.Tk()
window.title("GUI")

creating 3 simple Labels containing any text

sufficient width
tkinter.Label(window, text = "Suf. width", fg = "white", bg = "purple").pack()
width of X
tkinter.Label(window, text = "Taking all available X width", fg = "white", bg = "green").pack(fill =
"x")

height of Y
tkinter.Label(window, text = "Taking all available Y height", fg = "white", bg = "black").pack(side
= "left", fill = "y")
window.mainloop()
```

## Binding Functions

Calling functions whenever an event occurs refers to a binding function.

- In the below example, when you click the button, it calls a function called say\_hi.
- Function say\_hi creates a new label with the text Hi.

```
import tkinter

window = tkinter.Tk()
window.title("GUI")

creating a function called say_hi()
def say_hi():
 tkinter.Label(window, text = "Hi").pack()

tkinter.Button(window, text = "Click Me!", command = say_hi).pack() # 'command' is
executed when you click the button # in this above case we're calling the function
'say_hi'.
window.mainloop()
```

## Mouse Clicking Events

Clicking events are of 3 different types namely leftClick, middleClick, and rightClick. Now, you will learn how to call a particular function based on the event that occurs.

Run the following program and click the left, middle, right buttons to calls a specific function. That function will create a new label with the mentioned text.



```
import tkinter

window = tkinter.Tk()
window.title("GUI")

#creating 3 different functions for 3 events
def left_click(event):
 tkinter.Label(window, text = "Left Click!").pack()

def middle_click(event):
 tkinter.Label(window, text = "Middle Click!").pack()

def right_click(event):
 tkinter.Label(window, text = "Right Click!").pack()

window.bind("<Button-1>", left_click)
window.bind("<Button-2>", middle_click)
window.bind("<Button-3>", right_click)

window.mainloop()
```

## Drop-Down Menus

I hope all of you know what drop-down menus are. You will create drop-down menus in tkinter using the class Menu. Follow the below steps to create drop-down menus.

### Steps:-

- Create a root menu to insert different types of menu options using `tkinter.Menu(para)` and it takes a parameter where to place the Menu
- You have to tell the tkinter to initiate Menu using `window_variable.config(menu = para)` and it takes a parameter called menu which is the root menu you previously defined.
- Now, creating sub menus using same method `tkinter.Menu(para)` and it takes the parameter root menu.
- `root menu.add_cascade(para1, menu = para2)` creates the name of the sub menu, and it takes 2 parameters one is label which is the name of the sub menu, and another one is menu which is sub menu.
- `sub menu.add_command()` adds an option to the sub menu.
- `sub menu.add_separator()` adds a separator

```
import tkinter

window = tkinter.Tk()
window.title("GUI")

def function():
 pass

creating a root menu to insert all the sub menus
root_menu = tkinter.Menu(window)
window.config(menu = root_menu)

creating sub menus in the root menu
file_menu = tkinter.Menu(root_menu) # it initializes a new sub menu in the root menu
root_menu.add_cascade(label = "File", menu = file_menu) # it creates the name of the sub menu
file_menu.add_command(label = "New file.....", command = function) # it adds a option to the sub menu 'command' parameter is used to do some action
file_menu.add_command(label = "Open files", command = function)
file_menu.add_separator() # it adds a line after the 'Open files' option
file_menu.add_command(label = "Exit", command = window.quit)

creating another sub menu
edit_menu = tkinter.Menu(root_menu)
root_menu.add_cascade(label = "Edit", menu = edit_menu)
edit_menu.add_command(label = "Undo", command = function)
edit_menu.add_command(label = "Redo", command = function)

window.mainloop()
```

### Alert Box

You can create alert boxes in the tkinter using messagebox method. You can also create questions using the messagebox method.

```
import tkinter
import tkinter.messagebox

window = tkinter.Tk()
window.title("GUI")

creating a simple alert box
tkinter.messagebox.showinfo("Alert Message", "This is just a alert message!")
creating a question to get the response from the user [Yes or No Question]
response = tkinter.messagebox.askquestion("Simple Question", "Do you love Python?")
If user clicks 'Yes' then it returns 1 else it returns 0
if response == 1:
 tkinter.Label(window, text = "You love Python!").pack()
else:
 tkinter.Label(window, text = "You don't love Python!").pack()

window.mainloop()
```

## Simple Shapes

You are going to draw some basic shapes with the Canvas provided by tkinter in GUI.

