PYTHON- Fundamentals

- Python was developed by Guido Van Rossum in 1991 when he was working with National Research Institute of Mathematics and Science in Netherland.
- **Python** is a fantastic **programming language** that allows you to use both functional and **object-oriented** programming paradigms.

- **HOW TO INSTALL?**
- **Python can be downloaded from www.python.org.**
- It is available in two versions-
- Python 2.x
- Python 3.x

- We can work in Python in two ways-
  - Interactive Mode
    - Interactive mode works like a Command Interpreter as Shell Prompt works in DOS Prompt or Linux..
    - (>>>) we can execute any instruction of Python with this.
  - Script Mode
    - We can run a complete program by writing in Script mode.

Interactive mode

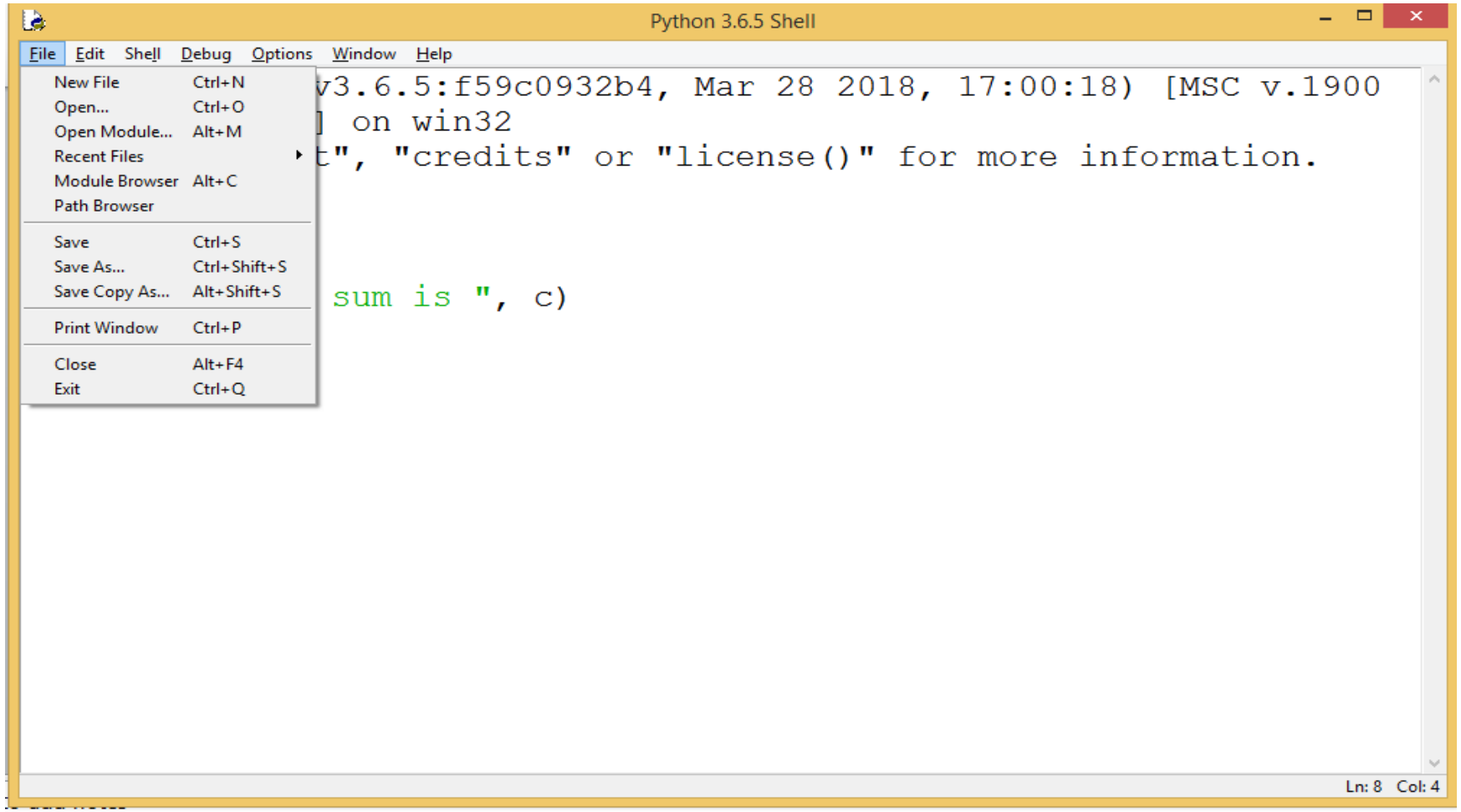
Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900
64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=10
>>> b=20
>>> c=a+b
>>> print("the sum is ", c)
the sum is  30
>>> |
```

Ln: 8 Col: 4

Script mode


Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
New File Ctrl+N
Open... Ctrl+O
Open Module... Alt+M
Recent Files
Module Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q
v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900
] on win32
t", "credits" or "license()" for more information.
sum is ", c)
