NPN Training

Training is the essence of success and we are committed to it.

PYTHON - Fundamentals

Naveen P.N

Topics for the Module

After completing the module, you will be able to understand:

- Introduction to Python
- History of Python
- Installing Python in Windows using PyCharm
- Features and Applications of Python
- Reasons to Choose Python
- Data Types
- Control Flow Statements

Introduction to Python

Python is free & open-source cross platform language, and it is also a powerful high-level, object-oriented programming language which has simple easy-to-use syntax.

- Python is a general-purpose language.
- It has wide range of applications like
  - Web development – (Django and Bottle)
  - Scientific and mathematical computing – (Orange, SymPy, NumPy)
  - Desktop graphical user Interfaces – (Pygame, Panda3D).

History of Python

- Python is a fairly old language created by Guido Van Rossum and its design began in the late 1980s and was first released in February 1991.

- Rossum wanted to create an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba Distributed Operating system calls, which is extensible.

- The name "Python" was adopted from TV series "Monty Python's Flying Circus“ in late seventies which was a favorite show of Rossum. But not from the dangerous snake **PYTHON.**

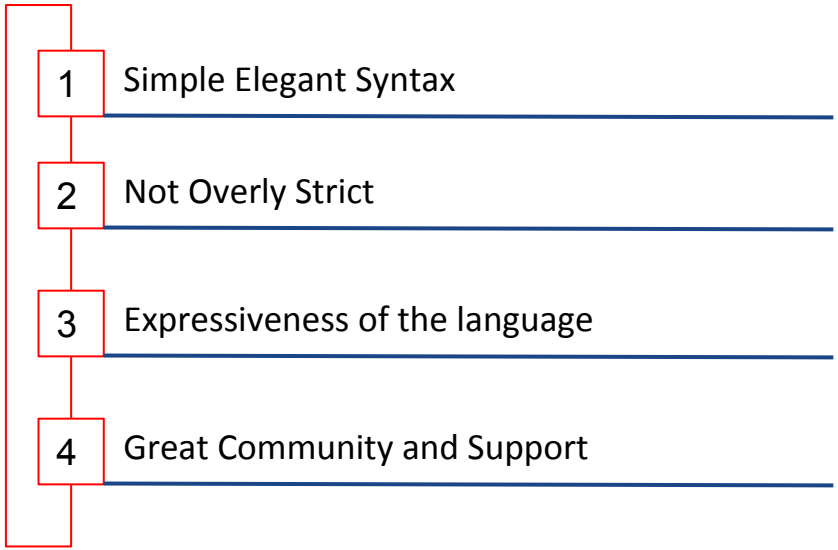
Features & Applications of PYTHON

1. A simple language which is easier to learn
2. Free and open-source
3. Portability
4. Extensible and Embeddable
5. A high-level, interpreted language
6. Large standard libraries to solve common tasks
7. Object-Oriented

Applications:

- ❑ Web Applications
- ❑ Scientific and Numeric Computing
- ❑ Creating software Prototypes
- ❑ Good Language to Teach Programming

Reasons to Choose Python as First Language



Hello, World! – Python Program [Hands-on]

- Very often we see a program called "Hello, World!" is used to introduce a new programming language to beginners. A "Hello, World!" is a simple program that outputs "Hello, World!".
- However, Python is one of the easiest language to learn, and creating "Hello, World!" program. It is as simple as writing print("Hello, World!").
- There are various ways to start python
  - **Immediate Mode** - Typing python in the command line will invoke the interpreter in immediate mode.

