Learn Python Programming

What is Python Programming?

- Fastest Growing Language
- Very Easy to Learn(High Level Language) with clean Syntax
- Its flexible and versatile
- Tech Giants are using python for latest tech (Web Development, Data Analysis, Data Science, GUI Applications, Game Design, web crawler/scraper, Machine Learning, AI
- Vast community of developers/programmers
- Huge open source repository
- Companies use python language: Google, Facebook, Instagram, Spotify, Quora, Netflix, Dropbox, and more.

Why Learn Python…?

Most in-demand programming languages of 2019

Based on Indeed.com job postings in the USA - Feb 1, 2019

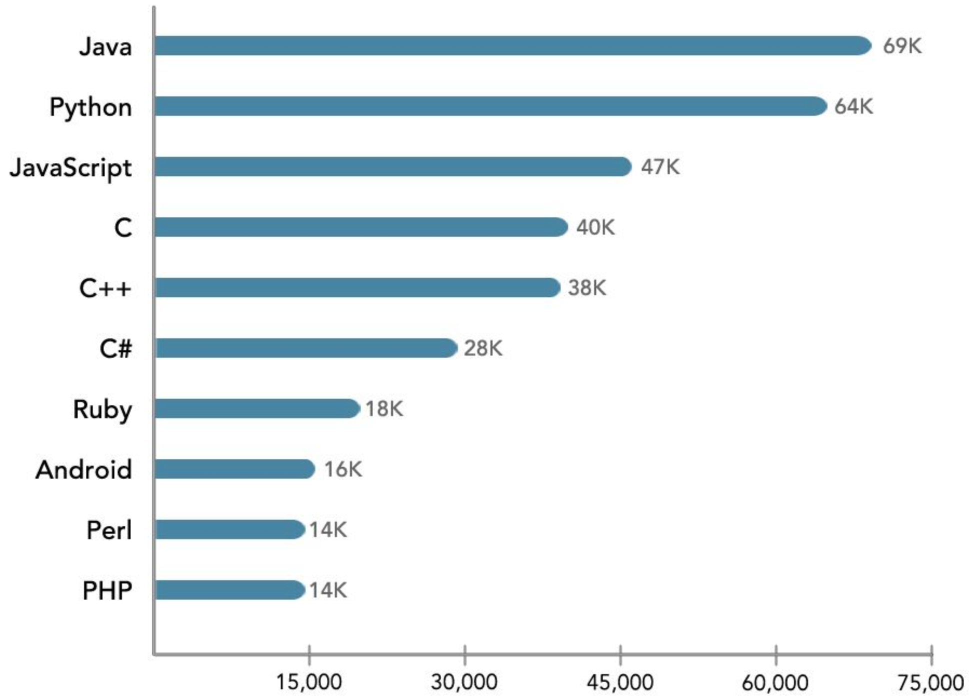
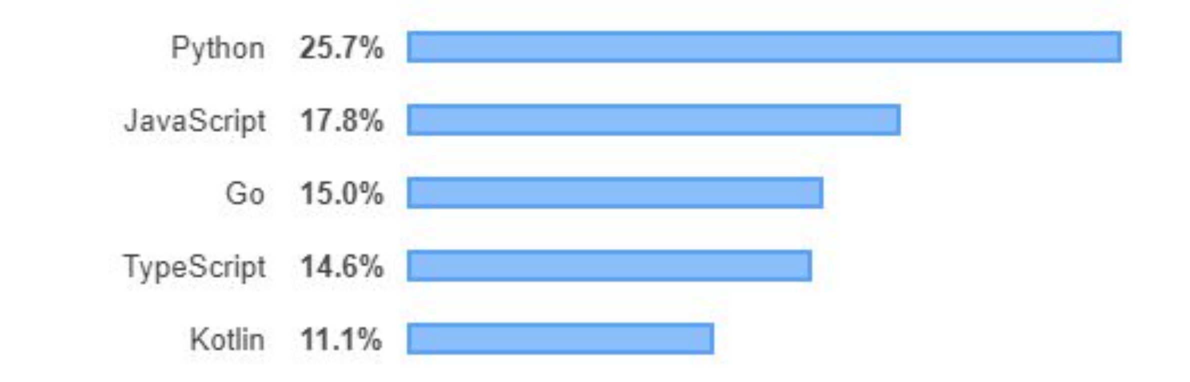
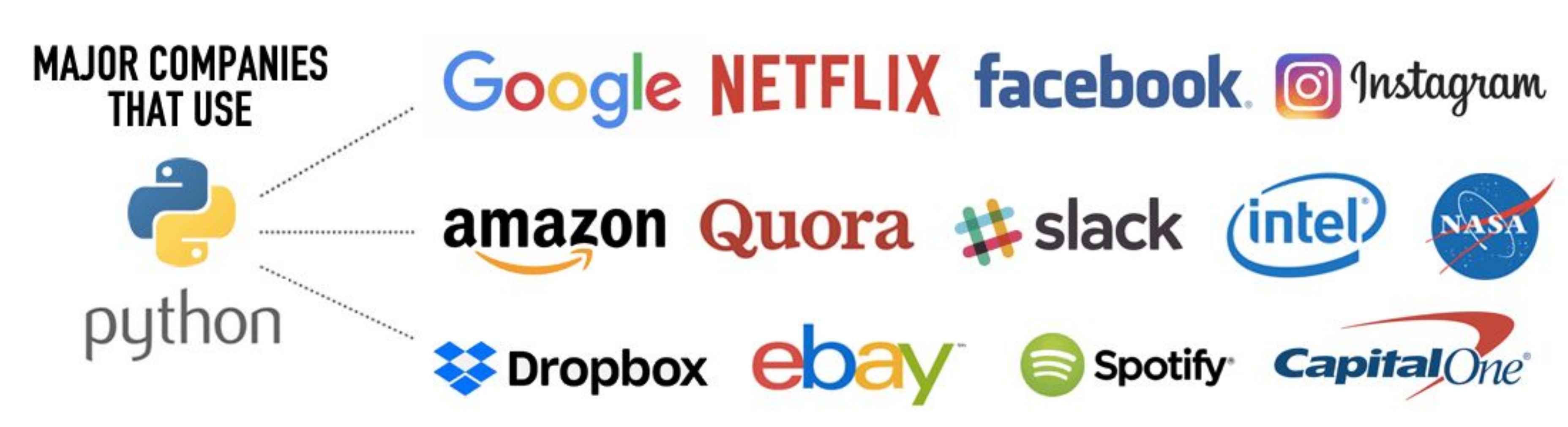