```
import tkinter

window = tkinter.Tk()
window.title("GUI")
canvas = tkinter.Canvas(window, width = 500, height = 500)
canvas.pack()
line1 = canvas.create_line(25, 25, 250, 150)
parameter:- (fill = color_name)
line2 = canvas.create_line(25, 250, 250, 150, fill = "red")

'create_rectangle' is used to create rectangle. Parameters:- (starting x-point, starting y-point, width, height, fill)
starting point the coordinates of top-left point of rectangle
rect = canvas.create_rectangle(500, 25, 175, 75, fill = "green")

you 'delete' shapes using delete method passing the name of the variable as parameter.
canvas.delete(line1)
you 'delete' all the shapes by passing 'ALL' as parameter to the 'delete' method
canvas.delete(tkinter.ALL)

window.mainloop()
```

## Images And Icons

You can add Images and Icons using PhotoImage method.

```
import tkinter

window = tkinter.Tk()
window.title("GUI")

taking image from the directory and storing the source in a variable
icon = tkinter.PhotoImage(file = "images/haha.png")
displaying the picture using a 'Label' by passing the 'picture' variable to 'image' parameter
label = tkinter.Label(window, image = icon)
label.pack()

window.mainloop()
```

## Python Assignments

### Python Basic

1. Write a Python program which accepts the radius of a circle from the user and compute the area
2. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.
3. Write a Python program that accepts an integer (n) and computes the value of  $n+nn+nnn$ .
4. Write a Python program to get the volume of a sphere with radius 6
5. Write a Python program to find whether a given number (accept from the user) is even or odd, print out an appropriate message to the user
6. Write a Python program to test whether a passed letter is a vowel or not.
7. Write a Python program to check whether a specified value is contained in a group of values.
8. Write a Python program that will accept the base and height of a triangle and compute the area
9. Write a Python program to compute the greatest common divisor (GCD) of two positive integers.
10. Write a Python program to get the least common multiple (LCM) of two positive integers.
11. Write a Python program to sum of three given integers. However, if two values are equal sum will be zero.
12. Write a Python program to sum of two given integers. However, if the sum is between 15 to 20 it will return 20.
13. Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.
14. Write a Python program to display your details like name, age, and address in three different lines.
15. Write a Python program to solve  $(x + y) * (x + y)$ .
16. Write a Python program to compute the future value of a specified principal amount, rate of interest, and a number of years.
17. Write a Python program to compute the distance between the points (x1, y1) and (x2, y2).
18. Write a python program to sum of the first n positive integers.
19. Write a Python program to convert height (in feet and inches) to centimetres.
20. Write a Python program to calculate the hypotenuse of a right angled triangle.
21. Write a Python program to convert the distance (in feet) to inches, yards, and miles.
22. Write a Python program to calculate body mass index.
23. Write a Python program to convert pressure in kilopascals to pounds per square inch, a millimetre of mercury (mmHg) and atmosphere pressure.
24. Write a Python program to calculate the sum of the digits in an integer.
25. Write a Python program to sort three integers without using conditional statements and loops.
26. Write a Python program to concatenate N strings.
27. Write a Python program to get the ASCII value of a character.
28. Write a Python program to swap two variables.
29. Write a Python program to check if a string is numeric.
30. Write a Python program to check if a number is positive, negative or zero

31. Write a Python programme to check whether a number is divisible by another number. Accept two integer's values form the user.
32. Write a Python program to get the Fibonacci series between 0 to 50.
33. Write a Python program that accepts a string and calculate the number of digits and letters.
34. Write a Python program to find numbers between 100 and 400 (both included) where each digit of a number is an even number. The numbers obtained should be printed in a comma-separated sequence.
35. Write a Python program to check whether an alphabet is a vowel or consonant.
36. Write a Python program to sum of two given integers. However, if the sum is between 15 to 20 it will return 20.
37. Write a Python program to calculate the sum and average of n integer numbers (input from the user). Input 0 to finish.
38. Write a Python program to calculate the length of a string
39. Write a Python program to count the number of characters (character frequency) in a string
40. Write a Python program to count the occurrences of each word in a given sentence