    Eg: >>>
  - **Script Mode** - This mode is used to execute Python program written in a file. Such a file is called a script. Scripts can be saved to disk for future use. Python scripts have the extension .py, meaning that the filename ends with .py. Eg: python helloWorld.py
  - **IDE Mode -** Using Ide like PyScripter, we can execute a program. By using IDE we can decrease the time required for application Development

</>

helloWorld.py

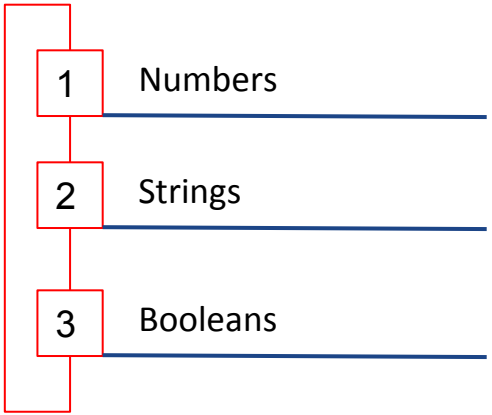
```
print("Hello, World!")
```

Execute

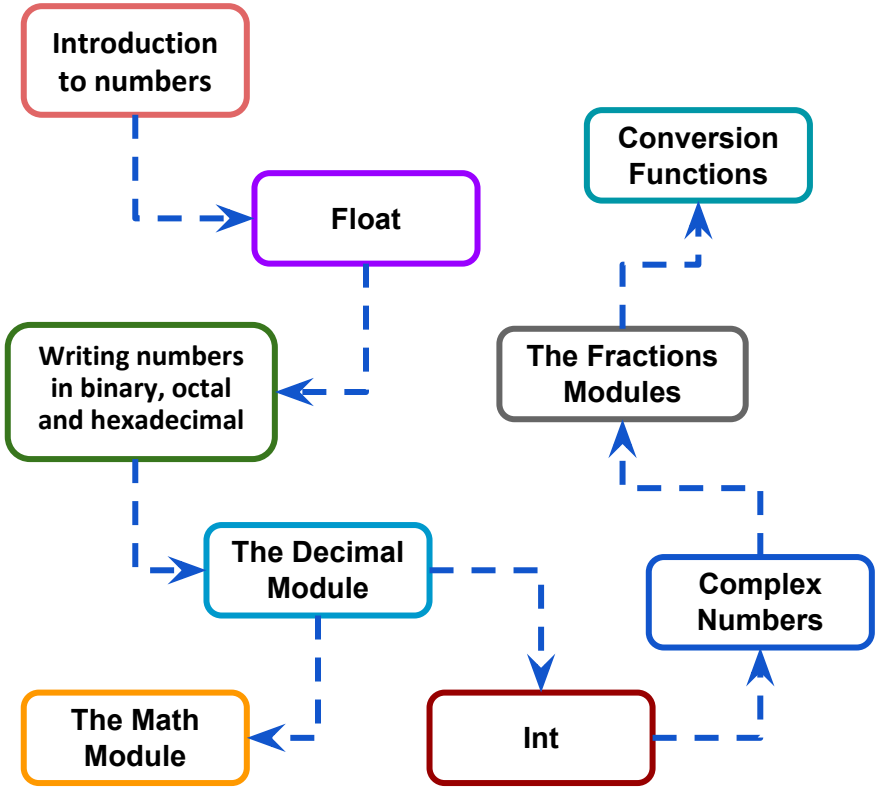
```
python helloWorld.py
```

Data Types

- Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
- There are various data types in Python. Some of the important types are listed below.



Numbers Type



Note: Long is no longer supported by Python 3.x.

- Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.
- Python can hold signed integers .It can hold a value of any length, the only limitation being the amount of memory available.
- Python also supports floating-point real values. An int cannot store the value of the mathematical constant pi, but a float can. Float is only accurate upto 15 decimal places. After that, it rounds the number off.
- A complex number is a Python number type made of real and imaginary parts. It is represented as a+bj. Where **a** is real part and **bj** is the imaginary part. To Denote the irrational part we cannot use the letter i or other alphabets other than j as we normally do.

Numbers Type Contd.. [Hands-on]

- We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

```
a = 555555555555555555555555555555555555
print(a, "is of type", type(a))
a = 1.1111111111111119
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
a = 2+3j
b = 3+5j
Print("The sum of Complex numbers is - ", a+b)
```

OUTPUT:

```
555555555555555555555555555555555555 is of type <class 'int'>
1.1111111111111112 is of type <class 'float'>
(1+2j) is complex number? True
The sum of Complex numbers is - 5+8j
```

Writing numbers in binary, octal, and hexadecimal

- More often than not, programmers need to deal with numbers other than decimal. To do this, you can use appropriate prefixes. Please the add the following prefixes to get appropriate number.