Image Source: CodingNomads

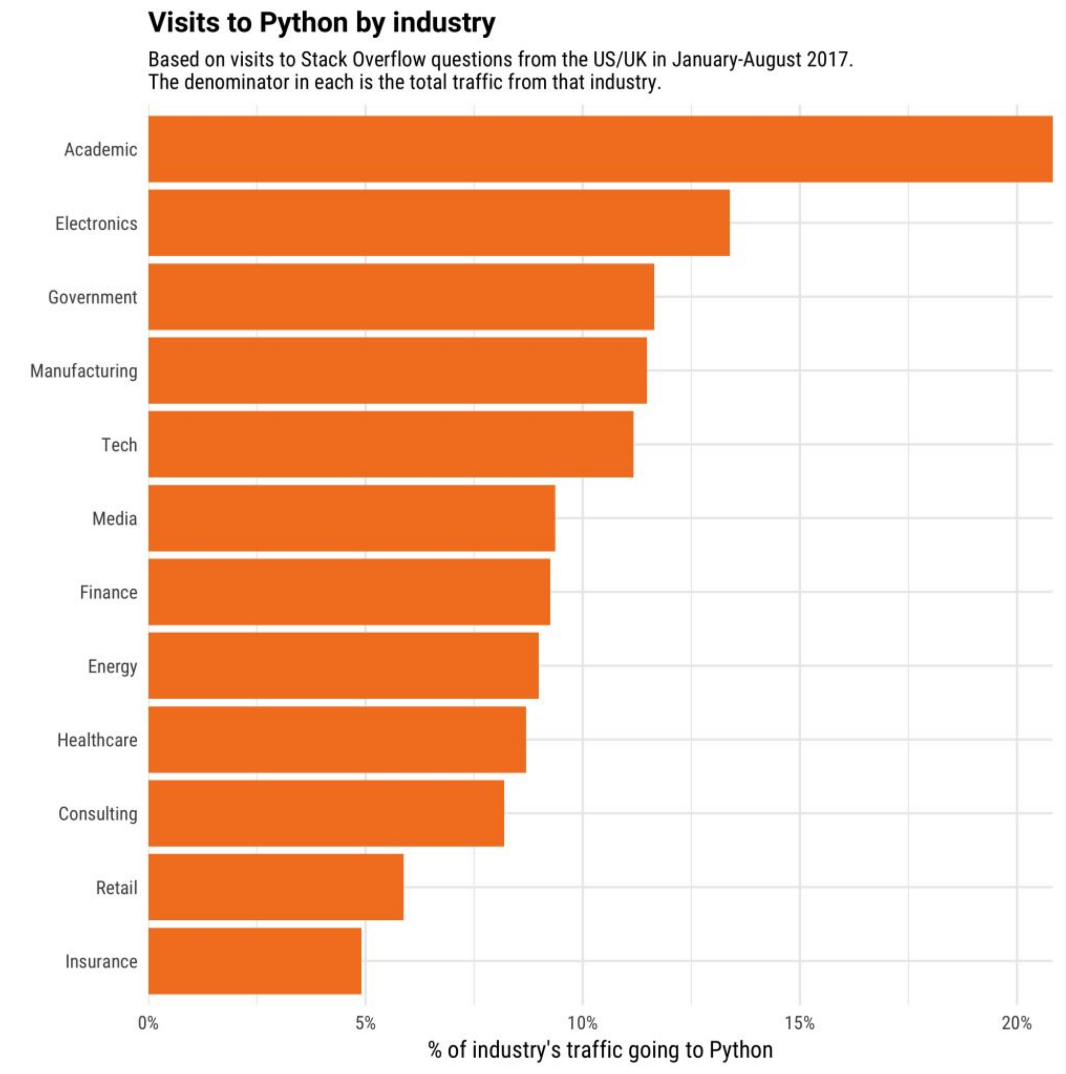
Stack Overflow’s 2019 Most Wanted Programming Languages



Companies Using Python



Python Used By Industries



History of Python

- Comes in picture in December 1989 as a hobby project by Guido van Rossum (a dutch mathematician), the creator of the Python programming language. However, the official release was in 1991.
- The name “python” was inspired from his most favourite and very famous tv show of the time.(Monty Python's Flying Circus)
- BDFL of Python: He was the “Benevolent dictator for life” and step down by BDFL of Python in july 2018

MONTY PYTHON'S FLYING CIRCUS

History of Python

His(Guido van Rossum’s) goals for Python:

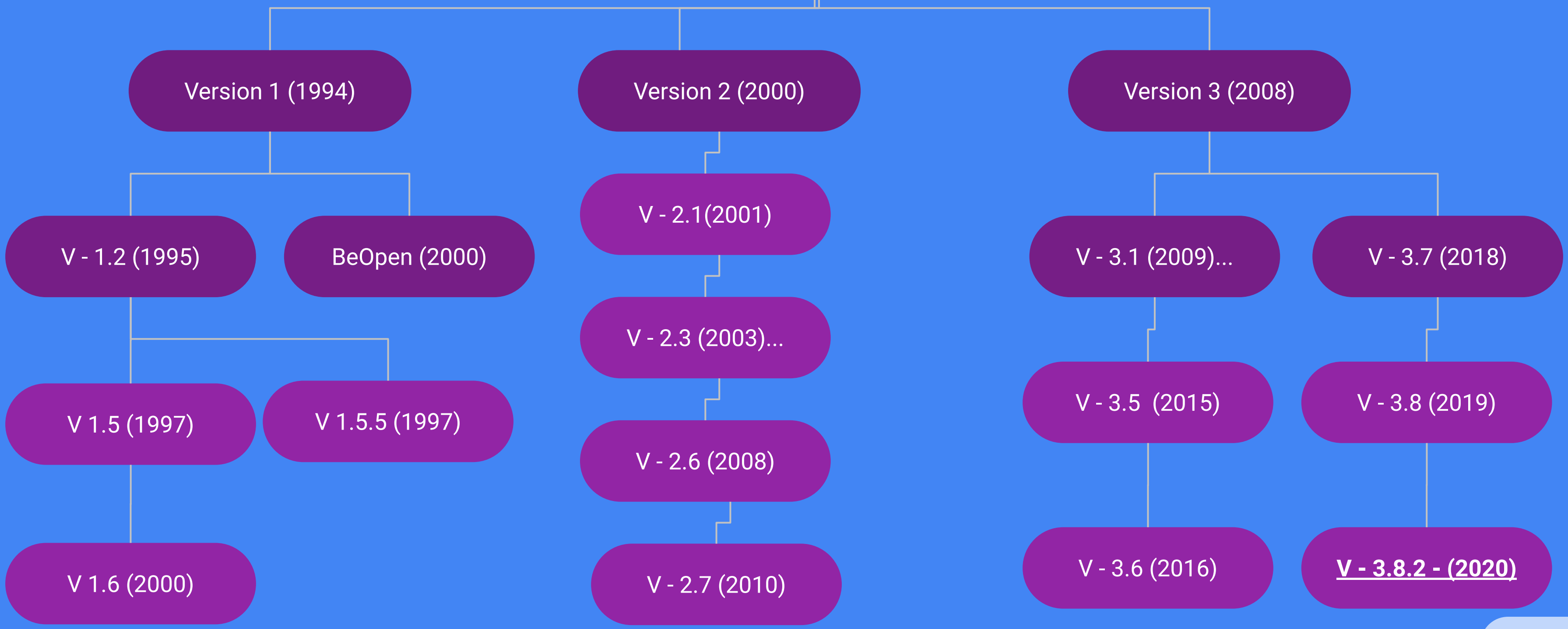
- Very easy to code, yet powerful in competitors.
- Should be open source, available to all, contribution from all, dynamic and powerful community.
- High level language, comprehensible to all as plain English
- Applicable to day to day tasks, along with small functionalities.

Python 2 vs python3

Python 2 been popular among the developer for more than a decade, and still stays in the software at certain companies.

On the other hand, python3 supports modern techniques like AI, ML, and has a vast set of libraries for data science as well. Needless to say Python3 is supported by an unmatched & huge Python developer's community.

Python Hierarchy



Version 1 (1994)

V - 1.2 (1995)

BeOpen (2000)

V 1.5 (1997)

V 1.5.5 (1997)

V 1.6 (2000)

Version 2 (2000)

V - 2.1(2001)

V - 2.3 (2003)...

V - 2.6 (2008)

V - 2.7 (2010)

Version 3 (2008)

V - 3.1 (2009)...

V - 3.7 (2018)

V - 3.5 (2015)

V - 3.8 (2019)

V - 3.6 (2016)

V - 3.8.2 - (2020)

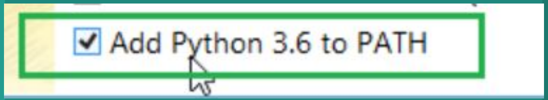
Python Installation - (Windows)

Python Installer

Download the Python installer from the ***Download Python***

Add Python to PATH

Please select the checkbox in the installer option.



