Python

```
# This is comment
print("Hello World")

"""
Multiple
Line
Comment
"""
```

Variables

String Integer Float

type() to get the datatype.

Casting:

str() int() float()

Naming Convention

Camel Case — myVariableName

Pascal Case — MyVariableName

Snake Case — my_variable_name

Assigning Multiple Values

```
x, y, z = "Orange", "Mango", "Cherry"

x = y = z = "Orange"

fruits = ["Orange", "Mango", "Chery"]
x, y, z = fruits
```

Print

```
print(x)
print(x, y, z)
```

print (x + y + z)

↳ Result may vary based on datatypes.

Global Variable

To declare a global variable inside a function, use global

Ex:
```
def fun():
    global var1
```

Data Types

Text	:	str
Numeric	:	int , float , complex
Sequence	:	list , tuple , range
Mapping	:	dict
Set	:	set , frozenset
Boolean	:	bool
Binary	:	bytes , bytearray , memoryview
None	:	NoneType

Strings

'hello' / "hello"

→ Can use a quotes inside a string as long as they are not matching

Multi line Strings

```
'''
Hello
World
'''
```

/

```
"""
Hello
World
"""
```

Strings are Arrays

→ Python don't have char datatype

→ A single character is a string with length 1.

Ex:-

```
a = "Hello"
print (a[1])  # Returns e
```

Ex: for x in "banana":
    print (x)

len() returns the length of the string

Ex: a = "Hello"
print (len (a)) # Returns 5

Use in to check for a phrase in string

Ex: print ("H" in "Hello) # Returns True

Vise Versa ⇒ not in

String slicing

string [start : end]

Ex: a = "Hello"
print (a [1:3]) # el
print (a [1:]) # ello
print (a [:2]) # He

```
print (a[:])        # Hello
print (a[-3:-1])    # ll
```

↳ Negative Slicing

Built-in String methods:

upper()

lower()

strip()

→ Removes whitespaces from the beginning and end of a string.

replace(str, str)

→ Replaces phrase with other phrase

```
Ex: a = "Hello"
    print (a.replace ("H", "C"))  # "Cello"
```

split (str)

→ Makes substrings based on the seperator

```
Ex: a = "Hello World"
    print (a.split (" "))
    # Returns ['Hello', 'World']
```

startswith (str)

join (obj)

→ Joins the elements in obj into a single string

Ex: list1 = ["Hello", "World"]

```
x = " ".join(list1)
print(x)        # "Hello World"
```

String Concatenation

Ex:

```
a = "Hello"
b = "World"
print(a + " " + b)   # Hello World
```

F-strings

→ Use {} as placeholders

Ex:

```
age = 26
txt = f"I'm {age} years old"
```

Ex:- price = 360

txt = f" Price is {price:.2f}

Escape Character

Back slash "\"

Ex:- " We are \" Vikings \" "

Booleans

True / False

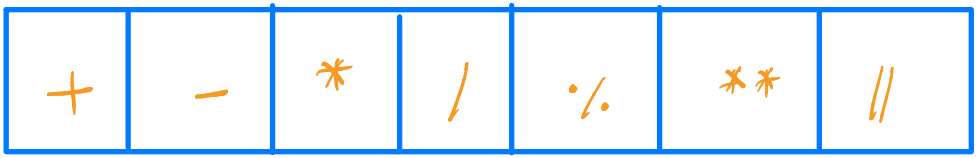
→ bool( ) is used to evaluate any valu

→ Any string is True, except empty string

→ Any number is True, except 0

→ Any list, set, tuple and dict are True, except empty ones.