Ln: 8 Col: 4

Python Character Set

- Character Set-is a group of letters or signs which are specific to a language.
- Character set includes letter, sign, number, symbol.
  - Letters: A-Z, a-z
  - Digits: 0-9
  - Special Symbols: _, +, -, *, /, (,), {, } . . . Etc.
  - White Spaces: blank space, tab, carriage return, newline, formfeed etc.
  - Other characters: Python can process all characters of ASCII and UNICODE.

Token

- Token- is the smallest unit of any programming
- language. It is also known as Lexical Unit. Types of token are-
  - Keywords
  - Identifiers (Names)
  - Literals
  - Operators
  - Punctuators

Keywords

- Keywords are those words which provides a special meaning to interpreter.
- These are reserved for specific functioning.

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	

Identifiers

- These are building blocks of a program and are used to give names to different parts/blocks of a program like - variable, objects, classes, functions.
- An identifier must begin with an alphabet or an underscore(_).
- Subsequent letters may be numbers(0-9).
- Python is case sensitive. Uppercase characters are distinct from lowercase characters (P and p are different for interpreter).
- Some valid identifiers are –
  - Myfile, Date9_7_17, Z2T0Z9, _DS, _CHK FILE13.
- Some invald identifiers are –
  - DATA-REC, 29COLOR, break, My.File

Literals / Values

- Literals are often called Constant Values.
- Python permits following types of literals -
  - String literals - “Pankaj”
  - Numeric literals – 10, 13.5, 3+5i
  - Boolean literals – True or False
  - Special Literal ***None***
  - Literal collections

String Literals

- String Literal is a sequence of characters that can be a combination of letters, numbers and special symbols, enclosed in quotation marks, single, double or triple(“ “ or ‘ ‘ or “’ ‘”).
- In python, string is of 2 types-
  - Single line string
    - Text = “Hello World” or Text = ‘Hello World’
  - Multi line string
    - Text = ‘hello\ or Text = “’hello
      world’ word “’

Numeric Literals

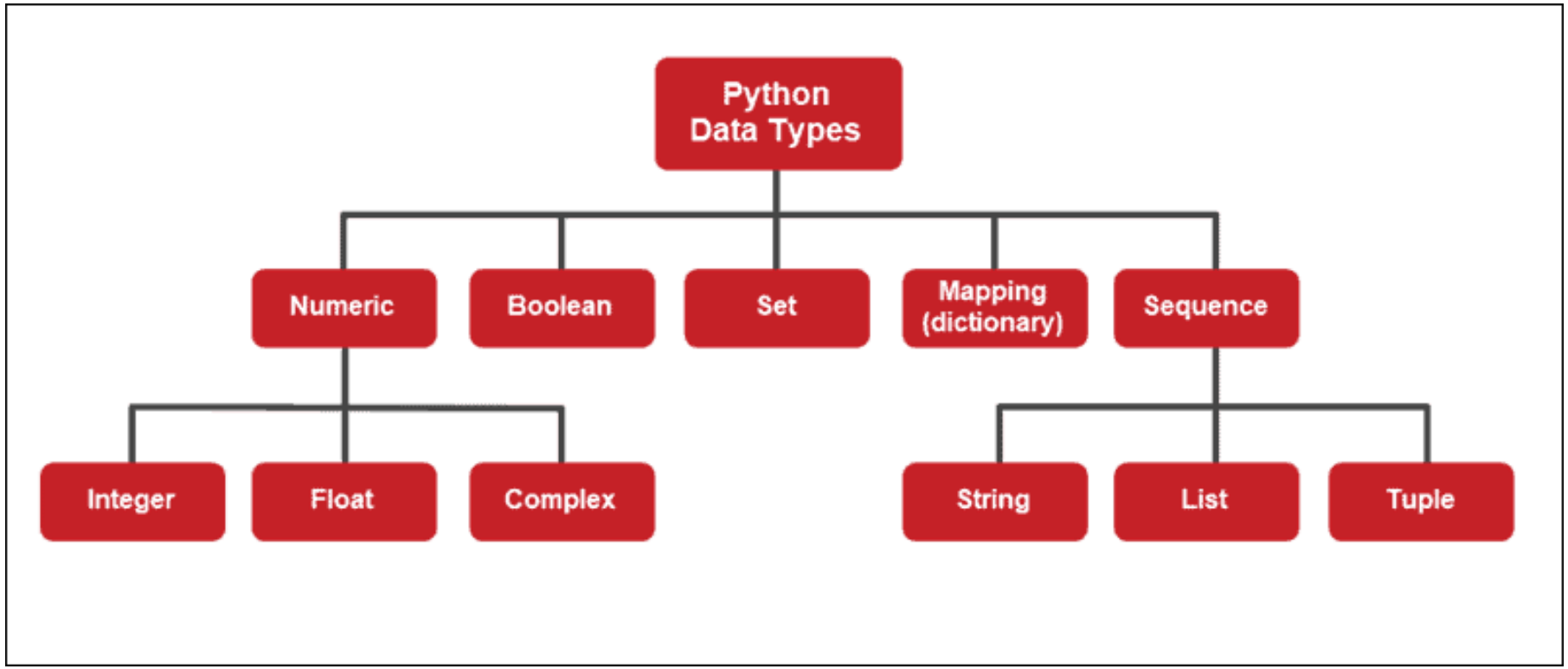
- Numeric values can be of three types -
- int (signed integers)
    - Decimal Integer Literals – 10, 17, 210 etc.
    - Octal Integer Literals - 0o17, 0o217 etc.
    - Hexadecimal Integer Literals – 0x14, 0x2A4, 0xABD etc.
- float (floating point real value)
    - Fractional Form – 2.0, 17.5 -13.5, -.00015 etc.
    - Exponent Form - -1.7E+8, .25E-4 etc.
- complex (complex numbers)
    - 3+5i etc.

Boolean Literals

- It can contain either of only two values – True or False
- A= True
- B=False

- **Special Literals**
- None, which means nothing (no value).
- X = None

