AMIT ACADEMY
for Computer Education

SUBJECT – PROGRAMMING WITH PYTHON

Faculty Name – Ankit Rami

Unit-2

Programming skills with Python

AMIT ACADEMY FOR COMPUTER EDUCATION

Nr Vardayini Mataji Temple, Rupal, Gandhinagar-382630

Email – amitacademy1117@gmail.com

Mobile No – 8460467193

YouTube Link –

https://www.youtube.com/@ankitramijoinar

Instagram Link –

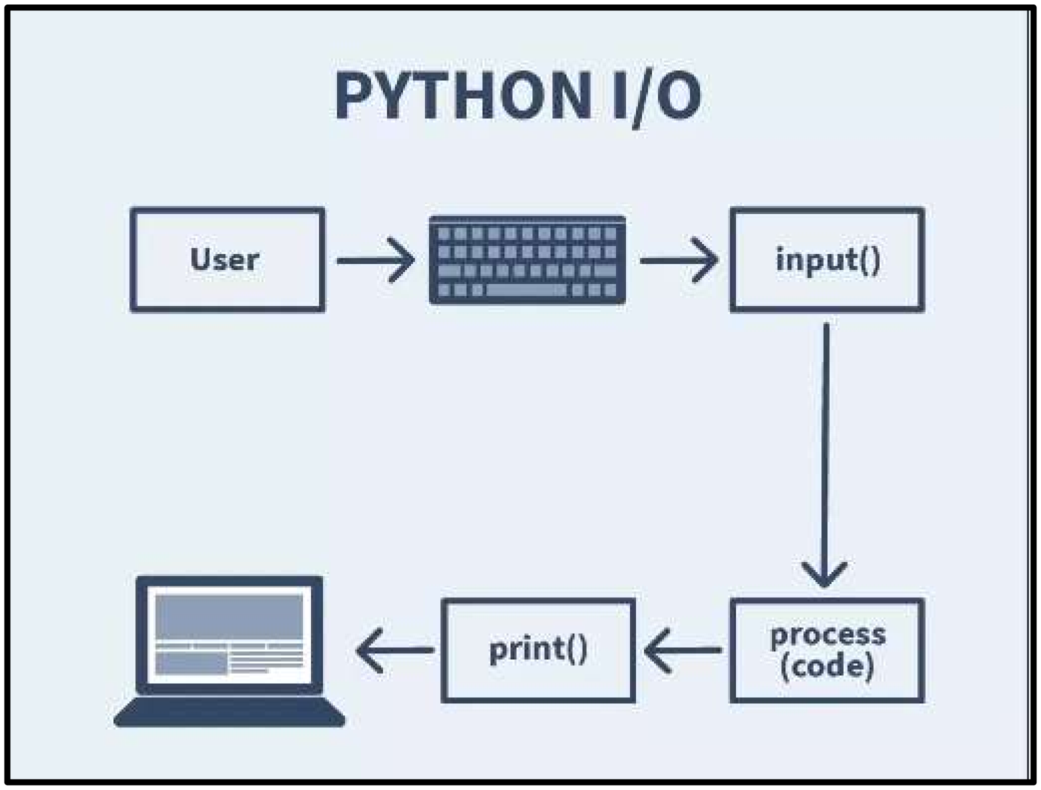
https://www.instagram.com/amitacademy17/

Facebook Link –

https://www.facebook.com/aramitacademy/

Input – Output statement of Python?

- ✓ There are several ways to deliver the output of a program. In Python, we use the print() function to output data to the screen.
- ✓ Sometimes we might want to take input from the user. We can do so by using the input() function.
- ✓ Python takes all the input as a string input by default. To convert it to any other data type, we have to convert the input explicitly.
- ✓ In Python, we have many inbuilt functions that help obtain data from the user and then display the data after processing some particular logic in the code.
- ✓ You can refer to the below flowchart to clarify how the system of inputting data and displaying it on some devices works.



Python Output

- ✓ We use the widely used print() statement to display some data on the screen.
- ✓ We can output the particular data on some device(screen) or even in some file.
- ✓ While writing real-world programs, we have to write statements that explicitly output numbers and strings.
- ✓ **Ex- print("Welcome to SCJPCCS")**
  Output - Welcome to SCJPCCS

- ✓ **For your better understanding of the syntax here we have defined a few keywords**
  - **object(s)** are the values to be printed on the screen. They are converted to strings before getting printed.
  - **sep** keyword is used to specify how to separate the objects inside the same print statement. By default, we have it as sep=' ', a space between two objects.
  - **end** is used to print a particular thing after all the values are printed. By default, we have end as \n, which provides a new line after each print() statement.
  - **file** is used to specify where to display the output. By default, it is sys.stdout (which is the screen).
  - **flush** specifies the boolean expression if the output is False or True. By default, it is False. In Python, the output from the print() goes into a buffer. Using the flush= True parameter helps in flushing the buffer as soon as we use the print() statement.