Operators



Division

Modulus

Exponentiation

Floor division

Assignment Operators

=

+=

-=

*=

/=

%=

//=

**=

&=

|=

^=

>>=

<<=

:=

Comparison Operators

==	!=	>	<	>=	<=

Logical Operators

and or not

Identity Operator

is is not

Membership Operator

in not in

Bitwise Operators

&	|	^	~	<<	>>
AND	OR	XOR	NOT	left shift	Right shift

Operator Precedence

Operator	Description
`()`	Parentheses
`**`	Exponentiation
`+x` `-x` `~x`	Unary plus, unary minus, and bitwise NOT
`*` `/` `//` `%`	Multiplication, division, floor division, and modulus
`+` `-`	Addition and subtraction
`<<` `>>`	Bitwise left and right shifts
`&`	Bitwise AND
`^`	Bitwise XOR
`|`	Bitwise OR
`==` `!=` `>` `>=` `<` `<=` `is` `is not` `in` `not in`	Comparisons, identity, and membership operators
`not`	Logical NOT
`and`	AND
`or`	OR

→ If 2 operators have same precedence, then expression is evaluated from left to right.

List

→ Used to store multiple items in a single variable

```
my_list = ["Apple", "Orange"]
```

→ List items are Ordered
Changable
Allows duplicate Values

len() returns the length of the list.

→ list can contain different data types.

type() returns <class 'list'>

Access list items

Ex: `thislist[1]`

`thislist[-1]`

`thislist[2:5]`

Check if Item Exist:

Ex:
```
fruits = ["apple", "banana", "orange"]
print("apple" in fruits)
```

Change item value in list:

```
thislist[1] = "apple"
thislist[2:5] = ["apple", 5, True]
thislist.insert(2, "apple")
thislist.append("apple")
thislist.extend(["apple", "banana", False])
```

→ extend also works with other iterable objects (Tuples, sets, dictionary, etc)

Remove Elements from list:

```
fruits.remove("banana")
```

→ If there are multiple occurrences of "banana", then first occurrence is removed.

```
fruits.pop(1)  # Item at index 1 is removed
```

→ If no index is mentioned, then last element is removed.

```
del fruits[0]  # Removes first element
del fruits     # Removes entire list
fruits.clear() # empties the list
```

Loop through a list:

Ex:
```
for x in fruits:
    print(x)
```

Ex:
```
for i in range(len(fruits)):
    print(fruits[i])
```

List Comprehension:

```
[print(x) for x in fruits]
```

```
newlist = [ expression for item in iterable
            if condition == True ]
```

Ex:
```
newlist = [ x if x != "banana" else "orange"
            for x in fruits]

# Replaces all banana with orange.
```

Sorting a list:

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

→ Sorting is case-sensitive, therefore Upper case will be sorted before lower case.

```
fruits.sort(key=str.lower)
```

Custom Sort function

Ex:

```
def fun(n):
    return len(n)

fruits = ["apple", "banana", "kiwi"]
fruits.sort(key=fun)
print(fruits)
```

→ Function will return a number that will be used to sort (lower number first)

Copy a list

copy() used to copy the list

```
mylist = oldlist.copy()
```

→ Use the list object to copy

```
mylist = list(oldlist)
```

→ Use the slicing method to copy list

```
mylist = list[:]
```

Join Lists

```
new_list = list1 + list2
new_list = list1.append(item)   # Need a loop
new_list = list1.extend(list2)
```

Tuple

→ Ordered

→ Unchangable (Immutable)

→ Allow duplicate values

```
fruits = ("apple", "banana")
```

len() returns the length of tuple

```
len(fruits)
```

→ To create a tuple with one item,

```
fruits = ("apple",)
```

type() returns <class 'tuple'>

Accessing Tuple Items

```
fruits[1]
fruits[-1]    # Negative indexing
fruits[2:5]   # Slicing
```

→ Check if item exists

```
"apple" in fruits
```

Adding / Removing Items from Tuple

Tuples are immutable

→ You need to convert tuple to list, then modify the list and convert the list to tuple.

→ Tuple Addition is Allowed

```
fruits = ("apple", "banana")
new_fruit = ("orange",)
fruits += new_fruit
```

Unpacking a tuple

```
(fruit1, fruit2, fruit3) = fruits
(fruit1, *fruitlist, fruit3) = fruits
```

→ Use * to collect the remaining value as a list.

Loop through tuple:

```
for item in fruits:
    ....
for i in range(len(fruits)):
    print(fruits[i])
```

Join Tuples

```
new_tuple = tuple1 + tuple2
```

→ Can also multiply tuples

```
new_tuple = fruits * 3
```

count() retuns the no. of times a specific value occurs in a tuple.

index() returns the index value of specified value.