41. Write a Python program to find those numbers which are divisible by 7 and multiple of 5, between 1500 and 2700 (both included)
42. Write a Python program to convert temperatures to and from Celsius, Fahrenheit.
43. Write a Python program to count the number of even and odd numbers from a series of numbers
44. Write a Python program that accepts a word from the user and reverse it
45. Write a Python program that prints each item and its corresponding type from the following list.

Sample List : datalist = [1452, 11.23, 1+2j, True, 'w3resource', (0, -1), [5, 12], {'class': 'V', 'section': 'A'}]

46. Write a Python program to get the Fibonacci series between 0 to 50.
47. Write a Python program to find numbers between 100 and 400 (both included) where each digit of a number is an even number. The numbers obtained should be printed in a comma-separated sequence
48. Write a Python program to print alphabet pattern 'E'
49. Write a Python program to print alphabet pattern 'L'
50. Write a Python program to convert month name to a number of days
51. Write a Python program to check a string represent an integer or not
52. Write a Python program to create the multiplication table (from 1 to 10) of a number
53. Write a Python program to construct the following pattern, using a nested loop number.  
1  
22  
333  
4444  
55555  
666666  
7777777  
88888888  
999999999
54. Python Program to Check Prime Number
55. Python Program to Print all Prime Numbers in an Interval
56. Python Program to Find the Factorial of a Numbers

## Python String

57. How to program to print first non-repeated character from String?
58. How to check if a String contains only digits?
59. How to find duplicate characters in a String?
60. How to count number of vowels and consonants in a String?
61. How to count occurrence of a given character in String?
62. How to find all permutations of String?
63. How to reverse words in a sentence without using library method?
64. How to check if String is Palindrome?
65. How to remove duplicate characters from String?
66. Write a program to check if a String contains another String
67. How to return highest occurred character in a String?
68. Write a program to remove a given characters from String?
69. Write a program to find longest palindrome in a string?
70. How to sort the letters in a string alphabetically in Python?

## Python List

71. Write a programme in python in which we can move element to end of the list.
72. Write a programme in python in which we can remove first element of list.
73. Write a programme in python in which we can shift last element to first position in list.
74. Write a programme in python in which we can.
75. Write a Python program to multiplies all the items in a list
76. Write a Python program to get the largest number from a list.
77. Write a Python program to get the smallest number from a list.
78. Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

Sample List : ['abc', 'xyz', 'aba', '1221']

Expected Result : 2

79. Write a Python program to remove duplicates from a list.
80. Write a Python program to check a list is empty or not
81. Write a Python program to find the list of words that are longer than n from a given list of words.
82. Write a Python function that takes two lists and returns True if they have at least one common member.
83. Write a Python program to print the numbers of a specified list after removing even numbers from it.
84. Write a Python program to find the second smallest number in a list.
85. Write a Python program to find the second largest number in a list.
86. Write a Python program to get unique values from a list.
87. Write a Python program to count the number of elements in a list within a specified range.
88. Write a Python program to create a list by concatenating a given list which range goes from 1 to n.

Sample list : ['p', 'q']

n = 5

Sample Output : ['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4', 'p5', 'q5']



89. Write a Python program to remove duplicates from a list of lists.  
Sample list : `[[10, 20], [40], [30, 56, 25], [10, 20], [33], [40]]`  
New List : `[[10, 20], [30, 56, 25], [33], [40]]`
90. Print list after removing element at given index.

### Python Dictionary

91. Write a Python script to sort (ascending and descending) a dictionary by value.  
92. Write a Python script to check if a given key already exists in a dictionary.  
93. Write a Python program to iterate over dictionaries using for loops.  
94. Write a Python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys  
95. Write a Python script to merge two Python dictionaries  
96. Write a Python program to sum all the items in a dictionary.  
97. Write a Python program to multiply all the items in a dictionary.  
100. Write a Python program to remove a key from a dictionary.  
101. Write a Python program to sort a dictionary by key.  
102. Write a Python program to get the maximum and minimum value in a dictionary.  