Number System	Prefix
Binary	0b or 0B
Octal	0o or 0O
HexaDecimal	0x or 0X

```
print("The actual value of Binary number 0b111 is - ",0b111)
int(0b10)
print("The actual value of Octal number 0O10 is - ",0O10)
float(0B10)
print("The actual value of HexaDecimal number 0xFF is - ",0xFF)
```

Output:

```
The actual value of Binary number 0b111 is - 7
2
The actual value of Octal number 0O10 is - 8
2.0
The actual value of HexaDecimal number 0xFF is - 255
```

Conversion Functions [Hands-on]

1. int()
2. float()
3. complex()
4. bin()
5. oct()
6. hex()

```
int(7), int (7.7) & int(2+3j)  → 7, 8 & TypeErro can't convert complex
to int

float(7), float (7.0) & float(0o10) → 7.0, 7.0 & 8.0

complex(2), complex(2.3) & complex(2+3.0j) → 2+0j, 2.3+0j & 2+3j

bin(2), bin(2.3) & bin(2+3j) → 0b10, TypeErro 'float' object cannot be
interpreted as an integer, TypeErro 'complex' object cannot be
interpreted as an integer

oct(8), oct(8.3) → 0o10, TypeErro 'float' object cannot be interpreted
as an integer

hex(255) & hex(0) → 0xff & 0x0
```

Fractions & Math Module [Hands-on]

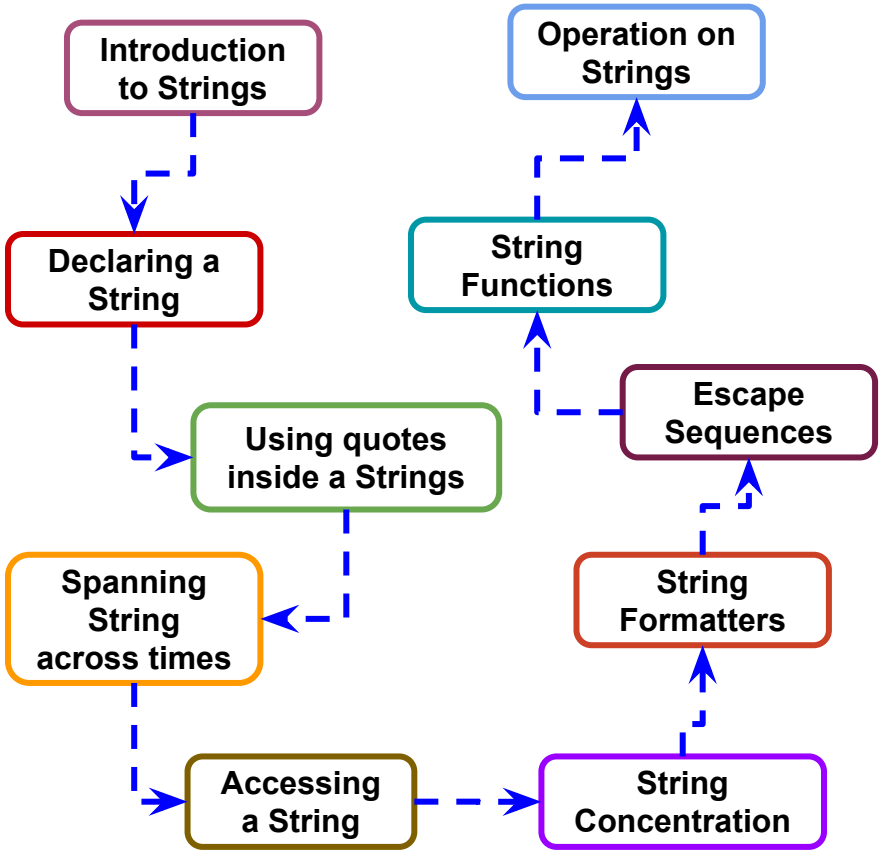
- Fractions module lets you deal with fractions. The Fraction() function returns the value in the form of numerator and denominator. It can also take two arguments. We can print the fraction of 1.5 as follows:

```
>>> from fractions import Fraction
>>> print(Fraction(1.5))
3/2
>>> print(Fraction(1,3))
1/3
```