Sets

→ Unordered

→ Unchangable

→ Unindexed

```
fruits = {"apple", "banana"}
```

★ Duplicate items are not allowed ★

True & 1 are considered as same value.

False & 0 are considered as same value.

`len()` returns the count of items in set

`type()` returns `<class 'set'>`

Access Set Items

```
for x in fruits:
    print(x)
```

```
"banana" in fruits      # Returns boolean
"apple" not in fruits
```

Add / Remove Items in a Set

`add()` method to add 1 item

`update()` method to add items from iterable object

`remove()`
`discard()`
} To remove an element

remove Vs discard

→ remove raise an Error if element not found

pop() method removes a random item from the set.

clear() method clears the set

del deletes the entire set

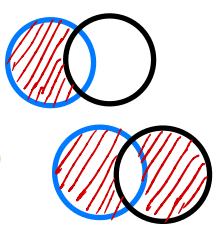
Join Sets

union() / update() (Use '|')

intersection() (Use '&')

difference()

symmetric_difference()



issubset()

issuperset()

isdisjoint()

Dictionary

→ Ordered

→ Changeable

→ Don't allow duplicate

```
thisdict = { key : value }

thisdict [ key ]
thisdict . get (key)
```

len()

type() returns < class 'dict' >

keys() returns all the keys as a list

values() returns all the values as a list

items() returns each item as tuple in a list

Check if key exists in dict

```
key in thisdict   # True / False
```

Change dictionary items

```
thisdict[key] = new_value
thisdict.update({key : value})
```

Add / Remove items

```
thisdict[new_key] = new_value
thisdict.update({new_key : new_value})
```

`pop()` removes the item with specified value in dictionary

`popitem()` removes the last element in the dictionary

clear() method empties the dictionary

del deletes the entire dictionary

Loop through dictionary

```
for x in fruits:
    print(x)          # Prints all keys

for x in fruits:
    print (fruits[x])    # Prints all values

for x in fruits.values:
    print (x)          # Prints all values

for x in fruits.keys:
    print(x)          # Prints all keys

for x in fruits.items:
    print( x, y)

# prints all key, value pairs
```

Copy Dictionary

```
new_dict = thisdict.copy()
new_dict = dict(thisdict)
```

→ A dictionary can contain multiple dictionaries in it. (Nested dictionaries)

IF / ELSE

a == b	Equals
a != b	Not equals
a < b	Less than
a > b	Greater than
a <= b	Less than or equals
a >= b	Greater than or equals

```
if condition:
    ----
    ----
elif condition:
    ----
    ----
else:
    ----
    ----
```

One line if

```
if condition: statement
```

One line if - else

```
print("A") if a>b else print("B")
```

AND
OR
NOT
} Logical Operators

Pass statement

Pass statement is used to avoid getting an error when if block is empty

While Loop

```
while condition:
    -----
    -----

else:
    -----
    -----
```

break statement is used to exit the loop even condition is True

Continue statement is used to skip the current iteration, and continue with the next.

Else statement can run a block of code when the while condition no longer satisfies.

For Loop

```
for x in iterable:
    - - - -
    - - - -

else:
    - - - -
    - - - -
```

Break Statement

Continue statement

range() method returns a sequence of numbers.

```
range(start, end, increment)
```

Else statement

Pass statement

Nested loops

Function

→ A function is a block of code which Runs on when called.

```
def function_name(parameter1):
    ____
    ____
    ____
    return ____
```

```
function_name(parameter1)
```

parameter Vs Argument

→ A parameter is the variable listed inside the parantheses in function definition

→ An Argument is the value sent to the function when it is called.

Arbitary Arguments (* args)

→ Use arbitary Arguments when you dont know the no.of arguments will be passed
→ Recieves a list.

Arbitary keyword Arguments (** kwargs)

→ Recieves a dictionary
→ Must mention key, value while calling the method.