☑ Add Python 3.6 to PATH

Check Python in system

1. Goto to cmd and type >python -version
2. Check python shell & python IDLE

Pro Tip

Always download a version previous to the latest one. (More Stable Version)

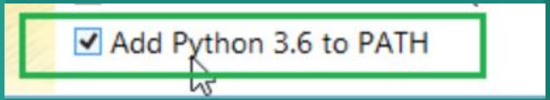
Python Installation - (Mac & Linux)

Python Installer

Download the Python installer from the ***Download Python***

Add Python to PATH

Please select the checkbox in the installer option.



☑ Add Python 3.6 to PATH

Check Python in system

1. Goto to terminal &type $python3 -version
2. Check python shell & python IDLE

Pro Tip

Always download a version previous to the latest one. (More Stable Version)

Note: Python 2 version will be in your system by default, still need to install python 3

What is a program?

Set of Instructions to perform a task

Example:
Perform addition, subtraction, etc.
Or
Print any message

Here task is to perform addition or subtraction.

Write lines of code to give particular instructions to system to carry out the task

Sample Program

Addition

Example instructions

- ***Collect data from user***
- ***Perform addition***
- ***Send back the result to user (output)***

```
A = 1
B = 2
Result= A + B
Send back Result
```

How Python works?

Python is an interpreted and compiled(behind the scene) language

Input

Python code
“program.py”
print(‘hello world’)

Cpython (Compiler)

Convert the code into bytecodes.
File: program.pyc
(Intermediate code)

Output

Machine code instruction executed by processor and give result

PVM uses Interpreter

Python Virtual Machine
...process the code
Binary/Machine Code

Python First Program - “Hello World”

```
print(‘Hello World’)
# using print method passing string in single quotes(‘)
print(“Hello World”)
# using print method passing string in double quotes(“)
print (“hello\n world”)
# using print method with nextline string annotation(\n)
print(“hello\world”)
```

Python Comments- Single line

```
# Working with Comments in python language -

# we use comments to write some self notes or message for team...

# it does not get read by the python interpreter

# Print (“Hello World”)

print(“I am learning Python”)

print(‘hello world’)

print('Hello World!!!') # Yes you can comment one end of the statement as well

print('Hello World') # rest part is comment
```

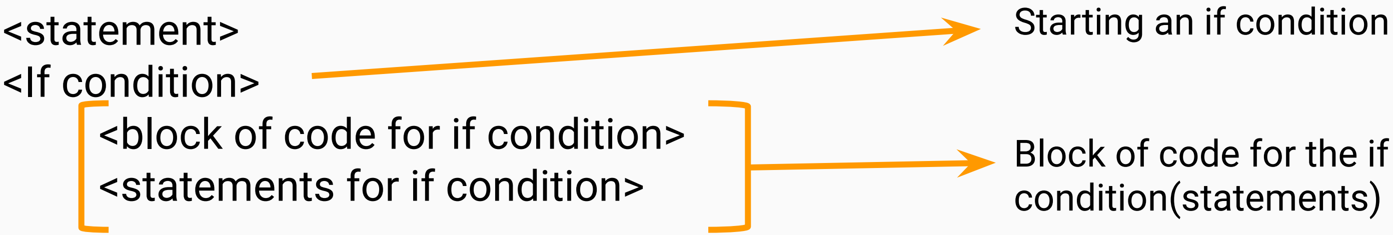
Python Comments - Multiline

```
# Multi-line Comment - delimiter (""") on each starting and end of the comment.

"""

This would be a multiline comment

in Python that spans several lines and

describes your code, your day, or anything you want it to

""""
```

Indentation in Python

In Python, indentation is used to separate the block of code from one another.

Demo example #1:

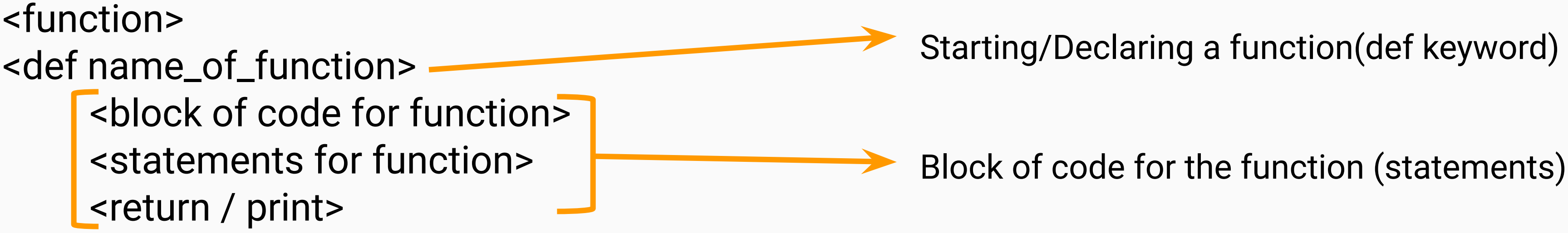


```
<statement>
<If condition>
    <block of code for if condition>
    <statements for if condition>
```

Starting an if condition

Block of code for the if condition(statements)

Demo example #2:

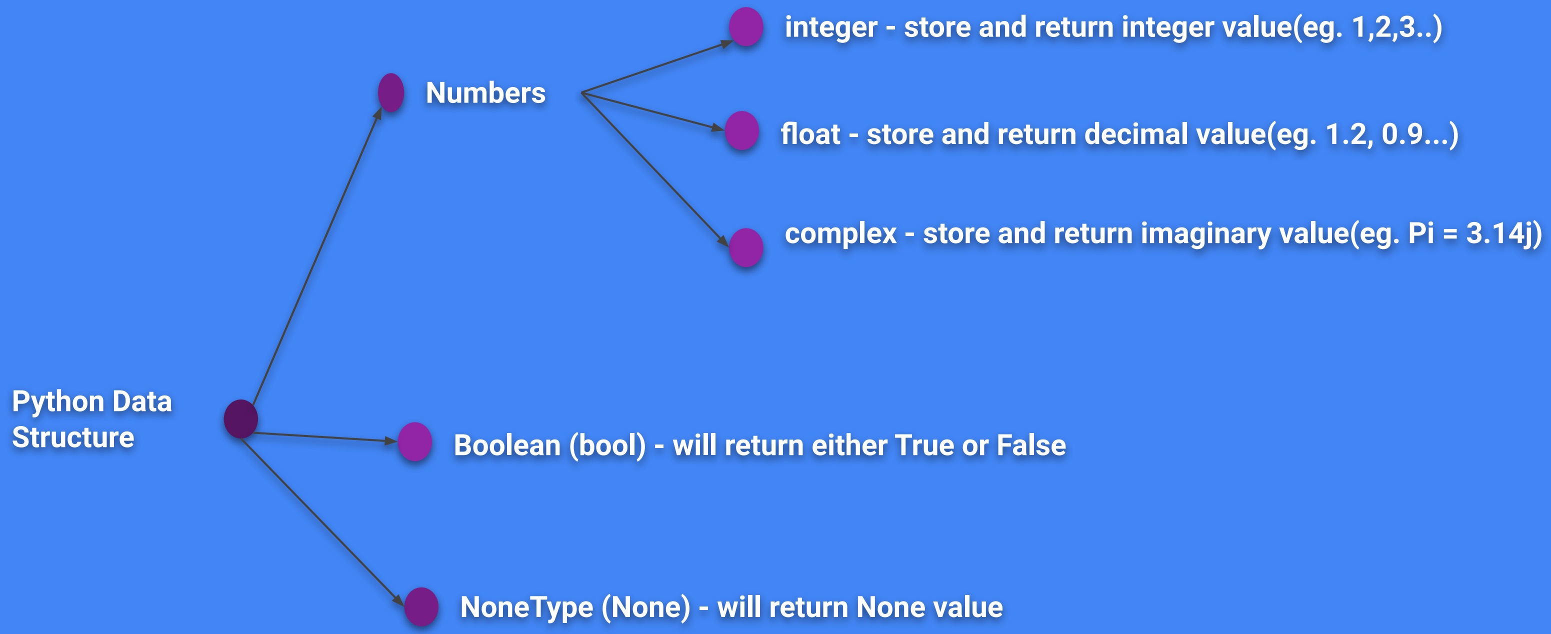


```
<function>
<def name_of_function>
    <block of code for function>
    <statements for function>
    <return / print>
```

Starting/Declaring a function(def keyword)

Block of code for the function (statements)

Python Data Structure - Basic



Python Data Structure

Numbers

integer - store and return integer value(eg. 1,2,3..)