Ex –

```
print(0, 1, 2)
print(0, 1, 2, sep='$')
print(1, 2, sep='@', end='%')
```

Output –

```
0 1 2  # Here we have space between values by default.
0$1$2
1@2%
```

✓ **Ex – Python Output Formatting**

```
x = 3
y = 12
mul = x * y
print('The value of x is {} and y is {}'.format(x, y))
```

Output –

```
The value of x is 3 and y is 12
```

Python Input

- Sometimes users or developers want to run the programs with their own data in the variables for their own ease.
- To execute this we will learn to take input from the users, in which the users themselves will define the values of the variables**.**
- The input() statement allows us to do such things in Python.
- **Ex-**

```
name = input("Enter your first name: ")
print("Hey! " + name)
```

Output -

```
Enter your first name: Ankit Rami
Hey! Ankit Rami
```

➢ **Ex-**

```
age = int(input("Enter your age: "))
print("My Age is -  " + age)
```

Output -

```
Enter your age: 28
My Age is – 28
```

➢ **Ex-**

```
per = float(input("Enter your PER(%): "))
print("My PER(%) is -  " + 78.8)
```

Output -

```
Enter your PER(%): 78.8
My PER(%) is – 78.8
```

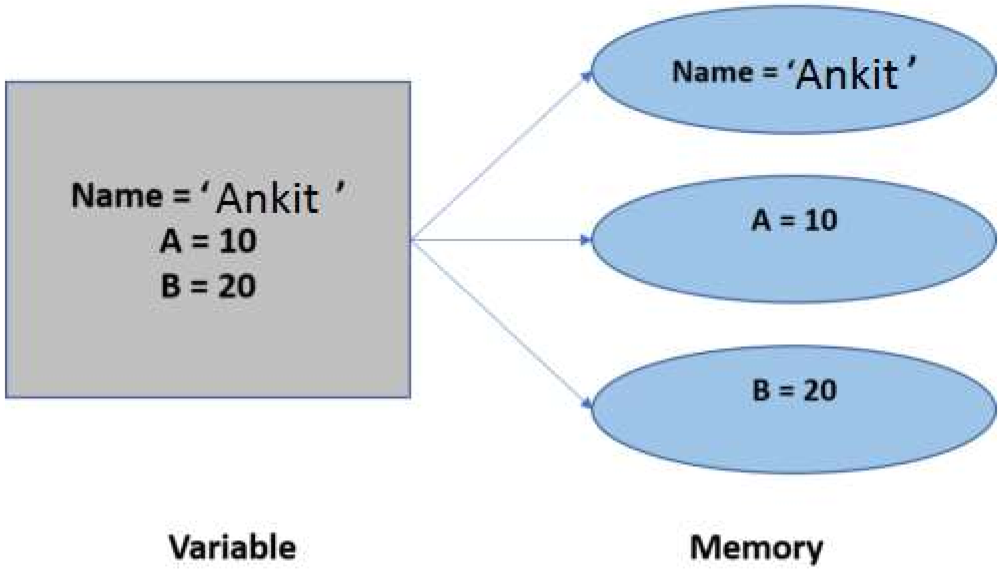
Note: By default, the python input() function takes the user's input as a string. In example 2, we had to convert the age (from string to integer) and example 3 convert per(string to float), which was inputted by the user, using the int() and float() along with the input function.

What Are Variables In Python?

✓ Variables and data types in python as the name suggests are the values that vary. In a programming language, a variable is a memory location where you store a value. The value that you have stored may change in the future according to the specifications.

✓ A Python Variable is created as soon as a value is assigned to it. It does not need any additional commands to declare a variable in python.

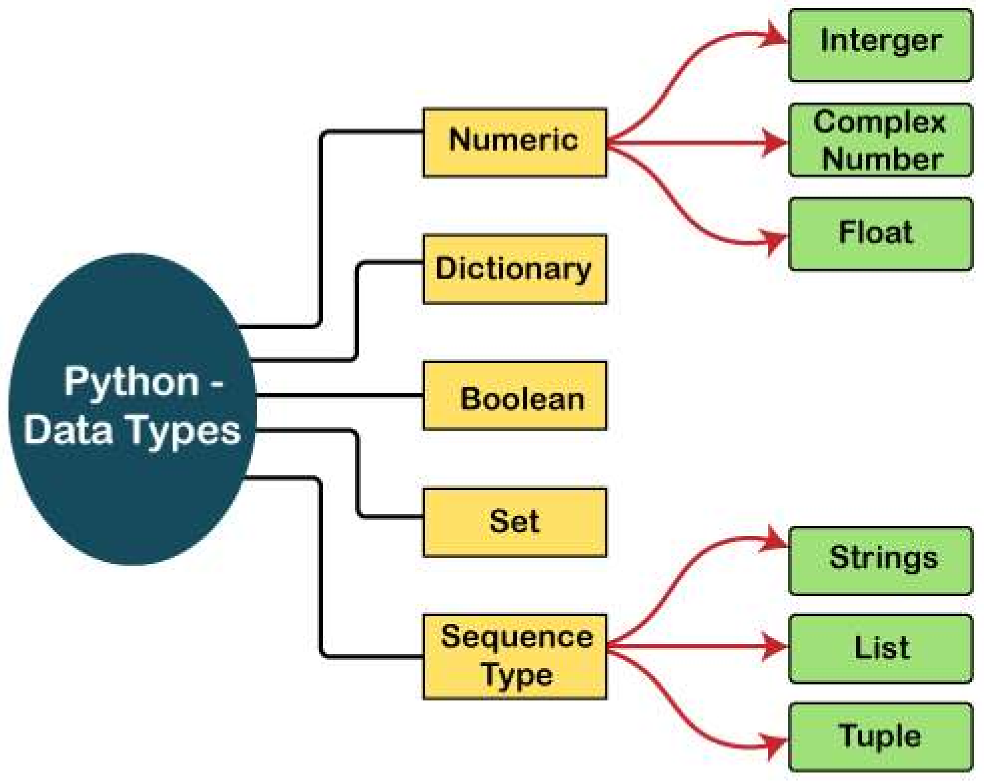
- ✓ There are a certain rules and regulations we have to follow while writing a variable, let’s take a look at the variable definition and declaration to understand how we declare a variable in python.