- Math Module has all important mathematical functions like exp, trigonometric functions, logarithmic functions, factorial, and more.

```
>>> import math
>>> math.factorial(5)
120
>>>  math.exp(3)
20.085536923187668
>>>  math.tan(90)
-1.995200412208242
```

Strings



- Python string is a sequence of characters. There is a built-in class 'str' for handling Python string. You can prove this with the type() function. Python does not have char data type like c++ and java.

```
>>> type('Welcome to NPNTraining')
<class 'str'>
```

Declaring Python String

- We can declare a string using a single or double quotes.

```
>>> a = 'welcome to "NPNTraining"'
>>> print(a)
Welcome to "NPNTraining"
>>> a = "welcome to 'NPNTraining'"
>>> print(a)
Welcome to 'NPNTraining'
```

Accessing the Python Strings [Hands-on]

- A string is immutable. It can't be changed.

```
>>> a = "NPNTraining"
>>> a
'NPNTraining'
>>> a[0]
'N'
>>> a[0]="T"
Traceback (most recent call
last):
File "<stdin>", line 1, in
<module>
TypeError: 'str' object does
not support item assignment
>>> a[3:8]
'Train'
>>> a[:8]
'NPNTrain'
```

```
>>> a[8:]
'ing'
>>>a[:]
'NPNTraining'
>>> a[:-3]
'NPNTrain'
>>> a[-3:]
'ing'
>>> a[-5:-3]
'in'
>>> a[-5:-5]
''
>>> a[2:2]
''
```

String Formatters [Hands-on]

- When we want to print the variable values along with a string, we either use comma or formatters to do the same.

```
>>> a = "NPNTraining"
>>> print("I am ",21," Years old. Studying at ",a)
I am  21  Years old. Studying at  NPNTraining
```

- We can use f-string for printing the same.The letter ‘f’ precedes the string, and the variables are mentioned in curly braces in their places.

```
>>> print(f"I am 21 Years old. Studying at {a}")
I am 21 Years old. Studying at NPNTraining
```

- You can use the format() method to do the same. It succeeds the string, and has the variables as arguments separated by commas. Inside the curly braces, you can either put 0,1,.. or the variables

```
>>> print("I am 21 Years old. Studying at {0}".format(a))
I am 21 Years old. Studying at NPNTraining
print("I am {age} Years old. Studying at {a}".format(age=21,a='xyz'))
I am 21 Years old. Studying at xyz
```

Python Escape Sequences

- In a string, you may want to put a tab, a linefeed, or other such things. Escape sequences allow us to do this.
- An escape sequence is a backslash followed by a character, depending on what you want to do. Python supports the following sequences.
  - \n – linefeed
  - \t – tab

```
>>> print("NPN\tTraining")
NPN     Training
```

- Since a backslash may be a part of an escape sequence, so, a backslash must be escaped by a backslash too.

  >>> \\

- \' – A single quote can be escaped by a backslash. This lets you use single quotes freely in a string.
- \" – Like the single quote, the double quote can be escaped too.

String Functions [Hands-on]

- Python provides us with a number of functions that we can apply on strings or to create strings.

```
>>> len(a)
11
>>> len(a[3:8])
5
>>> str(2+3j)
'(2+3j)'
>>> str(['red','green','blue'])
"['red', 'green', 'blue']"
>>> a.lower()
'npntraining'
>>> a.upper()
'NPNTRAINING'
>>> b=" NPNTRAINING "
>>> b.strip()
'NPNTRAINING'
```

```
>>> b='555'
>>> b.isdigit()
True
>>> a.isalpha()
True
>>> b=' '
>>> b.isspace()
True
>>> a.startswith('NPN')
True
>>> a.endswith('ng')
True
>>> a.find('NTrai')
2
I.e. the NTrai starts at position
2 in the String NPNTraining
```

String Functions contd.. [Hands-on]

```
>>> b.replace('na','ha')
'Bahaha'
>>> b="1,2,3,4,5,6"
>>> b.split(',')
['1', '2', '3', '4', '5', '6']
>>> "*".join(['red','green','blue'])
'red*green*blue'
```

Note: Join takes a list as an argument, and joins the elements in the list using the string it is applied on.