103. Write a Python program to check a dictionary is empty or not.  
104. Write a Python program to combine two dictionary adding values for common keys.  
`d1 = {'a': 100, 'b': 200, 'c': 300}`  
`d2 = {'a': 300, 'b': 200, 'd': 400}`  
Sample output: Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})
105. Write a Python program to print all unique values in a dictionary.  
Sample Data : `[{"V": "S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII": "S005"}, {"V": "S009"}, {"VIII": "S007"}]`  
Expected Output : Unique Values: {'S005', 'S002', 'S007', 'S001', 'S009'}
106. Write a Python program to find the highest 3 values in a dictionary.  
107. Write a Python program to create a dictionary from a string.  
Note: Track the count of the letters from the string.  
Sample string : 'w3resource'  
Expected output: {'3': 1, 's': 1, 'r': 2, 'u': 1, 'w': 1, 'c': 1, 'e': 2, 'o': 1}
108. Write a Python program to get the key, value and item in a dictionary.  
109. Write a Python program to print a dictionary line by line  
110. Write a Python program to check multiple keys exists in a dictionary.  
111. Write a Python program to count number of items in a dictionary value that is a list  
112. Write a Python program to replace dictionary values with their sum  
113. Define a function which can print a dictionary where the keys are numbers between 1 and 3 (both included) and the values are square of keys.  
114. Define a function which can generate a dictionary where the keys are numbers between 1 and 20 (both included) and the values are square of keys. The function should just print the values only.



## Python MCQ:-

**1. Is Python case sensitive when dealing with identifiers?**

- a) Yes
- b) no
- c) Machine dependent
- d) none of the mentioned

**2. What is the maximum possible length of an identifier?**

- a) 31 characters
- b) 63 characters
- c) 79 characters
- d) none of the mentioned

**3. Which of the following is invalid?**

- a) `_a = 1`
- b) `__a = 1`
- c) `__str__ = 1`
- d) none of the mentioned

**4. Which of the following is an invalid variable?**

- a) `my_string_1`
- b) `1st_string`
- c) `foo`
- d) `_`

**5. Why are local variable names beginning with an underscore discouraged?**

- a) they are used to indicate a private variables of a class
- b) they confuse the interpreter
- c) they are used to indicate global variables
- d) they slow down execution

**6. Which of the following is not a keyword?**

- a) `eval`
- b) `assert`
- c) `nonlocal`
- d) `pass`

**7. All keywords in Python are in**

- a) lower case
- b) UPPER CASE
- c) Capitalized
- d) None of the mentioned

**8. Which of the following is true for variable names in Python?**

- a) unlimited length
- b) all private members must have leading and trailing underscores

- c) underscore and ampersand are the only two special characters allowed
- d) none of the mentioned

**9. Which of the following is an invalid statement?**

- a) `abc = 1,000,000`
- b) `a b c = 1000 2000 3000`
- c) `a,b,c = 1000, 2000, 3000`
- d) `a_b_c = 1,000,000`

**10. Which of the following cannot be a variable?**

- a) `__init__`
- b) `in`
- c) `it`
- d) `on`

**11. Which is the correct operator for power(xy)?**

- a) `X^y`
- b) `X**y`
- c) `X^^y`
- d) None of the mentioned

**12. Which one of these is floor division?**

- a) `/`
- b) `//`
- c) `%`
- d) None of the mentioned

**13. What is the order of precedence in python?**

- i) Parentheses
- ii) Exponential
- iii) Multiplication
- iv) Division
- v) Addition
- vi) Subtraction

- a) i,ii,iii,iv,v,vi
- b) ii,i,iii,iv,v,vi
- c) ii,i,iv,iii,v,vi
- d) i,ii,iii,iv,vi,v

**14. What is the answer to this expression, `22 % 3` is?**

- a) 7
- b) 1
- c) 0
- d) 5

**15. Mathematical operations can be performed on a string. State whether true or false.**

- a) True
- b) False

**16. Operators with the same precedence are evaluated in which manner?**

- a) Left to Right
- b) Right to Left

c) Can't say mentioned

d) None of the

c) round()

d) round(7463.123,2,1)

**17. What is the output of this expression, 3\*1\*\*3?**

a) 27   b) 9   c) 3   d) 1

**18. Which one of the following has the same precedence level?**

- a) Addition and Subtraction
- b) Multiplication, Division and Addition
- c) Multiplication, Division, Addition and Subtraction
- d) Addition and Multiplication

**19. The expression Int(x) implies that the variable x is converted to integer. State whether true or false.**

a) True   b) False

**20. Which one of the following has the highest precedence in the expression?**

- a) Exponential   b) Addition
- c) Multiplication   d) Parentheses

**21. Which of these is not a core data type?**

- a) Lists   b) Dictionary
- c) Tuples   d) Class

**22. Given a function that does not return any value, What value is thrown by default when executed in shell.**

- a) int   b) bool
- c) void   d) None

**23. Following set of commands are executed in shell, what will be the output?**