Variable Definition & Declaration

- ✓ Python has no additional commands to declare a variable. As soon as the value is assigned to it, the variable is declared.
- ✓ x = 10
- ✓ #variable is declared as the value 10 is assigned to it.
- ✓ There are a certain rules that we have to keep in mind while declaring a variable:
- ✓ The variable name cannot start with a number. It can only start with a character or an underscore.
- ✓ Variables in python are case sensitive.
- ✓ They can only contain alpha-numeric characters and underscores.
- ✓ No special characters are allowed.

Python Data Types



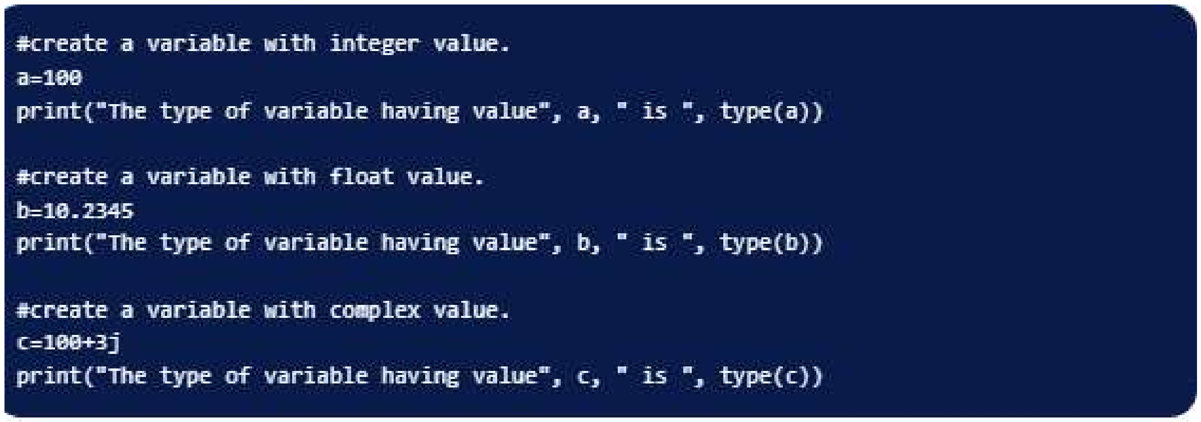
There are different types of data types in Python. Some built-in Python data types are:

1. Numeric data types: int, float, complex
2. String data types: str
3. Sequence types: list, tuple, range
4. Binary types: bytes, bytearray, memoryview
5. Mapping data type: dict
6. Boolean type: bool
7. Set data types: set, frozenset

1. **Python Numeric Data Type**

- ✓ Python numeric data type is used to hold numeric values like;
- ✓ int - holds signed integers of non-limited length.

- ✓ long- holds long integers(exists in Python 2.x, deprecated in Python 3.x).
- ✓ float- holds floating precision numbers and it's accurate up to 15 decimal places.
- ✓ complex- holds complex numbers.
- ✓ In Python, we need not declare a datatype while declaring a variable like C or C++. We can simply just assign values in a variable. But if we want to see what type of numerical value is it holding right now, we can use type(), like this

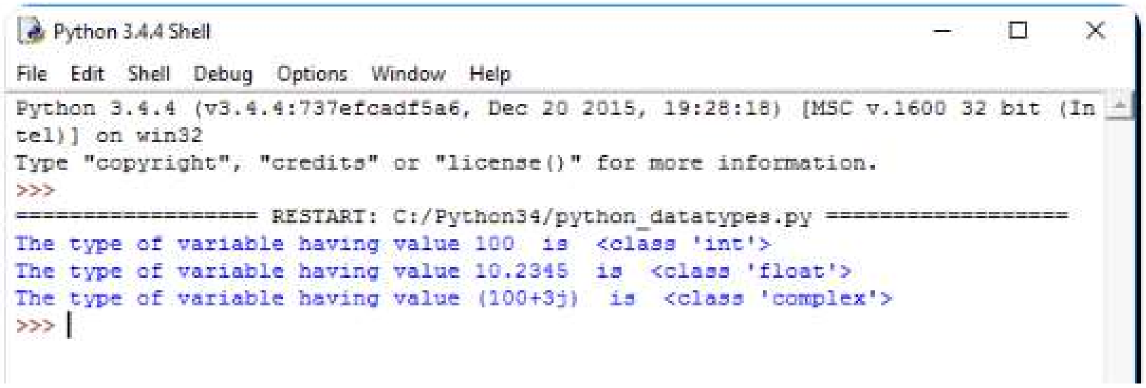


```
#create a variable with integer value.
a=100
print("The type of variable having value", a, " is ", type(a))

#create a variable with float value.
b=10.2345
print("The type of variable having value", b, " is ", type(b))

#create a variable with complex value.
c=100+3j
print("The type of variable having value", c, " is ", type(c))
```