- Following are the 68 String Methods in python

capitalize(), center(), casefold(), count(), endswith(), expandtabs(), encode(), find(), format(), index(), isalnum(), isalpha(), isdecimal(), isdigit(), isidentifier(), islower(), isnumeric(), isprintable(), isspace(), istitle(), isupper(), join(), ljust(), rjust(), lower(), upper(), swapcase(), lstrip(), rstrip(), strip(), partition(), maketrans(), rpartition(), translate(), replace(), rfind(), rindex(), split(), rsplit(), splitlines(), startswith(), title(), zfill(), format_map(), any(), all(), ascii(), bool(), bytearray(), bytes(), compile(), complex(), enumerate(), filter(), float(), input(), int(), iter(), len(), max(), min(), map(), ord(), reversed(), slice(), sorted(), sum(), zip()

Python String Operations [Hands-on]

- There are 5 Different Types of Categories in the String Operations.
  - **Comparison:** Strings can be compared using the relational operators.

```
>>> 'hey'<'hi'
True
Note: 'hey' is lesser than 'hi lexicographically (because
i comes after e in the dictionary)
>>> a == 'NPNTraining'
True
>>> 'yes'!='no'
True
```

  - **Arithmetic:** Some of the Arithmetics Operations can be applied on strings.

```
>>> 'ba'+'na'*2
'banana'
```

Python String Operations [Hands-on]

- **Membership:** membership operators of Python can be used to check if string is a substring to another.

```
>>> 'Train' in a
True
>>> 'aswdws' in a
False
```

- **Identity:** identity operators ‘is’ and ‘is not’ can be used on strings.

```
>>> 'NPNTraining' is a
True
>>> 'npntraining' is not a
True
```

Python String Operations [Hands-on]

- **Logical:** Python's and, or, and not operators can be applied too. An empty string has a Boolean value of False

  ✓ and- If the value on the left is True it returns the value on the right. Otherwise, the value on the left is False, it returns False.
  ✓ or- If the value on the left is True, it returns True. Otherwise, the value on the right is returned.
  ✓ not- As we said earlier, an empty string has a Boolean value of False.

</>

```
>>> '' and '1'
''
>>> '1' and ''
''
>>> '' or '1'
'1'
>>> not('1')
False
>>> not('')
True
```

Booleans

- Boolean values are the two constant objects False and True. They are used to represent truth values (other values can also be considered (false or true).
- In numeric contexts, they behave like the integers 0 and 1, respectively. The built-in function bool() can be used to cast any value to a Boolean, if the value can be interpreted as a truth value. They are written as False and True, respectively.
- A string in Python can be tested for truth value.The return type will be in Boolean value (True or False).