```
>>>str="hello"
>>>str[:2]
```

a) he   b) lo   c) olleh   d) hello

**24. Which of the following will run without errors ?**

a) round(45.8)   b) round(6352.898,2,5)

**25. What is the return type of function id?**

a) int   b) float   c) bool   d) dict

**26. In python we do not specify types, it is directly interpreted by the compiler, so consider the following operation to be performed.**

```
>>>x = 13 ? 2
```

objective is to make sure x has a integer value, select all that apply (python 3.xx)

- a) x = 13 // 2   b) x = int(13 / 2)
- c) x = 13 % 2   d) All of the mentioned

**27. What error occurs when you execute? apple = mango**

- a) SyntaxError   b) NameError
- c) ValueError   d) TypeError

**28. What data type is the object below?**

```
L = [1, 23, 'hello', 1].
```

- a) list   b) dictionary   c) array   d) tuple

**29. In order to store values in terms of key and value we use what core data type.**

a) list   b) tuple   c) class   d) dictionary

**30. Which of the following results in a SyntaxError ?**

- a) "Once upon a time...", she said.'
- b) "He said, 'Yes!'"
- c) '3\'
- d) "That's okay"

**31. Select all options that print hello-how-are-you**

- a) print('hello', 'how', 'are', 'you')
- b) print('hello', 'how', 'are', 'you' + '-' \* 4)
- c) print('hello-' + 'how-are-you')
- d) print('hello' + '-' + 'how' + '-' + 'are' + 'you')

**32. What is the return value of trunc() ?**

a) int   b) bool   c) float   d) None

**33. What is the output of print 0.1 + 0.2 == 0.3?**

- a) True                      b) False  
c) Machine dependent      d) Error

**34. Which of the following is not a complex number?**

- a)  $k = 2 + 3j$       b)  $k = \text{complex}(2, 3)$   
c)  $k = 2 + 3I$       d)  $k = 2 + 3J$

**35. What is the type of inf?**

- a) Boolean      b) Integer  
c) Float      d) Complex

**36. What does ~4 evaluate to?**

- a) -5    b) -4    c) -3    d) +3

**37. What does ~~~~~5 evaluate to?**

- a) +5    b) -11    c) +11    d) -5

**38. Which of the following is incorrect?**

- a)  $x = 0b101$                       b)  $x = 0x4f5$   
c)  $x = 19023$                       d)  $x = 03964$

**39. What is the result of cmp(3, 1)?**

- a) 1                      b) 0  
c) True                      d) False

**40. Which of the following is incorrect?**

- a)  $\text{float}('inf')$                       b)  $\text{float}('nan')$   
c)  $\text{float}('56'+78')$                       d)  $\text{float}('12+34')$

**41. What is the result of round(0.5) – round(-0.5)?**

- a) 1.0                      b) 2.0  
c) 0.0                      d) None of the mentioned

**42. What does  $3^4$  evaluate to?**

- a) 81                      b) 12  
c) 0.75                      d) 7

**43. The value of the expressions  $4/(3*(2-1))$  and  $4/3*(2-1)$  is the same. State whether true or false.**

- a) True                      b) False

**44. The value of the expression:**

`>>4 + 3 % 5`

- a) 4                      b) 7

- c) 2                      d) 0

**45. Evaluate the expression given below if A= 16 and B = 15.**

`>>A % B // A`

- a) 0.0                      b) 0  
c) 1.0                      d) 1

**46 What is the output of the following?**

`x = ['ab', 'cd']`

`for i in x:`

`i.upper()`

`print(x)`

- a) ['ab', 'cd'].                      b) ['AB', 'CD'].  
c) [None, None].                      d) none of the mentioned

**47. What is the output of the following?**

`i = 1`

`while True:`

`if i%3 == 0:`

`break`

`print(i)`

`i += 1`

- a) 1 2                      b) 1 2 3  
c) error                      d) none of the mentioned

**48. What is the output of the following?**

`i = 1`

`while True:`

`if i%007 == 0:`

`break`

`print(i)`

`i += 1`

- a) 1 2 3 4 5 6                      b) 1 2 3 4 5 6 7  
c) error                      d) none of the mentioned

**49. What is the output of the following?**

`True = False`

`while True:`

`print(True)`

`break`

- a) True                      b) False

c) None mentioned

d) none of the

**50. What is the output of the following?**

`i = 1`

`while True:`

`if i%2 == 0:`

`break`

`print(i)`

`i += 2`

a) 1

b) 1 2

c) 1 2 3 4 5 6 ...

d) 1 3 5 7 9 11 ...

**51. What is the output of the following?**

`x = "abcdef"`

`i = "a"`

`while i in x[1:]:`

`print(i, end = " ")`

a) a a a a a a

b) a

c) no

output

d) error

**52. What is the output of the following?**

`i = 0`

`while i < 5:`

`print(i)`

`i += 1`

`if i == 3:`

`break`

`else:`

`print(0)`

a) 0 1 2 0

b) 0 1 2

c) error

d) none of the

mentioned

**53. What is the output of the following?**

`x = "abcdef"`

`i = "i"`

`while i in x:`

`print(i, end=" ")`

a) no output

b) i i i i i ...

c) a b c d e f

d) abcdef

**54. What is the output of the following?**

`x = "abcdef"`

`i = "a"`

`while i in x:`

`x = x[:-1]`

`print(i, end = " ")`

a) i i i i i

b) a a a a a a

c) a a a a a

d) none of the

mentioned

**55. What is the output of the following?**

`x = 'abcd'`

`for i in x:`

`print(i)`

`x.upper()`

a) a B C D

b) a b c d

c) A B C D

d) error

**56. What is the output of the following?**

`x = 'abcd'`

`for i in range(len(x)):`

`print(x)`

`x = 'a'`

a) a

b) abcd abcd abcd abcd

c) a a a a

d) none of the mentioned

**57. What is the output of the following?**

`for i in range(2.0):`

`print(i)`

a) 0.0 1.0

b) 0 1

c) error

d) none of the

mentioned

**58. What is the output of the following?**

`for i in 'abcd'[::-1]:`

`print (i)`

a) a b c d

b) d c b a

c) error

d) none of the

mentioned

**59. What is the output of the following?**

`x = 2`

`for i in range(x):`

`x -= 2`

`print (x)`

a) 0 1 2 3 4 ...

b) 0 -2

c) 0

d) error

**60. What arithmetic operators cannot be used with strings?**

- a) +                      b) \*
- c) -                      d) All of the mentioned

**61. Given a string example="hello" what is the output of example.count('l')**

- a) 2                      b) 1  
c) None                  d) 0

**62. What is the output of the following?**