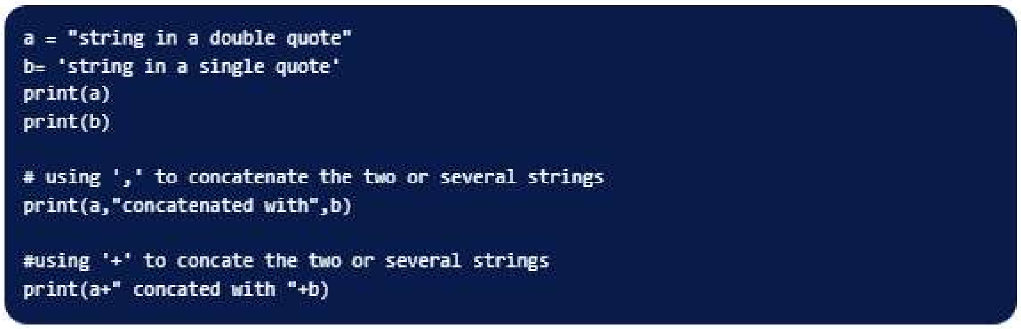
f you run the above code you will see output like the below image.



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Python34/python_datatypes.py ==================
The type of variable having value 100  is  <class 'int'>
The type of variable having value 10.2345  is  <class 'float'>
The type of variable having value (100+3j)  is  <class 'complex'>
>>> |
```

2. Python String Data Type

✓ The string is a sequence of characters. Python supports Unicode characters. Generally, strings are represented by either single or double-quotes.

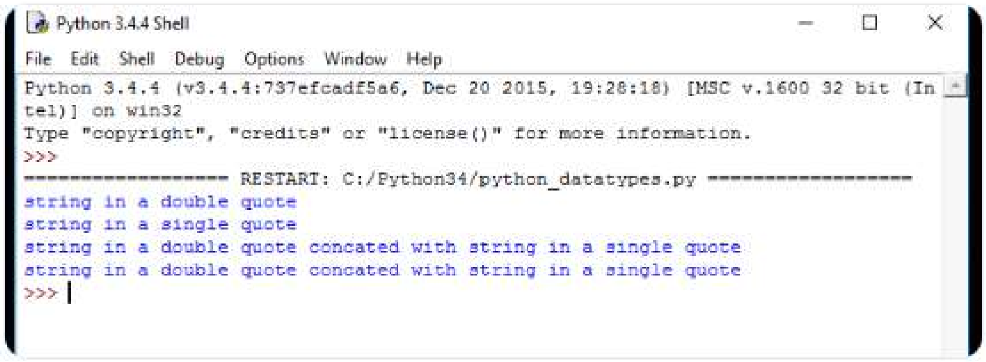


```
a = "string in a double quote"
b= 'string in a single quote'
print(a)
print(b)

# using ',' to concatenate the two or several strings
print(a,"concatenated with",b)

#using '+' to concate the two or several strings
print(a+" concated with "+b)
```

The above code produces output like the below picture-



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Python34/python_datatypes.py ==================
string in a double quote
string in a single quote
string in a double quote concated with string in a single quote
string in a double quote concated with string in a single quote
>>> 
```

3. Python List Data Type

✓ The list is a versatile data type exclusive in Python. In a sense, it is the same as the array in C/C++. But the interesting thing about the list in Python is it can simultaneously hold different types of data. Formally list is an ordered sequence of some data written using square brackets([]) and commas(,).

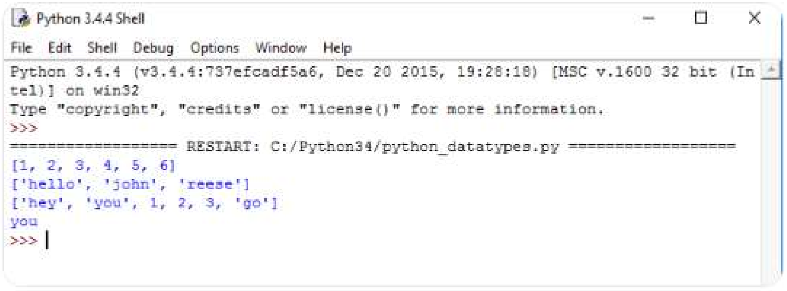
```
#list of having only integers
a= [1,2,3,4,5,6]
print(a)

#list of having only strings
b=["hello","john","reese"]
print(b)

#list of having both integers and strings
c= ["hey","you",1,2,3,"go"]
print(c)

#index are 0 based. this will print a single character
print(c[1]) #this will print "you" in list c
```

The above code will produce output like this-



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Python34/python_datatypes.py ==================
[1, 2, 3, 4, 5, 6]
['hello', 'john', 'reese']
['hey', 'you', 1, 2, 3, 'go']
you
>>> |
```

4. Python Tuple

- ✓ The tuple is another data type which is a sequence of data similar to a list. But it is immutable. That means data in a tuple is write-protected. Data in a tuple is written using parenthesis and commas.

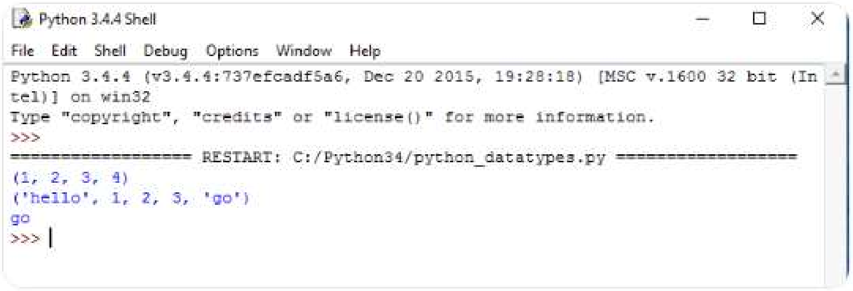
```
#tuple having only integer type of data.
a=(1,2,3,4)
print(a) #prints the whole tuple