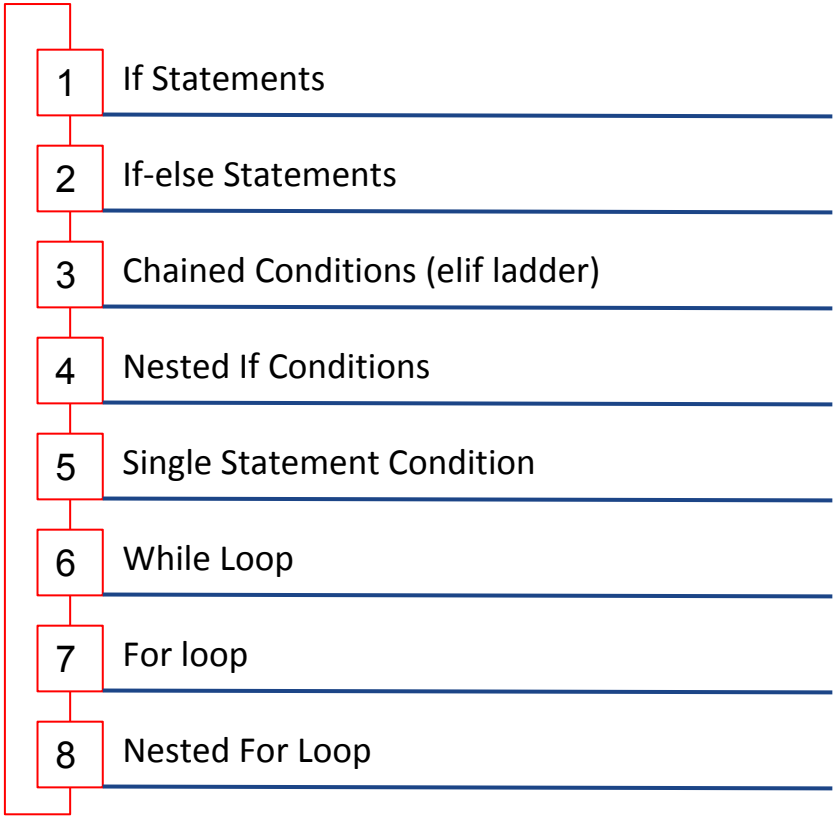
```
>>> bstring = "Hello World"
>>>>>> bstring.isalpha()  #check if all char in the string are alphabetic
False
>>> bstring.isupper() #test if string contains upper case
True
>>> bstring.startswith('H') #test if string startswith H
False
>>> 'a'>'h'
False
```

Control Flow Statements

- It is very important to control the program execution because in real scenarios the situations are full of conditions. For this you need to control the execution of your program statements.
- Python provide various flow controls. Some of them are if, if .. elif .. else, if..else, while, for, switch, pass, range, break, else, continue, function etc.
- Unlike in C++ or Java, Python does not use curly braces for indentation. Instead, it mandates indentation. In Flow Control Statements it is mandate that we use proper indentation.
- There are no strict rules on what kind of Python indentation you use. But it must be consistent throughout the block. Although, four whitespaces are usually preferred, and tabs are discouraged.

```
>>> if 2>1:
... print("1")
  File "<stdin>", line 2
    print("1")
        ^
IndentationError: expected an indented block
```

Control Flow Statements Contd..

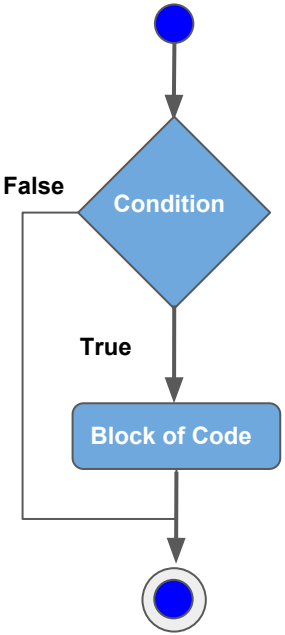


If Statement

- An if statement in python takes an expression with it. If the expression amounts to True, then the block of statements under it is executed. If it amounts to False, then the block is skipped and control transfers to the statements after the block.

- Always Remember to indent the statements in a block equally. This is because we don't use curly braces to delimit blocks. Also, use a colon(:) after the condition.

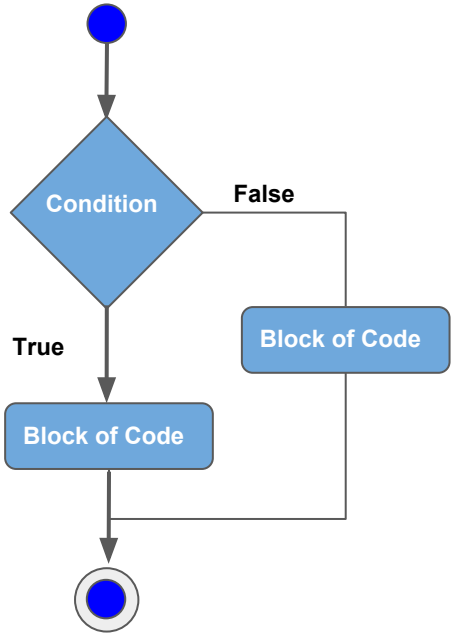
```
>>> a=7
>>> if a>6:
...     print(f"{a} is good")
...
7 is good

>>> if 1:
...     print("yay")
...
yay
```