CORE DATA TYPES



DATA TYPES

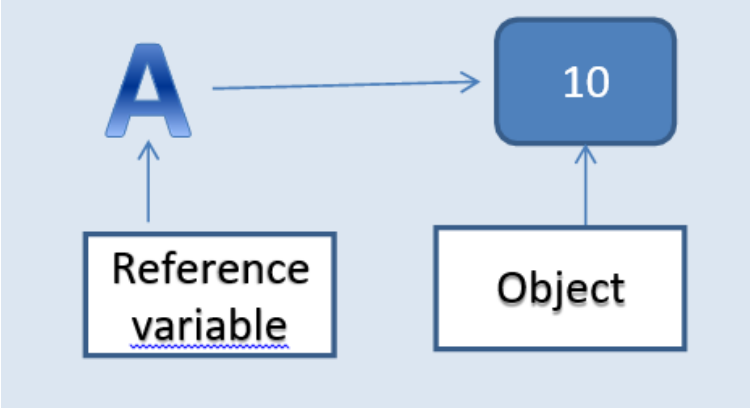
- Python supports following core data types-
  - Numbers(int like 10, 5) (float like 3.5, 302.24) (complex like 3+5i)
  - String (like “pankaj”, ‘pankaj’, ‘a’, “a”)
  - List like [3,4,5,”pankaj”] its elements are Mutable.
  - Tuple like(3,4,5,”pankaj”) its elements are immutable.
  - Dictionary like {‘a’:1, ‘e’:2, ‘I’:3, ‘o’:4, ‘u’:5} where a,e,i,o,u are keys and 1,2,3,4,5 are their values.

Input and Output

- In Python, we have the input() function for taking the user input.
- The syntax for input() is:
- input ([Prompt])
- Python uses the print() function to output data to standard output device — the screen.

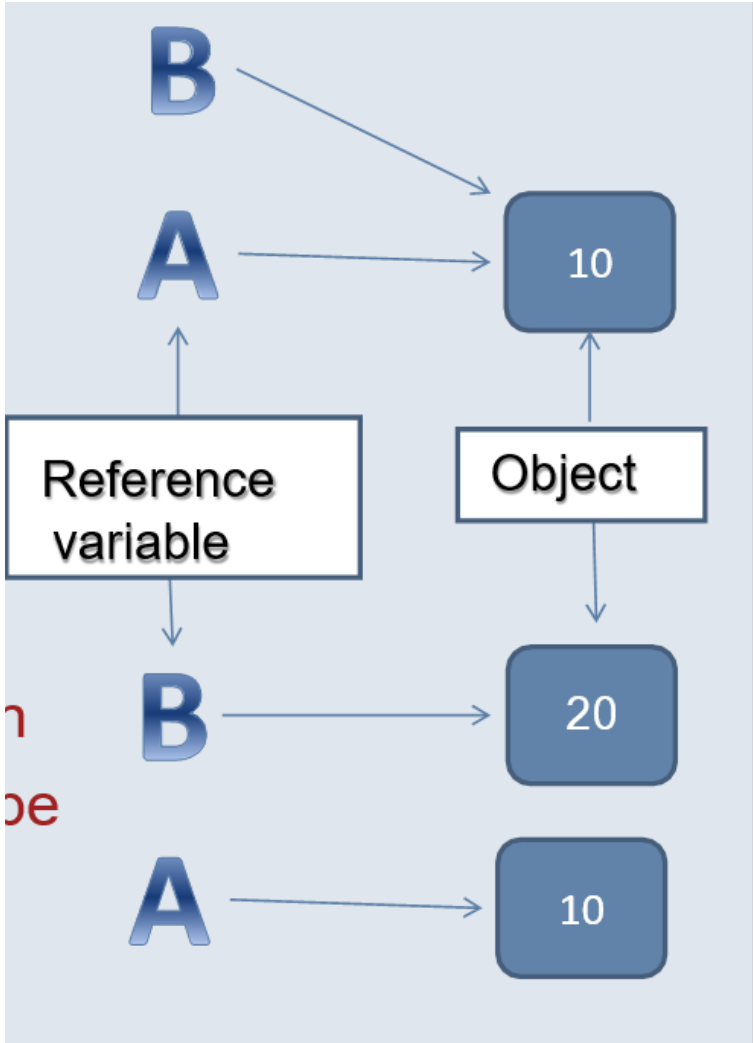
Variables and Values

- An important fact to know is-
- – In Python, values are actually objects.
- – And their variable names are actually their reference names.
- Suppose we assign 10 to a variable A. A = 10
- Here, value 10 is an object and A is its reference name.



Variables and Values

- If we assign 10 to a variable B, B will refer to same object.
- Here, we have two variables, but with same location.
- Now, if we change value of B like B=20.
- Then a new object will be created with a new location 20 and this object will be referenced by B.



- Pay attention to the following command