#tuple having multiple type of data.
b=("hello", 1,2,3,"go")
print(b) #prints the whole tuple

#index of tuples are also 0 based.

print(b[4]) #this prints a single element in a tuple, in this case "go"
```

The output of this above python data type tuple example code will be like the below image.



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Python34/python_datatypes.py ==================
(1, 2, 3, 4)
('hello', 1, 2, 3, 'go')
go
>>> |
```

5. Python Dictionary

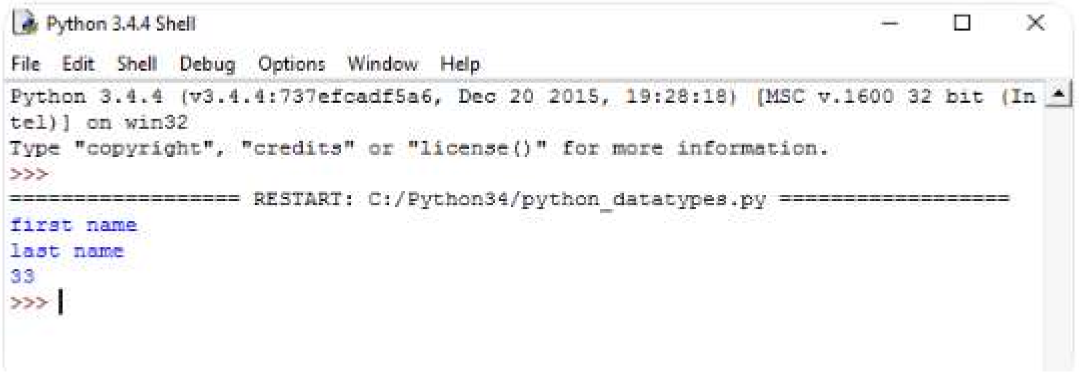
✓ Python Dictionary is an unordered sequence of data of key-value pair form. It is similar to the hash table type. Dictionaries are written within curly braces in the form key:value. It is very useful to retrieve data in an optimized way among a large amount of data.

```
#a sample dictionary variable

a = {1:"first name",2:"last name", "age":33}

#print value having key=1
print(a[1])
#print value having key=2
print(a[2])
#print value having key="age"
print(a["age"])
```

If you run this python dictionary data type example code, the output will be like the below image.

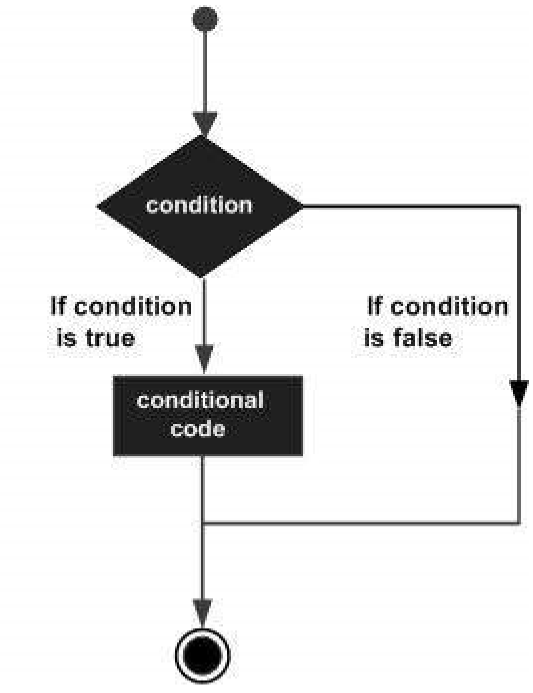


```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 19:28:18) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================== RESTART: C:/Python34/python_datatypes.py ==================
first name
last name
33
>>> |
```

Control statement (Decision making, looping, branching) implementation in python

Decision Making in Python()

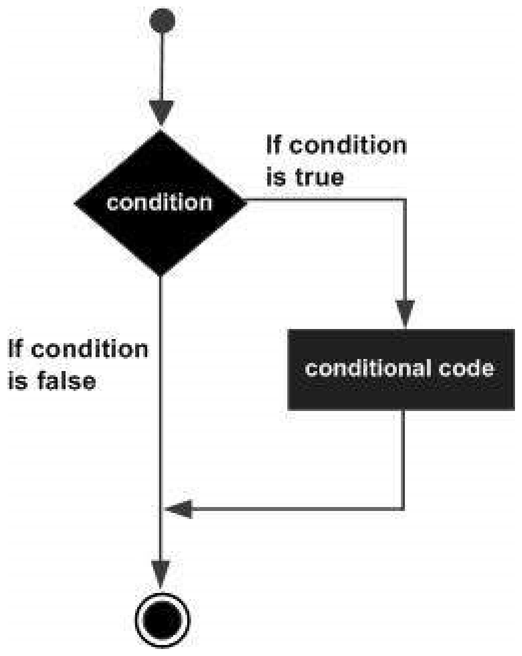
- ✓ Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.
- ✓ Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.