Default Parameter

```
def my_func( name = "Jack"):
    print( name)

my_func ("Alex")   # prints "Alex"
my_func ()         # prints "Jack"
```

return statement to return a value

pass statement

Positional Arguments only

```
def my_func( x , / ):
    - - - -
    - - - -

my_func ( 3 )      # Works
my_func ( x=3 )    # Error
```

key word Arguments Only

```
def my_func ( * , x):
    - - - -
    - - - -

my_func ( 3 )      #Error
my_func ( x=3 )    # Works
```

Combine Positional and keywords

→ Any argument before '/' is positional argument.

→ Any argument after '*' is keyword argument.

positional keyword

```
def func(a, b, /, *, c, d):
    ---
    ---
```

Recursion

Lambda

→ lambda function is an anonymous function which takes multiple args but can only have one expression

lambda arguments : expression

```
x = lambda a : a*2
print ( x(10))      # Prints 20
```

Class

→ Python is a OOP language

→ Almost everything in Python is Object

→ Class is like object constructor or blueprint for creating objects

```
class MyClass:
    ----

obj1 = MyClass()
```

__init__() is the constructor for any object.

__str__() is used to represent the object

Object Methods

→ Methods in objects are functions the belong to the object.

Self Parameter

→ The self parameter is a reference to the current instance of class.

→ You can name whatever you want, but it has to be the first parameter.

Modify Object Properties

```
p1.age = 40
```

```
del p1.age
```

pass statement

Inheritance

→ Inheritance allows a class to inherit all the methods and properties from another class.

→ Parent Class / Base class

→ Child Class / Derived class

```
class Child(Parent):
    pass
```

→ While using __init__ in Child class, always call parent's __init__

```
Parent.__init__(...):
```

(OR)

→ Use super()

```
super.__init__(...):
```

Iterators

__iter()__

__next()__

→ Iterable Objects have iter() method, used to iterator.

Stop Iteration statement to stop the iterati

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

Polymorphism

→ Polymorphism ⇒ Many forms

→ Can have multiple classes with different method names.

```
class Vehicle:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Move!")

class Car(Vehicle):
  pass

class Boat(Vehicle):
  def move(self):
    print("Sail!")

class Plane(Vehicle):
  def move(self):
    print("Fly!")

car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
plane1 = Plane("Boeing", "747")     #Create a Plane object

for x in (car1, boat1, plane1):
  print(x.brand)
  print(x.model)
  x.move()
```

→ len() method is an example of polymorphism

Scope

Local Scope

→ Can be accessed within a block

Global Scope

→ A variable created in the main body of python code is a global variable and belongs to global scope

→ global keyword can be used to declare global variables

→ Use nonlocal to access the outer function variables inside a nested functions

Modules

→ A file containing a set of functions you want to use in your application.

```
import modulename as mn
```

→ Use dir( ) to list all the functions in a module

```
from modulename import methodname
```

Math

min( )
max( )
abs( )
pow( )

math.sqrt( )
math.ceil( )
math.floor( )
math.pi

Must import math module

Try Except

```
Try :
    ____
    _____
    ____

except NameError:
    ____
    ____

except:
    ____
    ____

else:                  # Else will be executed
    ____               # if there are no errors
    ____

finally:               # Finally will be executed
    ____               # regardless of errors.
    ____
```

→ Use raise to raise an exception

```
raise TypeError("Only int are allowed")
```

User Input

```
input("Enter")
```

File Handling

r	read
a	append
w	write
x	create
t	Text mode
b	binary mode

```
f = open("filename.txt", "rt")
```

read() to read the contents of the file.

```
f.read()        # Reads entire content
f.read(5)       # Reads 5 chars
f.readline()    # Reads one line

for x in f:
    print(x)

f.close()       # Close the file.
```

Create a file

```
f=open("file.txt","x")
# New empty file is created
f=open("file.txt","w")
# Created a file if it does not exist
```

File Write

```
f.write("String")
```

Delete a file

```
os.remove("file.txt")
```

check if a file exist

```
if os.path.exists("file.txt"):
    os.remove("file.txt")
```

Delete Folder

```
os.rmdir("FolderName")
```

→ You can only remove empty folders.