```
>>> a=4
>>> type(4)
<class 'int'>
>>> type(a)
<class 'int'>
```

here 4 is an object and its class is int

here a is referring to the object which is of int class.

- The Value of an Object

```
>>> print(4)
4
>>> print(a)
4
```

- The ID of an Object

```
>>> id(4)
1817668720
>>> id(a)
1817668720
>>> |
```

Operators

- Python supports following operators-
  - Arithmetic Operator
  - Relation Operator
  - Identity Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators

Arithmetic Operators

- Python has following binary arithmetic operator -
  - For addition + for ex- 2+3 will result in to 5
  - For subtraction – for ex- 2-3 will result in to -1
  - For multiplication * for ex- 2*3 will result in to 6
  - For division / its result comes in fraction.
    for ex- 13/2 will result in to 6.5
  - For quotient // its result comes as a whole number
    for ex- 13/2 will result into 6.
  - For remnant % its result comes as a whole remnant number.For ex-13/2will result into 1.
  - For exponent ** it will come as per exponent value. For ex- 2^3 will result into 8.

Assignment Operators and shorthand

- Python has following assignment operator and shorthand -
  - = a=10 , 10 will be assigned to a.
  - += a+=5 is equal to a=a+5.
  - -= a-=5 is equal to a=a-5.
  - *= a*=5 is equal to a=a*5.
  - /= a/=5 is equal to a=a/5.
  - //= a//=5 is equal to a=a//5.
  - %= a%=5 is equal to a=a%5.
  - **= a**=5 is equal to a=a**5.

Relational Operators

• <	Less Than	like a<b
• >	Greater Than	like a>b
• <=	Less Than and Equal to	like a<=b
• >=	Greater Than and Equal to	like a>=b
• ==	Equal to	like a==b
• !=	not Equal to	like a!=b

Identity Operators

- Identity operator is also used to check for equality. These expression also results into True or False. Identity Operators are of following types-

- “is” operator — if a=5 and b=5 then a is b will come to True
- “is not” operator — if a=5 and b=5 then a is not b will come to False

- Relational Operator (==) and Identity operator (is) differs in case of strings that we will see later.

Logical Operators

- Python has two binary logical operators -
- or operator
    - » if a = True and b = False then **a or b** will return ***True*.**
- and operator
    - » If a = True and b = False then **a and b** will return ***False*.**

- Python has one Unary logical operator –
  - not operator
    - if a = True then **not a** will return ***False*.**

Operator Associativity

- In Python, if an expression or statement consists of multiple or more than one operator then operator associativity will be followed from left-to- right.

```
>>> 7*8/5//2
5.0
```

- Only in case of **, associativity will be followed from right-to-left.

```
>>> 3**3**2
19683
```

- Above given example will be calculated as 3**(3**2).

Type Casting

- Consider the following program

```
num1 = input("Enter a number and I'll double it: ")
num1 = num1 * 2
print(num1)
```

- num1 = input("Enter a number and I'll double it: ")

num1 = num1 * 2

print(num1)

- Enter a number and I'll double it: 2

22

Note:

In Python, we use the input() function to take input from the user.

Whatever you enter as input, the input function converts it into a string.

- To get 4 as output,
- We need to convert the data type of the value entered by the user to integer. Thus, we modify the program as follows:
- num1 = input("Enter a number and I'll double it: ")

```
num1 = int(num1) #convert string input to integer
num1 = num1 * 2
print(num1)
```

- We can change the data type of a variable in Python from one type to another. Such data type conversion can happen in two ways:
  - 1. Implicit
  - 2. Explicit
- In python, following are the data conversion functions-
- (1) int ( ) (2) float( ) (3) complex( ) (4) str( ) (5) bool( )

Program to show implicit conversion

```
num1 = 10 #num1 is an integer
num2 = 20.0 #num2 is a float
sum1 = num1 + num2 #sum1 is sum of a float
and an integer
print(sum1)
print(type(sum1))
```

- Output:

```
30.0
<class 'float'>
```

Working with math Module of Python

- Python provides math module to work for all mathematical works.