- ✓ Python programming language provides following types of decision making statements.

1. If statements (if Else)

- An if statement consists of a Boolean expression followed by one or more statements.
- It is similar to that of other languages. The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.



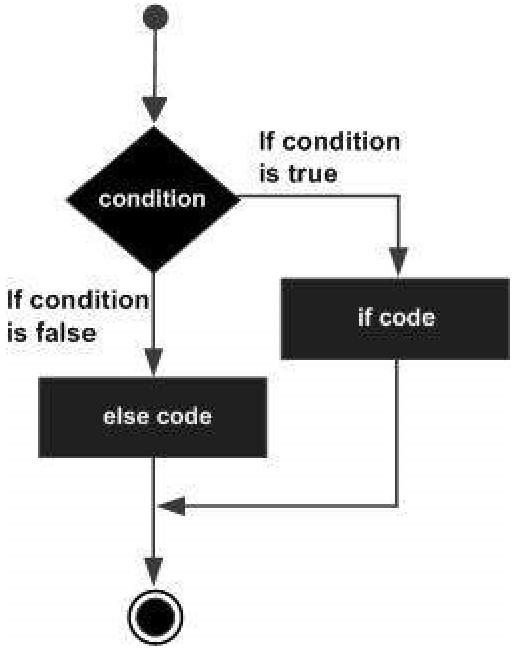
- **Ex-**

```
var1 = 100
if var1==100:
  print " true expression value"
```

- **Output – True**

2. If...else statements

- An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.
- The else statement is an optional statement and there could be at most only one else statement following if..



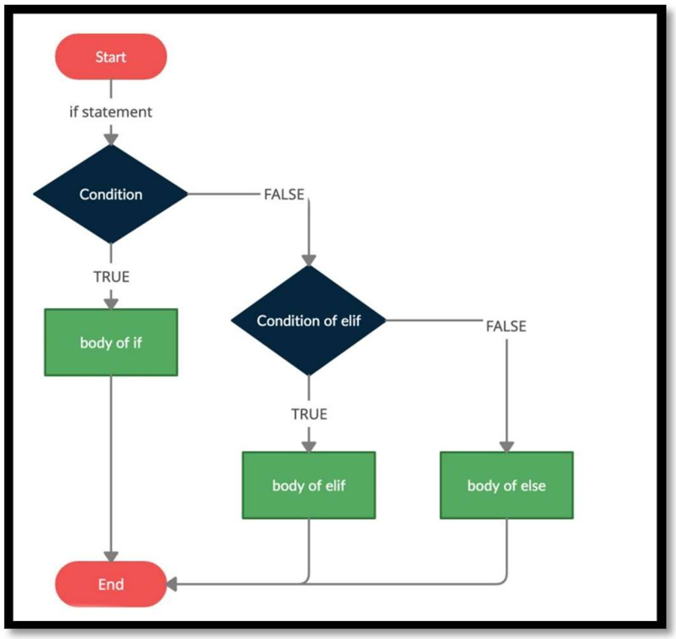
- **Ex-**

```
var1 = 100
if var1==100:
   print " true expression value"
else:
   print " false expression value"
```

- **Output – True**

3. If...elif…else statements

- The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to the else, the elif statement is optional. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.



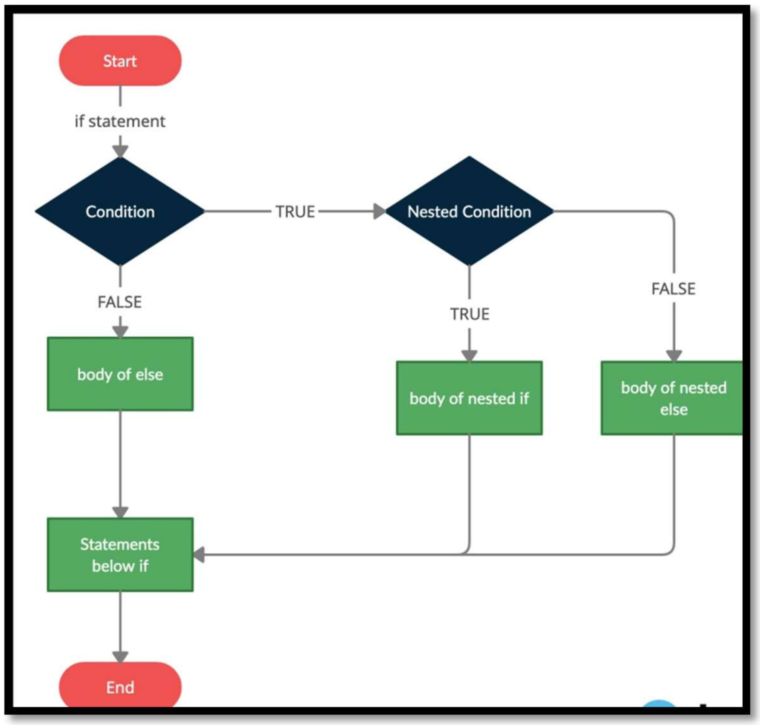
- **Ex-**

```
var1 = 200
if var1==100:
   print " 100 value true expression value"
elseif(var1==200):
   print " 200 value true expression value"
else:
   print " false expression value"
```

- **Output – 200 value true expression value**

4. Nested if statements

- There may be a situation when you want to check for another condition after a condition resolves to true. In such a situation, you can use the nested if construct.
- In a nested if construct, you can have an if...elif...else construct inside another if...elif...else construct.