```
print("Hello {name1} and
{name2}".format(name1='foo', name2='bin'))
```

- a) Hello foo and bin  
b) Hello {name1} and {name2}  
c) Error  
d) Hello and

**63. What is the output of the following?**

```
print("Hello {0!r} and {0!s}".format('foo',
 'bin'))
```

- a) Hello foo and foo
- b) Hello 'foo' and foo
- c) Hello foo and 'bin'
- d) Error

**64. What is the output of the following?**

```
print("Hello {0[0]} and {0[1]}".format(('foo',
'bin')))
```

- a) Hello foo and bin  
b) Hello ('foo', 'bin') and ('foo', 'bin')  
c) Error  
d) None of the mentioned

**65. What is the output of the following?**

```
print('The sum of {0:b} and {1:x} is
{2:o}'.format(2, 10, 12))
```

- a) The sum of 2 and 10 is 12  
b) The sum of 10 and a is 14  
c) The sum of 10 and a is c  
d) Error

**66. What is the output of the following?**

```
print('{:$}'.format(1112223334))
```

- a) 1,112,223,334  
b) 111,222,333,4  
c) 111222333  
d) Error

**67. What is the output of the following?**

```
print('Hello!2@#World'.istitle())
```

- a) True                      b) False  
c) None                      d) error

**68. What is the output of the following?**

```
print("\tfoo".lstrip())
```

- a) \tfoo      b) foo  
c) foo      d) none of the mentioned

**69. What is the output of the following?**

```
print('abcdefcdghcd'.split('cd', 2))
```

- a) ['ab', 'ef', 'ghcd'].  
b) ['ab', 'efcdghcd'].  
c) ['abcdef', 'ghcd'].  
d) none of the mentioned

**70. Which of the following commands will create a list?**

- a) list1 = list()  
b) list1 = [].  
c) list1 = list([1, 2, 3])  
d) all of the mentioned

**71. What is the output when we execute `list("hello")`?**

- a) ['h', 'e', 'l', 'l', 'o'].                      b) ['hello'].  
c) ['llo'].                                              d) ['olleh'].

**72. What is the output when we execute `list("hello")`?**

- a) ['h', 'e', 'l', 'l', 'o'].      b) ['hello'].  
c) ['llo'].      d) ['olleh'].

73. If  $a=(1,2,3,4)$ ,  $a[1:-1]$  is

- a) Error, tuple slicing doesn't exist
- b) [2,3]
- c) (2,3,4)
- d) (2,3)

**74. What is the output of the following piece of code?**