```
import math
a=25
print(math.sqrt(a))
```

```
>>> import math
>>> dir (math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acos
h', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cos
h', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor
', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfini
te', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'mo
df', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau
', 'trunc']
>>>
```

Examples

- Addition
- >>> num1 = 5

>>> num2 = 6

>>> num1 + num2

Output:

- >>> str1 = "Hello"

>>> str2 = "India"

>>> str1 + str2

Output:

Multiplication

- >>> num1 = 5
- >>> num2 = 6
- >>> num1 * num2

- >>> str1 = 'India'
- >>> str1 * 2

Division

- /
- >>> num1 = 8
- >>> num2 = 4
- >>> num2 / num1

- //
- >>> num1 = 13
- >>> num2 = 4
- >>> num1 // num2

- >>> num1 = 5

>>> type(num1) is int

Output:

- num2 = num1

num1 is num2

Output:

- >>> num1 is not num2

Output:

Control Statements

Flow control statements are used to control the flow of execution depending upon the specified condition/logic.

Sequential control statement - Sequential execution is when statements are executed one after another in order.

There are three types of control statements.

1. Decision Making Statements/If control statement
2. Iteration Statements (Loop control statement)
3. Jump Statements (break, continue, pass)

Decision Making Statements

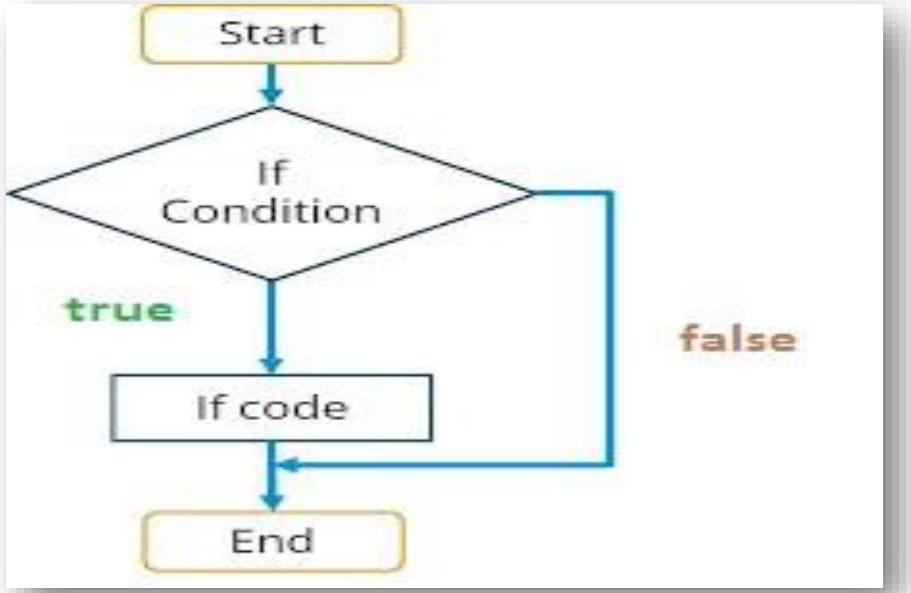
Decision making statement used to control the flow of execution of program depending upon condition.

There are three types of decision making statement.

1. if statements
2. if-else statements
3. Nested if-else statement

1. if statements

An if statement is a programming conditional statement that, if proved true, performs a function or displays information.



1. if statements

Syntax:

```
if(condition):
        statement
        [statements]
```

e.g.

```
noofbooks = 2
if (noofbooks == 2):
        print('You have ')
        print('two books')
print('outside of if statement')
```

Output

You have two books

Note:To indicate a block of code in Python, you must indent each line of the block by the same amount. In above e.g. both print statements are part of if condition because of both are at same level indented but not the third print statement.

1. if statements

Using logical operator in if statement

```
x=1
y=2
if(x==1 and y==2):
  print(‘condition matcing the criteria')
```

Output :-

condition matcing the criteria

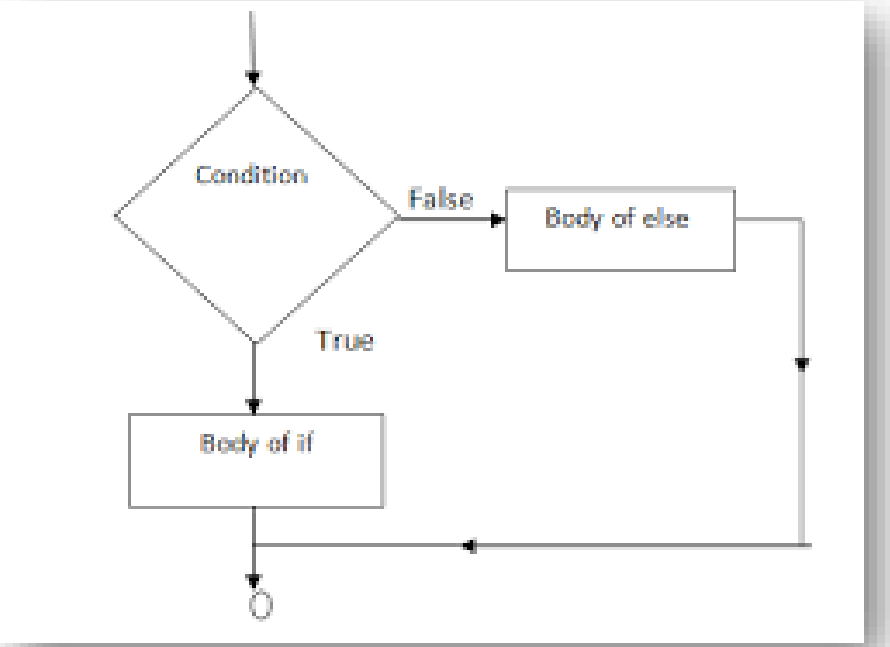
```
a=100
if not(a == 20):
  print('a is not equal to 20')
```

Output :-

a is not equal to 20

2. if-else Statements

If-else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false.



2. if-else Statements

Syntax:

```
        if(condition):
                statements
        else:
                statements
```

e.g.