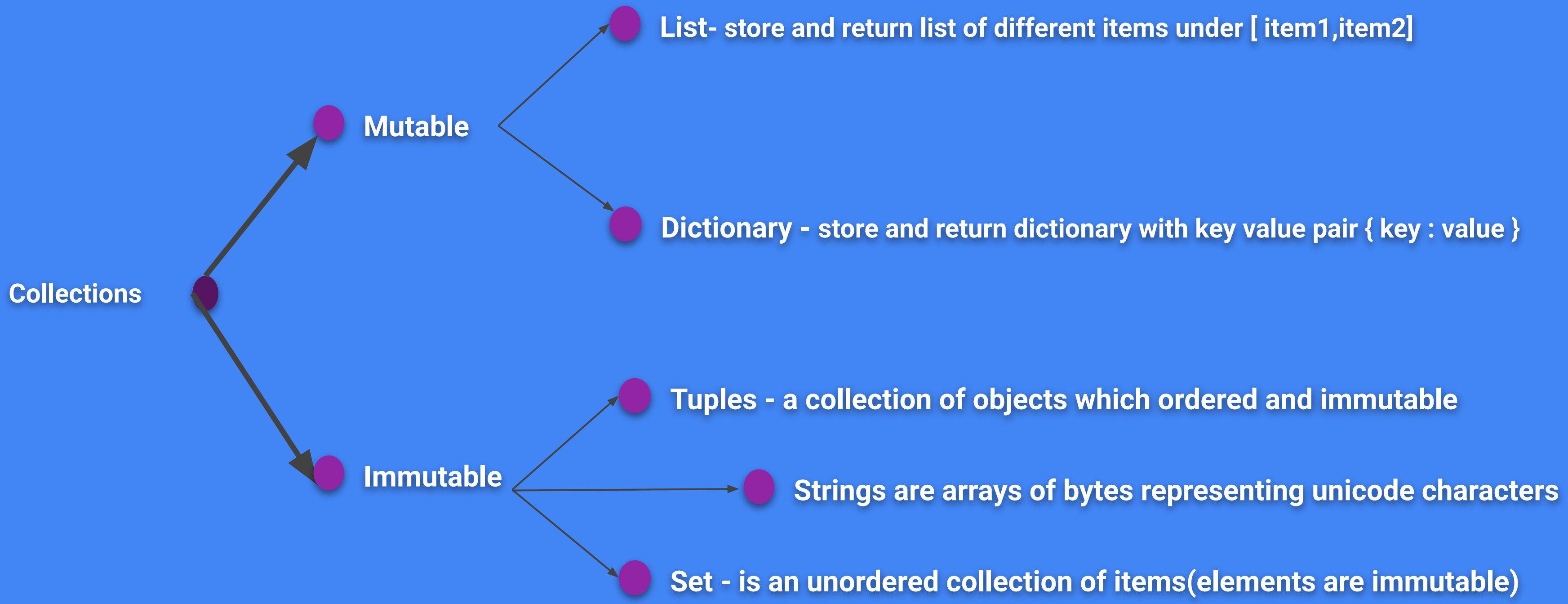
float - store and return decimal value(eg. 1.2, 0.9...)

complex - store and return imaginary value(eg. Pi = 3.14j)

Boolean (bool) - will return either True or False

NoneType (None) - will return None value

Python Data Structure - Advanced



Collections

Mutable

List- store and return list of different items under [item1,item2]

Dictionary - store and return dictionary with key value pair { key : value }

Immutable

Tuples - a collection of objects which ordered and immutable

Strings are arrays of bytes representing unicode characters

Set - is an unordered collection of items(elements are immutable)

Everything is an Object in Python

Scalar Object

<class ‘int’> represents integer type in python

<class ‘float’> represents float type in python

<class ‘bool’> represents boolean type in python

Non - Scalar Object

<class ‘str’> represents string type in python

<class ‘list’> represents list type in python

<class ‘tuple’> represents tuple type in python

<class ‘dict’> represents dictionary type in python

Python Variables

Creating Variables: Variables are containers for storing data values in program.

Python has no code syntax for declaring a variable.

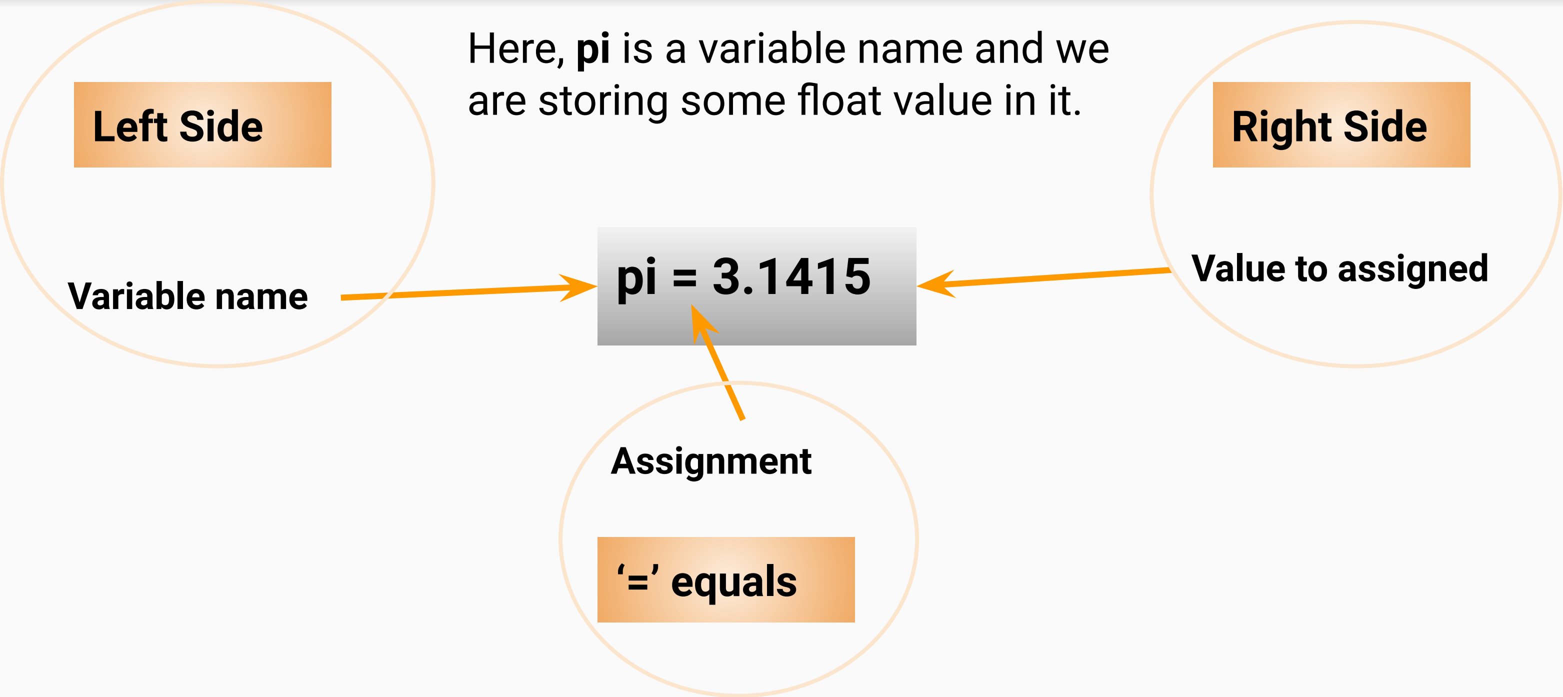
Variable is created the moment you first assign a value to it.