```
a = ('check',)
```

**n = 2**

```
for i in range(int(n)):
```

$$\mathbf{a} = (a_i)$$

```
print(a)
```

- a) Error, tuples are immutable
- b) (('check',),)  
((( 'check',),),).
- c) (('check',)'check',)
- d) (('check',)'check',)  
((( 'check',)'check',)'check',)

**75. To open a file c:\scores.txt for reading, we use**

- a) infile = open("c:\scores.txt", "r")
- b) infile = open("c:\\scores.txt", "r")
- c) infile = open(file = "c:\scores.txt", "r")
- d) infile = open(file = "c:\\scores.txt", "r")

**76. To read the entire remaining contents of the file as a string from a file object infile, we use**

- a) infile.read(2)
- b) infile.read()
- c) infile.readline()
- d) infile.readlines()

**77. In file handling, what does this terms means "r, a"?**

- a) read, append
- b) append, read
- c) all of the mentioned
- d) none of the the mentioned

**78. \_\_\_\_\_ represents an entity in the real world with its identity and behaviour.**

- a) A method
- b) An object
- c) A class
- d) An operator

**79. What are the methods which begin and end with two underscore characters called?**

- a) Special methods
- b) In-built methods
- c) User-defined methods
- d) Additional methods

**80. What is the output of the following piece of code?**

```
class Test:
 def __init__(self):
 self.x = 0
class Derived_Test(Test):
```

```
 def __init__(self):
 self.y = 1
def main():
 b = Derived_Test()
 print(b.x,b.y)
main()
```

- a) 0 1
- b) 0 0
- c) Error because class B inherits A but variable x isn't inherited
- d) Error because when object is created, argument must be passed like Derived\_Test(1)

**81. What is the output of the following piece of code?**

```
class Test:
 def __init__(self):
 self.x = 0
class Derived_Test(Test):
 def __init__(self):
 Test.__init__(self)
 self.y = 1
def main():
 b = Derived_Test()
 print(b.x,b.y)
main()
```

- a) Error because class B inherits A but variable x isn't inherited
- b) 0 0
- c) 0 1
- d) Error, the syntax of the invoking method is wrong

**82. What is the output of the following piece of code?**

```
class A:
 def __init__(self):
 self.__i = 1
 self.j = 5

 def display(self):
 print(self.__i, self.j)
class B(A):
 def __init__(self):
 super().__init__()
 self.__i = 2
 self.j = 7
c = B()
```

**c.display()**

- a) 2 7
- b) 1 5
- c) 1 7
- d) 2 5

**83. What is the output of the following piece of code?**

```
class A:
 def __init__(self):
 self._x = 5
class B(A):
 def display(self):
 print(self._x)
def main():
 obj = B()
 obj.display()
main()
```

- a) Error, invalid syntax for object declaration
- b) Nothing is printed
- c) 5
- d) Error, private class member can't be accessed in a subclass

**84. Which of the following best describes polymorphism?**

- a) Ability of a class to derive members of another class as a part of its own definition
- b) Means of bundling instance variables and methods in order to restrict access to certain class members
- c) Focuses on variables and passing of variables to functions
- d) Allows for objects of different types and behaviour to be treated as the same general type

**85. What is the output of the following piece of code?**

```
class A:
 def __repr__(self):
 return "1"
class B(A):
 def __repr__(self):
 return "2"
class C(B):
 def __repr__(self):
 return "3"
```

```
o1 = A()
o2 = B()
o3 = C()
print(obj1, obj2, obj3)
```

- a) 1 1 1
- b) 1 2 3
- c) '1' '1' '1'
- d) An exception is thrown

**86. Which of the following statements is true?**

- a) A non-private method in a superclass can be overridden
- b) A subclass method can be overridden by the superclass
- c) A private method in a superclass can be overridden
- d) Overriding isn't possible in Python

**87. Which of the following is the most suitable definition for encapsulation?**

- a) Ability of a class to derive members of another class as a part of its own definition
- b) Means of bundling instance variables and methods in order to restrict access to certain class members
- c) Focuses on variables and passing of variables to functions
- d) Allows for implementation of elegant software that is well designed and easily modified

**88. What is the output for the following piece of code?**