```
a=10
if(a < 100):
  print(‘less than 100')
else:
  print(‘more than equal 100')
```

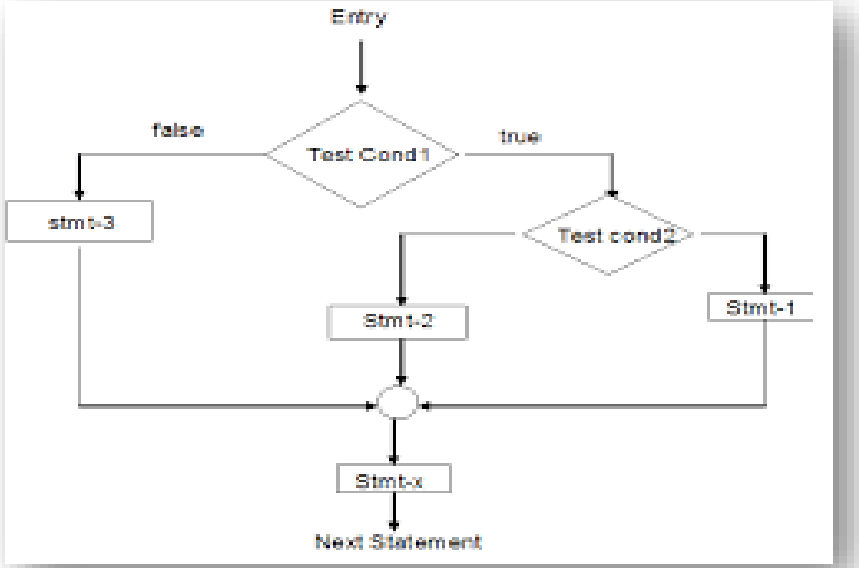
OUTPUT

less than 100

*Write a program in python to check that entered numer is even or odd

3. Nested if-else statement

The nested if...else statement allows you to check for multiple test expressions and execute different codes for more than two conditions.



3. Nested if-else statement

Syntax

```
If (condition):
        statements
elif (condition):
        statements
else:
        statements
```

E.G.

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

OUTPUT

```
Enter a number: 5
Positive number
```

* Write python program to find out largest of 3 numbers.

3.Nested if-else Statements

#sort 3 numbers

```
first = int(input("Enter the first number: "))
second = int(input("Enter the second number: "))
third = int(input("Enter the third number: "))
small = 0
middle = 0
large = 0
if first < third and first < second:
    small = first
    if second < third and second < first:
        small = second
    else:
        small = third
elif first < second and first < third:
    middle = first
    if second > first and second < third:
        middle = second
    else:
        middle = third
elif first > second and first > third:
    large = first
    if second > first and second > third:
        large = second
    else:
        large = third
print("The numbers in accending order are: ", small, middle, large)
```

Iteration Statements (Loops)

Iteration statements(loop) are used to execute a block of statements as long as the condition is true.

Loops statements are used when we need to run same code again and again.

Python Iteration (Loops) statements are of three type :-

1. While Loop

2. For Loop

3. Nested For Loops

1. While Loop

It is used to execute a block of statement as long as a given condition is true. And when the condition become false, the control will come out of the loop. The condition is checked every time at the beginning of the loop.

Syntax

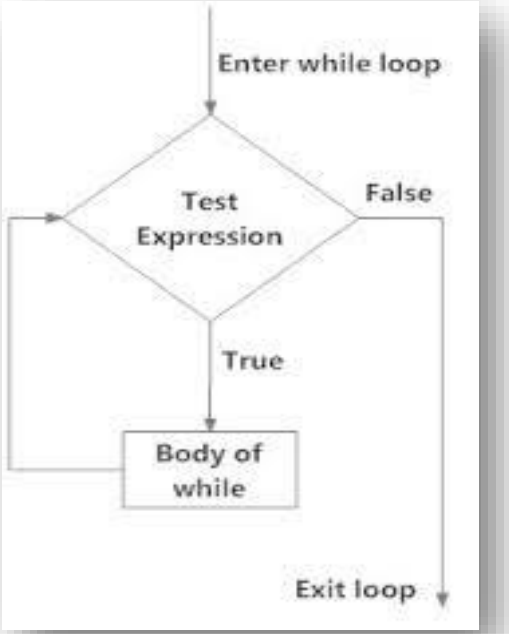
```
while (condition):
        statement
        [statements]
```

e.g.