Python - Variables

```
# Types of Data in python
int_id = 1001
float_price = 98.98
string_name = 'String'
bool_counter = True

print(int_id)
print(float_price)
print(string_name)
print(bool_counter)

print('Type of data variable - int_id:’, type(int_id))
print('Type of data variable - float_price:', type(float_price))
print('Type of data variable - string_name:', type(string_name))
print('Type of data variable - bool_counter:’, type(bool_counter))
```

Storing Values in Variable

Here, **pi** is a variable name and we are storing some float value in it.



Left Side

Variable name

pi = 3.1415

Assignment

‘=’ equals

Right Side

Value to assigned

Giving name to Expressions(some math formula)

Abstrating math equation

```
pi = 3.1415
radius_cm = 20
area_circle = (pi)*(radius_cm ** 2)
print(area_circle)
```

$$\text{Area} = \pi r^{2}$$

Express(variable with formula)

Changing Binding - restoring different in same variable

```
pi = 3.1415
radius_cm = 20
area_circle = (pi)*(radius_cm ** 2)
print(area_circle)
radius_cm = 25
area_circle = (pi)*(radius_cm ** 2)
print(area_circle)
```

Changing Radius (using same variable)

Checking Types of Variables - type()

```
# Types of Data in python

int_id = 1001
float_price = 98.98
string_name = 'String'
bool_counter = True
print('--------------------name of variable------ ','  -----Type------','\t\t\t--Actual Value--')
print('Type of data variable - int_id:     \t\t',type(int_id),'\t\t\tValue = ',int_id)
print('Type of data variable - float_price: \t\t',type(float_price),'\t\tValue = ',float_price)
print('Type of data variable - string_name: \t\t',type(string_name),'\t\t\tValue =
',string_name)
print('Type of data variable - bool_counter: \t\t',type(bool_counter),'\t\tValue =
',bool_counter)
```

Type Casting

x=”101”

To convert to integer- int(x) #output- 101

y=101

To convert to string- str(y) #output- “101”

z=”101.32”

To convert to float/decimal- float(z) #output- 101.32

int type in Python 3

Python has a integer type class which is used to store integer type values

```
x = 1
print(type(x))
# output: <class 'int'>
```

str (String) type in Python 3

Python has a String type class which is used to store string type values

```
message = 'I am good'
print(type(x))
#output: <class 'str'>
```

Type Casting - integer to float and vice versa

```
# Converting integer to float and vice versa

int_num = 1

print('Going to convert this identifier\' type\n from INT to >> FLOAT')

print('Type Before: ',type(int_num),'\nType After: ',type(float(int_num)))

#and vice versa

float_num = 1.0

print('Going to convert this identifier\' type\n from Float to >> Int')
print('Type Before: ',type(float_num),'\nType After: ',type(int(float_num)))
```

Type Casting - Int to string and vice versa

```
# Int to string

int_num = 1

print('Going to convert this identifier\' type\n from INT to >> String')

print('Type Before: ',type(int_num),'\nType After: ',type(str(int_num)))

#  and vice versa

string_num = '1'

print('Going to convert this identifier\' type\n from String to >> Int')

print('Type Before: ',type(string_num),'\nType After: ',type(int(string_num)))
```

Type Conversion in Python - part 1

```
# implicit Type Casting

num_int = 123
num_flo = 1.23
num_new = num_int + num_flo
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

Type Conversion in Python - part 2

```
# Let’s try to add number with a string

num_int = 123
num_str = "456"
print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))
print(num_int+num_str)

# it will throw an error --to correct it, we use Explicit Type Conversion
```

Type Conversion in Python - Explicit Type Conversion

```
num_int = 123
num_str = "456"

print("Data type of num_int:",type(num_int))
print("Data type of num_str:",type(num_str))

# type casting ‘num_str’ from string to integer (performing addition)
print(num_int  +  int(num_str))
print("and new type of generated number is",type(num_int  +  int(num_str)))

    # output:
    # Data type of num_int: <class 'int'>
    # Data type of num_str: <class 'str'>
    # 579
    # and new type of generated number is <class 'int'>
```

Naming Variables in Python

Do’s	Don'ts
Can be of any length.	Cannot start with a number, no spacing within name
Combination of(a-z)(A-Z)(0-9)	Keywords cannot be used as identifiers
Can use _ (underscore) in between	special symbols like !, @, #, $, % etc
name_01,Name_01,a,b,c,d Var001, var001, long_name really_long_name_identifier	@name, &new, 01number, 0string,0,1,2,3 True,False, try,def,from,while,if,in,global 01_not_valid_identifier, with space

Keywords & Identifiers in Python

There are 35 keywords in Python 3

False,await,else,import,pass,None,break,except,in,raise,True,class,finally,is,return,and,continue,for,lambda,try,as,def,from,nonlocal,while,assert,del,global,

Not,with,async,elif,if,or,yield

Identifiers:(How to name variables in python)

Print() Function

Print in Python - Part 1

```
#some different print example
# simple printing
print('hello')

# adding \n for printing value in next line
print('nextLine printing \nthis will be on next line')

# print function with tab - TAB option (using tab space)
print('Printing with space \tThis text is using tab space')

#let's use + in print --
print('string1'+'string2')
print('string '*3) # print 'string' 3X times
```

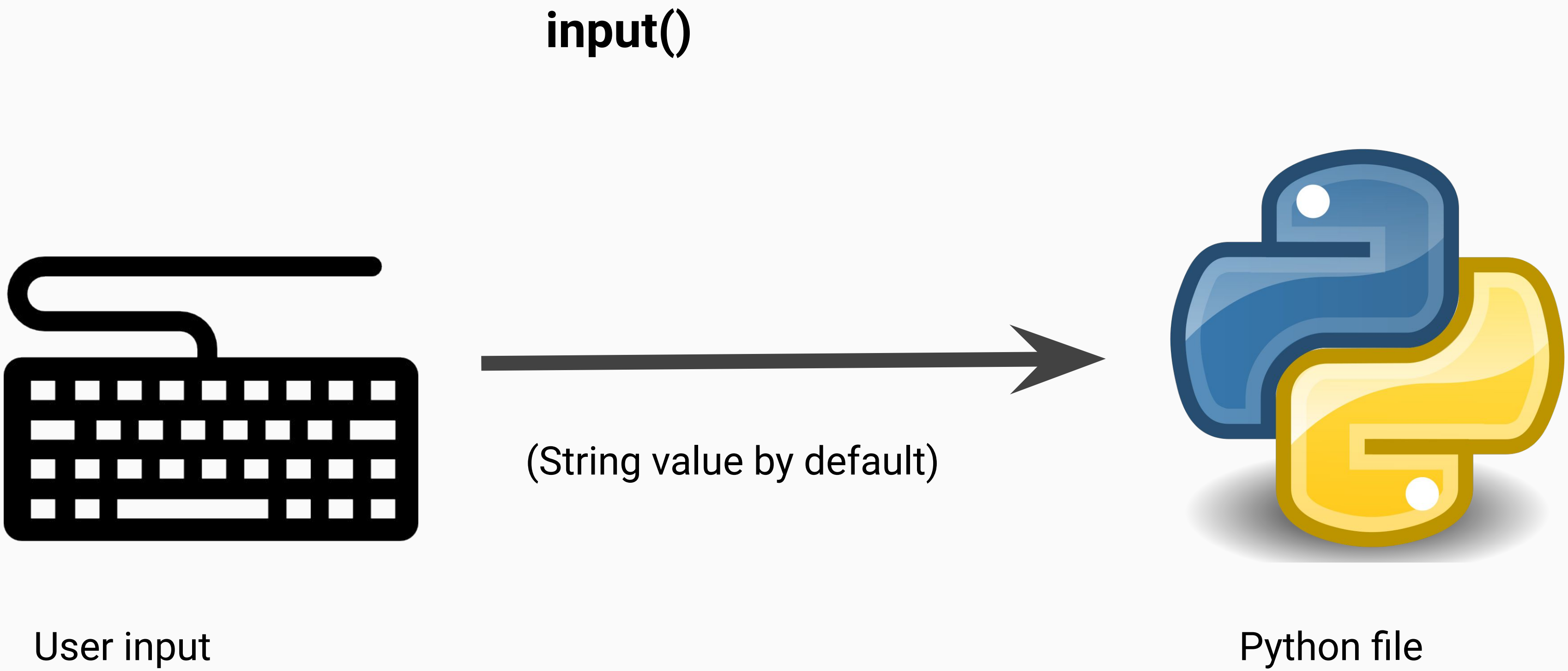
Print in Python - Part 2