- **Ex-**

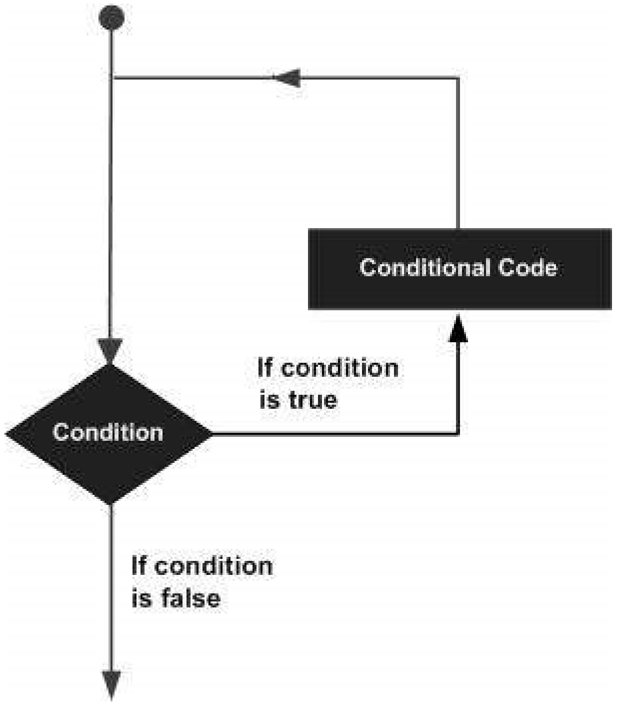
```
var1 = 200
if var1==100:
   print " 100 value true expression value"
   if var1<=100:
       print "Value is lees than 100"
   else:
      print "Value is more than 100"
else:
   print " false expression value"
```

- **Output – 100 value true expression value**

  Value is more than 100

Looping in Python()

- In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.
- Programming languages provide various control structures that allow for more complicated execution paths.
- A loop statement allows us to execute a statement or group of statements multiple times.



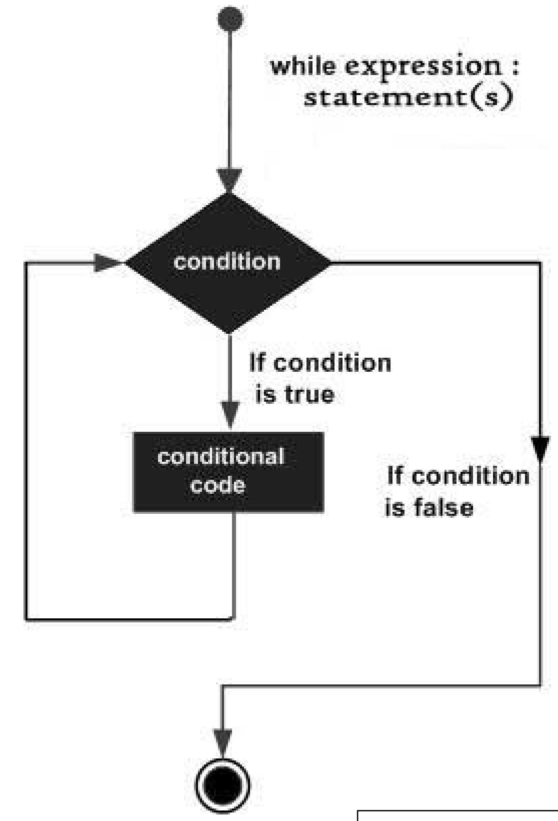
- Python programming language provides following types of loops to handle looping requirements.

1.while loop

- Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

PROGRAMMING WITH PYTHON

- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.
- In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.



- Ex-

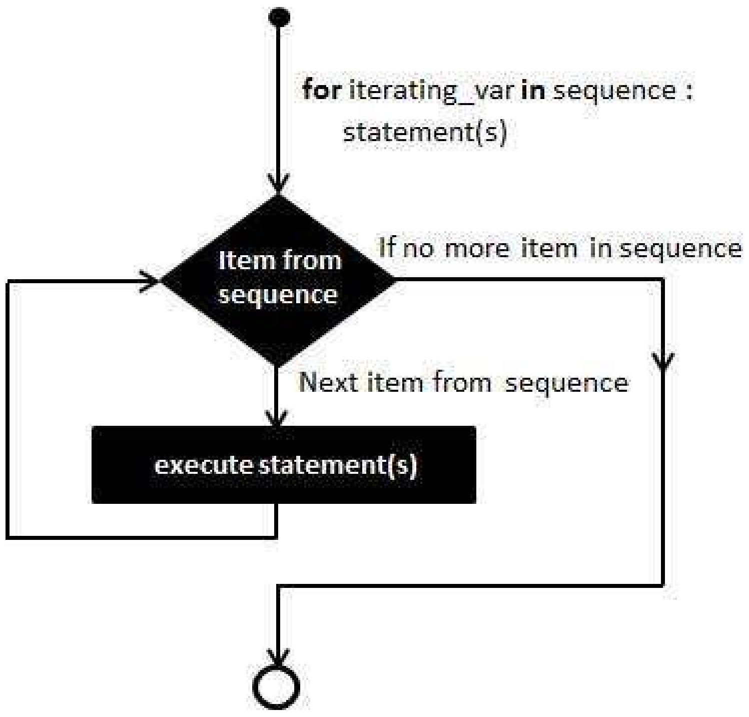
```
count = 0
while (count < 6):
   print 'The count is:', count
   count = count + 1
```