```
class Demo:
 def __init__(self):
 self.a = 1
 self.__b = 1
 def get(self):
 return self.__b
obj = Demo()
obj.a=45
print(obj.a)
```

- a) The program runs properly and prints 45
- b) The program has an error because the value of members of a class can't be changed from outside the class
- c) The program runs properly and prints 1



d) The program has an error because the value of members outside a class can only be changed as self.a=45

**89. How many except statements can a try-except block have?**

- a) zero                                      b) one
- c) more than one                      d) more than zero

**90. What is the output of the code shown?**

```
def f(x):
 for i in range(5):
 yield i
g=f(8)
print(list(g))
```

- a) [0, 1, 2, 3, 4]
- b) [1, 2, 3, 4, 5, 6, 7, 8]
- c) [1, 2, 3, 4, 5]
- d) [0, 1, 2, 3, 4, 5, 6, 7]

**91. What is the output of the code shown below?**

```
g = (i for i in range(5))
type(g)
a) class <'loop'>
b) class <'iteration'>
c) class <'range'>
d) class <'generator'>
```

**92. What is the output of the code shown below?**

```
lst = [1, 2, 3]
lst[3]
a) NameError b) ValueError
c) IndexError d) TypeError
```

**93. Which of the following is not a standard exception in Python?**

- a) NameError                      b) IOError
- c) AssignmentError              d) ValueError

**94. An exception is:**

- a) an object                      b) a special function
- c) a standard module              d) a module

**95. Which of the following is not a valid attribute of a file object (fp)?**

- a) fp.name                              b) fp.closed
- c) fp.mode                              d) fp.size

**96. What is the output of the code shown?**

```
l=[1,2,3,4,5]
[x&1 for x in l]
```

- a) [1, 1, 1, 1, 1]                      b) [1, 0, 1, 0, 1]
- c) [1, 0, 0, 0, 0]                      d) [0, 1, 0, 1, 0]

**97. What is the output of the following code?**

```
l=[[1, 2, 3], [4, 5, 6]]
for i in range(len(l)):
 for j in range(len(l[i])):
 l[i][j]+=10
```

- a) No output
- b) Error
- c) [[1, 2, 3], [4, 5, 6]]
- d) [[11, 12, 13], [14, 15, 16]]

**98. Which of the following is true about top-down design process?**

- a) The details of a program design are addressed before the overall design
- b) Only the details of the program are addressed
- c) The overall design of the program is addressed before the details
- d) Only the design of the program is addressed

**99. Is the output of the function abs() the same as that of the function math.fabs()?**

- a) sometimes                      b) always
- c) never                              d) none of the mentioned

**100. What is the output of print(math.factorial(4.5))?**

- a) 24                                      b) 120
- c) error                                  d) 24.0



- |       |       |        |
|-------|-------|--------|
| 1. a  | 35. c | 69. a  |
| 2. c  | 36. c | 70. b  |
| 3. d  | 37. a | 71. a  |
| 4. b  | 38. a | 72. d  |
| 5. a  | 39. d | 73. c  |
| 6. a  | 40. a | 74. c  |
| 7. d  | 41. c | 75. b  |
| 8. a  | 42. d | 76. b  |
| 9. b  | 43. b | 77. a  |
| 10. b | 44. d | 78. b  |
| 11. b | 45. a | 79. a  |
| 12. b | 46. b | 80. c  |
| 13. a | 47. b | 81. c  |
| 14. b | 48. a | 82. b  |
| 15. b | 49. c | 83. c  |
| 16. a | 50. a | 84. d  |
| 17. c | 51. d | 85. b  |
| 18. a | 52. d | 86. a  |
| 19. a | 53. c | 87. b  |
| 20. d | 54. a | 88. a  |
| 21. d | 55. b | 89. d  |
| 22. d | 56. b | 90. a  |
| 23. a | 57. b | 91. d  |
| 24. a | 58. d | 92. c  |
| 25. a | 59. b | 93. c  |
| 26. d | 60. b | 94. a  |
| 27. b | 61. b | 95. d  |
| 28. a | 62. c | 96. b  |
| 29. d | 63. a | 97. d  |
| 30. c | 64. a | 98. c  |
| 31. c | 65. b | 99. a  |
| 32. a | 66. a | 100. c |
| 33. a | 67. b |        |
| 34. b | 68. d |        |