```
#more example
a,b = 1,2
print(a,b)
print(a,b,sep=',') # using separator attribute
print(192,168,1,0,sep=':') # we can use it to show ip address
print('hello',end='') # this will end the method by removing the default \n nextline to none
print('hello',end='\t')# this will end the method by removing the default \n nextline to \t tab space
print('=ending with tab space')
```

More on print - using “” & ‘’ (single/double quotes)

```
# Python Accepts double quotes(" ") and single quotes(' ') in same manner
# some basic example
print('A beautiful day !!!','--using single quotes')
print("A beautiful day !!!","--using double quotes")
# plus we can use both in one statement
# but, in different string of-course
print("A beautiful day !!!",'mixed--using single quotes in latter')
print('A beautiful day !!!',"mixed--using double quotes in latter")
# how to print It's a beautiful day - single quote in the message itself
# we simply use the double quotes - RIGHT!
print("It's a beautiful day",": - USING SINGLE quote INSIDE DOUBLE quotes")
# then try to use DOUBLE quotes inside SINGLE quotes
# we simply use the single quotes - RIGHT!
print('A "beautiful" day',': - USING DOUBLE quote INSIDE SINGLE quotes')
# how can we use both in one message - to print: It's a "beautiful" day
# TRICK #1 - USE FORWARD SLASH '\' >> It\'s
print('It\'s "beautiful" day',': - USING BOTH quotes INSIDE SINGLE quotes')
# TRICK #2 >>  \"beautiful\"
print("It's \"beautiful\" day",': - USING BOTH quotes INSIDE DOUBLE quotes')
```

Input from User in Python



input()

(String value by default)

User input

Python file

Taking input in Python

input() function executes program flow will be stopped until an input from user.

The string or message shows on the screen to ask user to enter value is optional i.e. the prompt, will be showed on the screen is optional.

When enter as input, the input function convert it into a string. if you enter an integer value still input() function change it into a string. You need to individually convert it into an integer in your code using typecasting.

Example: Taking input in Python

```
# Program to check input

num_input = input ("Enter number :")
print(num_input)
string_input = input("Enter name : ")
print(string_input)

# Printing type of input value with Values

print (f"type of number {type(num_input)}, and value is {num_input}")
print (f"type of string {type(string_input)}, and value is {string_input}")
```

Input in Python - with int and float type-casting

```
# simple input method in python
default_type_variable = input("Input any type of data\n")
# default type is string data
print("type of the data entered: ",type(default_type_variable)," Value = ",
default_type_variable)
# defining type of data to be entered input
# Integer Data - type casting
my_int_data = int(input("enter integer type data\n"))
print("type of the data entered: ",type(my_int_data)," Value = ", my_int_data)

# Float Data - type casting
my_float_data = float(input("enter decimal/float type data\n"))
print("type of the data entered: ",type(my_float_data)," Value = ", my_float_data)
```

Operators

Operators are for manipulating the value of operands.

Example:
4 + 5 = 9.
Here, 4 and 5 = operands
+ = operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Operators

Operators are special symbols in Python.

It simply carry out arithmetic or logical computation for us.

“Operand” is value that the operator operates on.
2 + 3 = 5

Here, “+” is the “operator” which performs addition.
2 and 3 are the “operands” and 5 is the output of the operation.

Example: Python Arithmetic Operators

```
x = 15
y = 4
# Output: x + y = 19
print('x + y =',x+y)
# Output: x - y = 11
print('x - y =',x-y)
# Output: x * y = 60
print('x * y =',x*y)
# Output: x / y = 3.75
print('x / y =',x/y)
# Output: x // y = 3
print('x // y =',x//y)
# Output: x ** y = 50625
print('x ** y =',x**y)
```

Comparison Operators

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

```
x = 10
y = 12
# Output: x > y is False
print('x > y is',x>y)
# Output: x < y is True
print('x < y is',x<y)
# Output: x == y is False
print('x == y is',x==y)
# Output: x != y is True
print('x != y is',x!=y)
# Output: x >= y is False
print('x >= y is',x>=y)
# Output: x <= y is True
print('x <= y is',x<=y)
```

Logical Operators - and, or, not

Logical Operator - ‘and’

T/F	and	F/T	=
True	and	True	True
True	and	False	False
False	and	True	False
False	and	False	False

Logical Operator - ‘or’

T/F	or	F/T	=
True	or	True	True
True	or	False	True
False	or	True	True
False	or	False	False

Logical Operator - ‘not’

T/F	not	=
True	not	True
False	not	False

Example #1 : Logical Operators

Logical operators are the and, or, not operators.

```
x = True

y = False

print('x and y is',x and y)

print('x or y is',x or y)

print('not x is',not x)
```

Example #2 : Logical Operators

```
# Logical Operation
print('Using Logical operator - "and"\n--')
print(f'True and False:= {True and False}')
print(f'True and True:= {True and True}')
print(f'False and True:= {False and True}')
print(f'False and False:= {False and False}')

print('Using Logical operator - "or"\n--')

print(f'True or False:= {True or False}')
print(f'True or True:= {True or True}')
print(f'False or True:= {False or True}')
print(f'False or False:= {False or False}')
```

Bitwise Operators

Bitwise operators treat the operands strings of binary digits.

```
x = 70          # 70 = 1000110
y = 23          # 23 = 0010111
z = 0
z = x & y       # Binary AND
print (f"Binary AND, z = { z }")
z = x | y      # Binary OR
print (f"Binary OR, z = { z }")
z = x ^ y       # Binary XOR
print (f"Binary XOR, z = { z }")
z = ~x         # Binary Ones Complement
print (f"inary Ones Complement, z = { z }")
z = x << 2      # Binary Left Shift
print (f"Binary Left Shift, z = { z }")
z = x >> 2      # Binary Right Shift
print (f"Binary Right Shift, z = { z }")
```

Assignment Operators

Assignment operators(mathematical operator)in Python.