If - else Statement

- What happens if the condition is not true? We can mention that it in the block after the else statement. An else statement comes right after the block after ‘if’.

```
>>> if 2<1:
...     print("2")
... else:
...     print("1")
...
1
* Here the else comes under the if not inside the if, so the indent has to be removed for the else statement
```

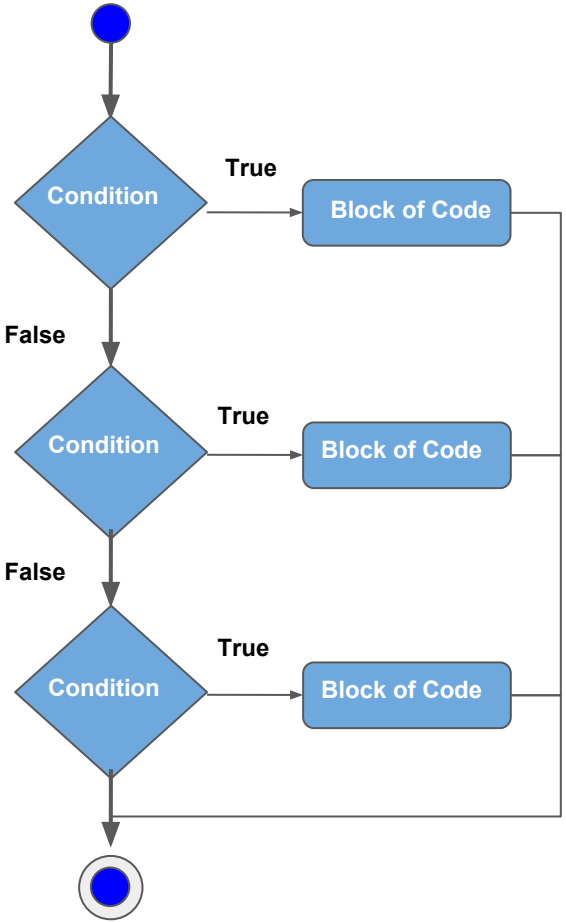


Chained Condition (elif Conditions)

- Python allows the elif keyword as a replacement to the else-if statements in Java or C++.

- When we have more than one condition to check, we can use it. If condition 1 isn't True, condition 2 is checked. If it isn't true, condition 3 is checked.

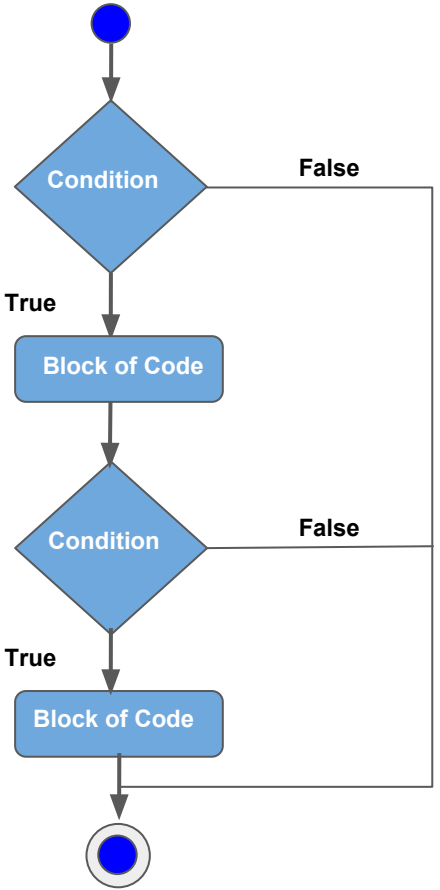
```
>>> if 2<1:
...     print("2")
... elif 3<1:
...     print ("3")
... else:
...     print("1")
...
1
* Here the else & elif comes out of if so the
indent has to be removed
```