- **Output -**

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
```

2. for loop

- Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
- It has the ability to iterate over the items of any sequence, such as a list or a string.
- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed. Each item in the list is assigned to iterating_var, and the statement(s) block is executed until the entire sequence is exhausted.



- **Ex-**

```
for letter in 'Python':     # First Example
   print 'Current Letter :', letter
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:        # Second Example
   print 'Current fruit :', fruit
```

```
for a in range(0,5):     # Third Example
   print("Value is",a)
```

- **Output-**

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Value is 0
Value is 1
Value is 2
Value is 3
Value is 4
```

3.nested loops

- You can use one or more loop inside any another while, for or do..while loop.
- Python programming language allows to use one loop inside another loop.
- **Ex-**

```
i = 2
while(i < 20):
   j = 2
   while(j <= (i/j)):
      if not(i%j): break
      j = j + 1
   if (j > i/j) : print i, " is prime"
   i = i + 1
```

- **Output -**

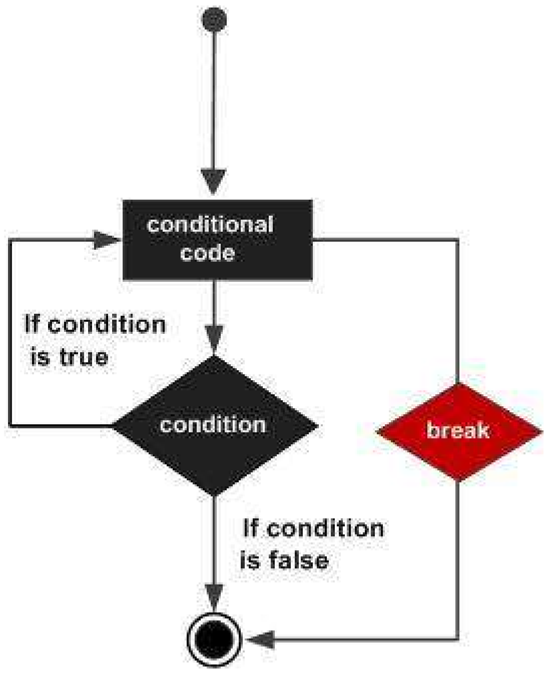
```
2  is prime
3  is prime
5  is prime
7  is prime
11  is prime
13  is prime
17  is prime
19  is prime
```

Loop Control Statements (Branching Statements)

- Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Python supports the following control statements.

1. break statement

- Terminates the loop statement and transfers execution to the statement immediately following the loop.
- The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.
- If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.



- **Ex-**

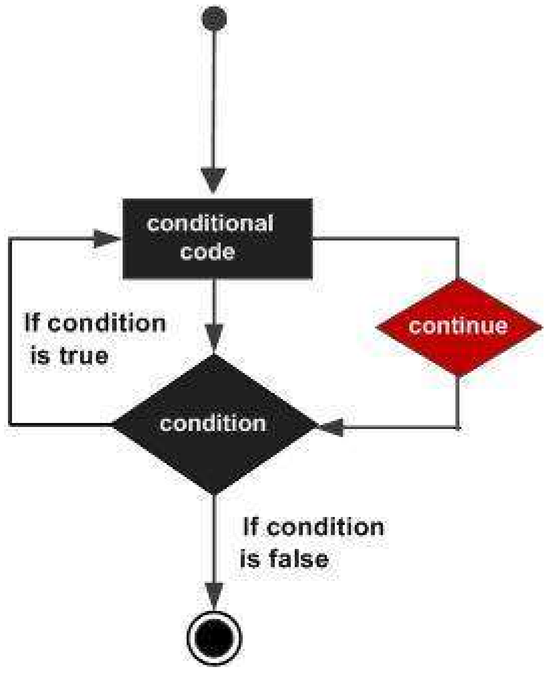
```
for letter in 'Python':
   if letter == 'h':
      break
print('Current Letter :', letter)
```

- **Output -**

```
Current Letter : P
Current Letter : y
Current Letter : t
```

2. Continue statement

- Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- It returns the control to the beginning of the while loop.. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- The continue statement can be used in both while and for loops.



- **Ex-**

```
for letter in 'Python':
   if letter == 'h':
      continue
print('Current Letter :', letter)
```

- **Output -**

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

3. pass statement

- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- It is used when a statement is required syntactically but you do not want any command or code to execute.
- The pass statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet
- **Ex-**

```
for letter in 'Python':
    if letter == 'h':
      pass
      print('This is pass block')
    print('Current Letter :', letter)
```

- **Output -**

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
```

Reference Link

1. https://www.geeksforgeeks.org/input-and-output-in-python/
2. https://www.edureka.co/blog/variables-and-data-types-in-python/
3. https://www.tutorialspoint.com/python/python_decision_making.htm
4. https://www.tutorialspoint.com/python/python_loops.htm

Any Query Contact Us

Faculty Name- Ankit Rami

Email – ankitramiblog@gmail.com

Contact No – +91 8460467193

Website - amit.arinfoway.com

Subscribe Our YouTube Channel

https://www.youtube.com/channel/UCWbJh2iQ8w-8nrU0Xpjpw7g