```
x ,y ,z  = 26, 18, 0
z = x + b
print('z = x + y assigns value of x + y into z ')
print (f"Value = z is { z }" )
z += a
print('z += x is equivalent to z = z + x ')
print (f"Value = z is { z }" )
z *= a
print('z *= x is equivalent to z = z * a')
print (f"Value = z is { z }" )
z /= x
print('z /= x is equivalent to z = z / a')
print (f"Value = z is { z }" )
z  = 2
z %= a
print('z %= x is equivalent to z = z % x ')
print (f"Value = z is { z }" )
```

```
z **= a
print('z **= x is equivalent to z = z ** x ')
print (f"Value = z is { z }" )
z //= a
print('z //= x is equivalent to z = z // x ')
print (f"Value = z is { z }" )
```

= VS == (assign vs equal)

The = is a simple assignment operator.
It assigns values from right to the left.

Assignment (=)

```
x = 5
a= “python”
print(x)
Output 5
```

(***Note- the value must be on the right side of the equal sign, 4= x would throw an error)***

== is for checking the values of two operands are equal or not.
If yes, then **True**, If no, then **False**

Equal (==)

```
x= 5
a= 5
x==a
print(x==a)
#output- true
```

Membership Operator

Membership operators in Python.
(in) and (not in) Use to check the presence of element in the python sequence (eg. string, list, tuple, set and dictionary)

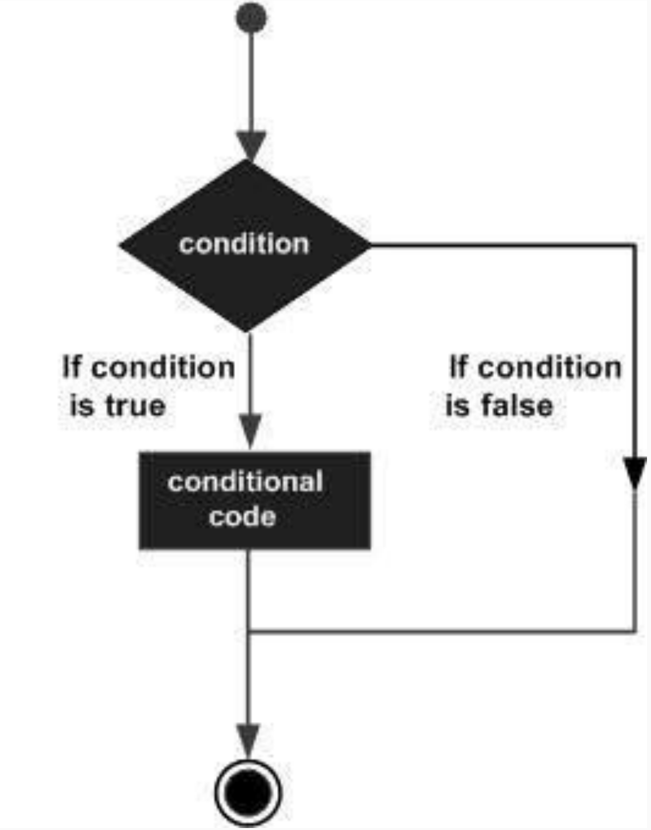
```
x = 'Hello world'
y = {1:'a',2:'b'}

# Output: True
print('H' in x)

# Output: True
print('hello' not in x)

# Output: True
print(1 in y)

# Output: False
print('a' in y)
```

Control Flow

Control flow in programming language is the way of making decision for the system execution of a certain set of instructions, taking feasible conditions in account.

Feasible conditions based to the fact of the concept/behaviour of the program. Either it return a **TRUE** or a **FALSE**.

For example, pick all the odd numbers from the given list...if number divided by 2 gives remainder 1, then according to flow, the number is odd.



Types of Control Flow

Sr.No.	Statement & Description
1	**if statements** Using an if… statement will return either True or False.(program block will execute only if **True**)
2	**if...else statements** Using an **if...else** statement in program gives us 2 part execution of 2 programming blocks. (first for TRUE and second for FALSE)
3	**if - elif - else** Now, we can put (n) number of coding block in our program using ELIF as many times we want. (**elif** in python means **else if)** Example: **if (1) >>print('one'), elif (2) >>print('two'), elif (3) >>print('three'), … , and so on.**

CONTROL FLOW - Branching

```
if<condition>:
         <expression>
         <expression>
         • • •
```

```
if<condition>:
         <expression>
         <expression>
         • • •
else: <expression>
         <expression>
         • • •
```

```
if<condition>:
         <expression>
         <expression>
         • • •
elif <condition>:
         <expression>
         <expression>
         • • •
else: <expression>
         <expression>
         • • •
```

- <condition> has a value True or False
- evaluate expressions in that block if <condition> is True

If statement

In this example, we use two variables, a and b, using **if** statement here we can find which number is greater or smaller than other.

```
a = 33

b = 200

if b > a:

  print("b is greater than a")
```

If ..else statement

Extending the previous example, here we are checking both ways, using the **else** statement as well.

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
else:
 print(“a is greater than b”)
```

Example #1 : If-else

```
salary=int(input("Enter the Salary : "))

if salary >= 50000:

  print("20% Bonus")

else:

  print("10% Bonus")
```

Here, the salary value is being checked, and

(**condition if**)

if it is greater than or equal to 50k, you’ll get bonus of 20%.

(**condition else**)

or **else**, you’ll get bonus of 10%.

Elif Statements

The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

```
a = 33

b = 33

if b > a:

  print("b is greater than a")

elif a == b:

  print("a and b are equal")
```

In Python

elif = else + if

elif ..else Statement

The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Example #2 : If-elif-else

```
marks=int(input("Enter the Marks : "))

if marks >= 90:

  print("Grade A")

elif marks >= 80:

  print("Grade B")

elif marks >= 70:

  print("Grade C")

elif marks >= 60:

  print("Grade D")

else:

  print("Grade F")
```

Example #3 : Simple Calculator

```
# building a simple calculator
num_1 = float(input('Enter First number\n'))
num_2 = float(input('Enter Second number\n'))
# taking float type for performing arithmetic operations
math_operator = input(
    'please enter symbol\nFor Addition => enter +\nFor Subtraction => enter -\nFor Multiplication =>
enter *\nFor Division => enter /\n\n\n')

if math_operator == '+':
         print('Addition Result:\nnum_1\tnum_2\n', num_1,
          math_operator, num_2, '\n---------------\nAnswer\t', num_1 + num_2)
elif math_operator == '-':
          print('Subtraction Result:\nnum_1\tnum_2\n', num_1,
          math_operator, num_2, '\n---------------\nAnswer\t', num_1 - num_2)
elif math_operator == '*':
         print('Multiplication Result:\nnum_1\tnum_2\n', num_1,
          math_operator, num_2, '\n---------------\nAnswer\t', num_1 * num_2)
elif math_operator == '/':
         print('Division Result:\nnum_1\tnum_2\n', num_1,
          math_operator, num_2, '\n---------------\nAnswer\t', num_1 / num_2)
else:
    print('wrong entry')
```

Indentation

Tab Spaces are used for indentation in Python. When we use methods or any statement(looping or conditional), tab-space or indentation is provided by default