Nested If Conditions

- We can use a if statement in the block of another if statement.
- This is to implement further check any conditions.

```
>>> a=1
>>> b=2
>>> if a==1:
...     if b==2:
...         print("a is 1 and b is 2")
...
a is 1 and b is 2

>>> if a<4: print("Greater")
...
Greater
*This is called single line statements
```

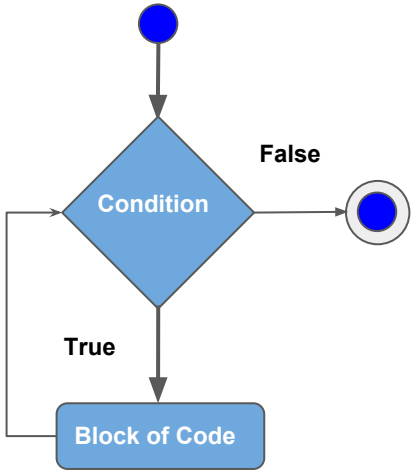


While loop

- while loop in python iterates till its condition becomes False. In other words, it executes the statements under itself while the condition it takes is True.

```
>>> a=3
>>> while(a>0):
...     print(a)
...     a-=1
...
3
2
1
```

```
>>> while(a>0):
...     print(a)
...     a-=1
...     if(a==1):
...         break;
...     else:
...
print("reached 0")
...
3
reached 0
2
```



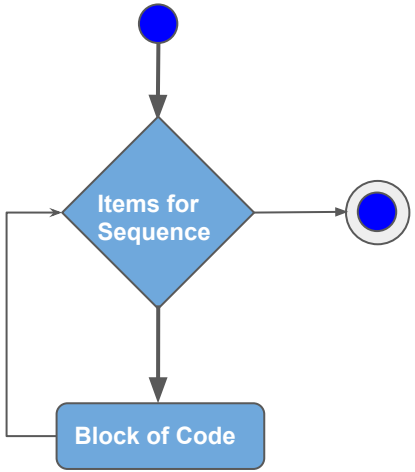
- Be careful while using a while loop. Because if you forget to increment the counter variable in python, or write flawed logic, the condition may never become false. In such a case, the loop will run infinitely, and the conditions after the loop will starve.

For Loop

- for loop can iterate over a sequence of items. The structure of a for loop in Python is different than that in C++ or Java. That is, for(int i=0;i<n;i++) won't work here. In Python, we use the 'in' keyword

```
>>> for a in range(3):
...     print(a+1)
...
1
2
3
>>> for i in "RAM":
...     print(i)
...
R
A
M
```

```
>>> for i in "RAM":
...     print(i)
...     if(i=="A"):
...         break;
...     else:
...
print("Reached else")
...
R
Reached else
A
```



Range Function

- Range function yields a sequence of numbers. When called with one argument, say n, it creates a sequence of numbers from 0 to n-1.

We use the list function to convert the range object into a list object.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2,7))
[2, 3, 4, 5, 6]
>>> list(range(12,2,-2))
[12, 10, 8, 6, 4]
>>> list(range(2,12,-2))
[]
>>> for i in {2,3,3,4}:
...     print(i)
...
2
3
4
```

Nested For Loop

- We can also nest a loop inside another. You can put a for loop inside a while, or a while inside a for, or a for inside a for, or a while inside a while. Or you can put a loop inside a loop inside a loop. You can go as far as you want.

Using Nested For Loop

```
>>> for i in range(1,6):
...     for j in range(i):
...         print("*",end='')
...     print("\n",end='')
...
*
**
***
****
*****
```

Using Nested while Loop

```
>>> i=6
>>> while(i>0):
...     j=6
...     while(j>i):
...         print("*",end='')
...         j-=1
...     i-=1
...     print("\n",end='')
...
*
**
***
****
```

aaa

Key Takeaways

Hard work beats talent
when talent fails to **work hard**.