```
x = 1
while (x <= 4):
  print(x)
  x = x + 1
```

Output

1
2
3
4



While Loop continues

While Loop With Else

e.g.

```
x = 1
while (x < 3):
  print('inside while loop value of x is ',x)
  x = x + 1
else:
  print('inside else value of x is ', x)
```

Output

```
inside while loop value of x is 1
inside while loop value of x is 2
inside else value of x is 3
```

*Write a program in python to find out the factorial of a given number

While Loop continue

Infinite While Loop

e.g.

```
x = 5
while (x == 5):
   print(‘inside loop')
```

Output

Inside loop

Inside loop

…

…

2. For Loop

It is used to iterate over items of any sequence, such as a list or a string.

Syntax

```
for val in sequence:
        statements
```

e.g.

```
for i in range(3,5):
  print(i)
```

Output

3

4

2. For Loop continue

Example programs

```
for i in range(5,3,-1):
  print(i)
```

Output

5

4

range() Function Parameters

Returns a sequence of numbers

start: Starting number of the sequence.

stop: Generate numbers up to, but not including this number.

step(Optional): Determines the increment between each numbers in the sequence.

2. For Loop continue

Example programs with range() and len() function

```
fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
  print ('Current fruit :', fruits[index])
```

range() with len() Function Parameters

2. For Loop continue

For Loop With Else

e.g.

```
for i in range(1, 4):
    print(i)
else:  # Executed because no break in for
    print("No Break")
```

Output

```
1
2
3
No Break
```

2. For Loop continue

Nested For Loop

e.g.

```
for i in range(1,3):
  for j in range(1,11):
    k=i*j
    print (k, end=' ')
  print()
```

Output

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
```

2. For Loop continues

Factorial of a number

```
factorial = int(input(‘enter a number’))

# check if the number is negative, positive or zero
if num < 0:
  print("Sorry, factorial does not exist for negative
numbers")
elif num == 0:
  print("The factorial of 0 is 1")
else:
  for i in range(1,num + 1):
     factorial = factorial*i
  print("The factorial of",num,"is",factorial)
```

3. Jump Statements

Jump statements are used to transfer the program's control from one location to another. Means these are used to alter the flow of a loop like - to skip a part of a loop or terminate a loop

There are three types of jump statements used in python.

1. break
2. continue
3. pass

1.break

it is used to terminate the loop.

e.g.

```
for val in "string":
    if val == "i":
        break
    print(val)

print("The end")
```

Output

```
s
t
r
The end
```

2.continue

It is used to skip all the remaining statements in the loop and move controls back to the top of the loop.

e.g.

```
for val in "init":
    if val == "i":
        continue
    print(val)
print("The end")
```

Output

```
n
t
The end
```

3. pass Statement

This statement does nothing. It can be used when a statement is required syntactically but the program requires no action.

Use in loop

```
while True:
        pass # Busy-wait for keyboard interrupt (Ctrl+C)
```

In function

It makes a controller to pass by without executing any code.

e.g.

```
def myfun():
    pass   #if we don’t use pass here then error message will be shown
print(‘my program')
```

OUTPUT

My program

3. pass Statement continue

e.g.

```
for i in 'initial':
    if(i == 'i'):
        pass
    else:
        print(i)
```

OUTPUT

```
n
t
a
L
```

NOTE : continue forces the loop to start at the next iteration while pass means "there is no code to execute here" and will continue through the remainder or the loop body.

Thank You