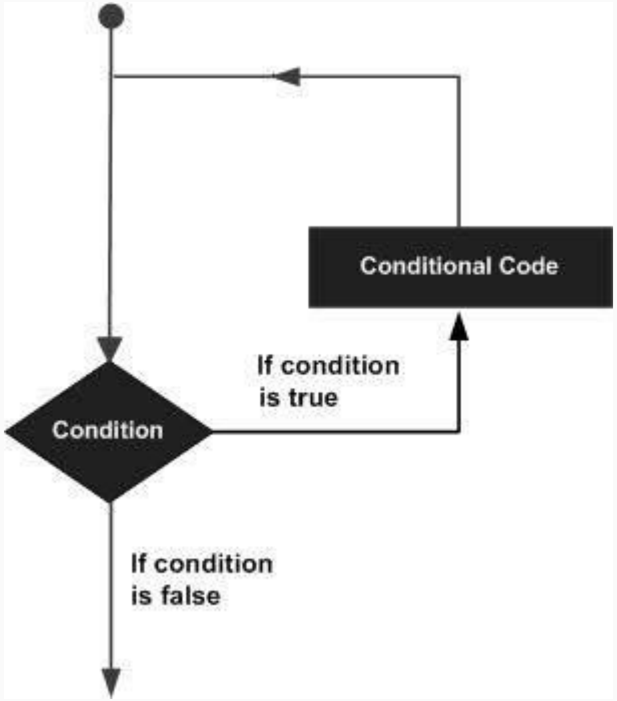
```
# building a simple calculator
num_1 = float(input('Enter First number\n'))
num_2 = float(input('Enter Second number\n'))
# taking float type for performing arithmetic operations
math_operator = input(
    'please enter symbol\nFor Addition => enter +\nFor Subtraction => enter -\nFor Multiplication
=> enter *\nFor Division => enter /\n\n\n')

if math_operator == '+':
        print('Addition Result:\nnum_1\tnum_2\n', num_1,
         math_operator, num_2, '\n---------------\nAnswer\t', num_1 + num_2)
elif math_operator == '-':
        print('Subtraction Result:\nnum_1\tnum_2\n', num_1,
         math_operator, num_2, '\n---------------\nAnswer\t', num_1 - num_2)
elif math_operator == '*':
        print('Multiplication Result:\nnum_1\tnum_2\n', num_1,
         math_operator, num_2, '\n---------------\nAnswer\t', num_1 * num_2)
elif math_operator == '/':
        print('Division Result:\nnum_1\tnum_2\n', num_1,
         math_operator, num_2, '\n---------------\nAnswer\t', num_1 / num_2)
else:
    print('wrong entry')
```

Looping Statements in Python

Looping is programming means repeating a certain task for certain occurrence

However, a looping statement consist three important parts.

1. Start Condition
2. Stop Condition
3. Increment/Decrement



Control Flow: while LOOPS

```
While <Condition>:

    <expression>

    <expression>
```

- <condition> evaluates to a boolean
- if<condition> is True, do all the steps inside the white code block
- Check<condition>again
- Repeat until <condition> is False

How to print name 500 times ?

```
#while loop in Python

counter = 0
while counter<=400:
        print(“my name is Arnav”)
        counter = counter + 1
```

Python For Loops

For loops can be used for iterating(looping) over a sequence.

A sequence can be(list, tuple, dictionary, set and, string).

```
#Python program to print name of fruits in list using for loop
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
#output- apple, banana, cherry
```

Example: Python For Loops

For loops can be used for iterating(looping) over a sequence.

A sequence can be(list, tuple, dictionary, set and, string).

Using the range() function in for loop

```
# for loop in python using range()

for counter in range(15):
    print(“my name is Arnav”)
```

Range (start,stop,step)

Range() is used parameter/argument values for starting and stopping point.

Step is used to give increment/decrement to the starting value each time, till it reaches the stopping value.

```
mysum =0
for i in range (1,5,2)
        mysum+=i
print(mysum)

#Output- 4
```

Start- Starting Value(Default 0)

Stop- Stopping Value

Step- increment/decrement.
(Default 1 increment)

Example: “For” Loops with “range()” Function

For loops can be used for iterating(looping) over a sequence.

range(5) means elements [0,1,2,3,4]

```
for x in range(5):
    print(x)
# Prints out the numbers 0,1,2,3,4

#range(initial, final, step)

for x in range(3, 6):
    print(x)
# Prints out 3,4,5

for x in range(3, 8, 2):
    print(x)
# Prints out 3,5,7
```

Infinite Loops

[Caution: Press CTRL + C, or close the terminal/program to exit loop]

```
# Having fun the infinite looping using while loop
x= 1
while(x==1):
    print('python says Hello!!!’) Press Ctrl + C to exit infinite loop')
```

Nested Loops

Loops inside Loop - multiplication tables(1-10 x 10)
# nested Loop

```
for i in range(1,11):
    for j in range(1,11):
        mult_table = i*j
        print(f'{mult_table}',end=' ')
    print(f'\tTable of {i} x 10:')
```

```
<for loop condition>
        ...
        ...statements
        <for loop condition>
                ...statements
                ...
```

(Nested For Loop)

Looping and Conditional statement - Print all even/odd numbers between 1-100

```
# Print all even/odd numbers between 1-100 - using 'FOR' loop
for i in range(1,101): # looping from 1-100 numbers

# print(i)
    chk = i % 2 # ‘chk’ variable stores the mod value (i modular division 2)
    if chk == 0: # checking condition for even number
        print(i,"is an even number")
    elif chk == 1: # checking condition for odd number
        print(i,'is an odd number')
```

for VS while LOOPS

For Loops

- Know number of iterations
- Can end early via break
- Uses a counter
- Can rewrite a for loop using a while loop

While Loops

- Unbounded number of iterations
- Can end early via break
- Can use a counter but must initialize before loop and increment it inside loop
- May not be able to rewrite a while loop using a for loop

Break Statement

Break STATEMENT

- Immediately exits whatever loops it is in
- Skips remaining expression in code block
- Exists only innermost loop!

```
While <condition_1>:
    While <condition_2>:
        <expression_a>
        Break
        <expression_b>
        <expression_c>
```

Example#1: Continue and Break

```
# Continue statement will ignore the in the loop
print(f'This will skip number \'8\' from the series
0-9')
for count in range(10):
    if count == 8:# when 8 comes in loop
        continue
    print(count,end=':')
```

```
# Break statement will ignore the in the loop
print(f'This will skip all number after \'7\'
from the series 0-9')
for count in range(10):
    if count == 7: # when 7 comes in loop
        break
    print(count,end=':')
```

Example#2: Continue and Break

continue

```
for x in “door”:
   if x == ‘o’:
       continue
   print(“the current letter is”, x)
# output; the current letter is d
          The current letter is r
```

break

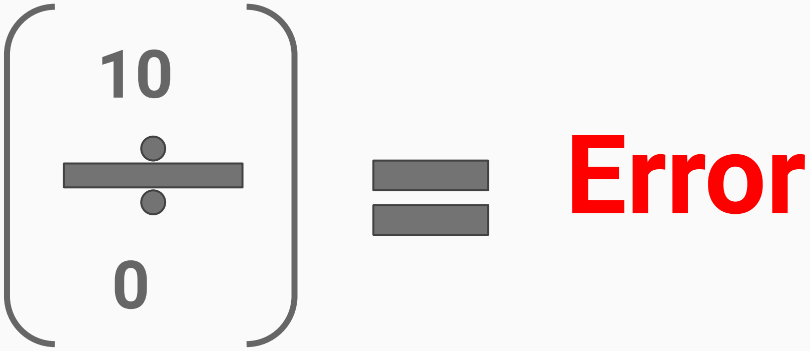
```
for x in “door”:
   if x == ‘o’:
       break
   print(“the current letter is”, x)
#output;  the current letter is d
```

pass in Python

Pass in python means to **do nothing - (used to come out of block of code/statement)**

Example: Division by zero

```
<if divided by zero>
        <custom error message>
        pass
```



Will prevent the error in the flow of program

pass in Python

```
#Example#1 of pass statement for function

def do_nothing_function():
        pass

#Example#2 Pass

# pass statement in python
print('Pass Statement in python, using which leads to no operation to the program statement')
print('example of division function, will use pass when divisor = 0\n')
def divide_two_number(dividend,divisor):
    if divisor == 0:
        print('divisor cannot be zero - program will end here')
        pass
    else:
        print(f'Result of division: {dividend/divisor}')

# calling the function - passing divisor = 0
divide_two_number(10,0)
```