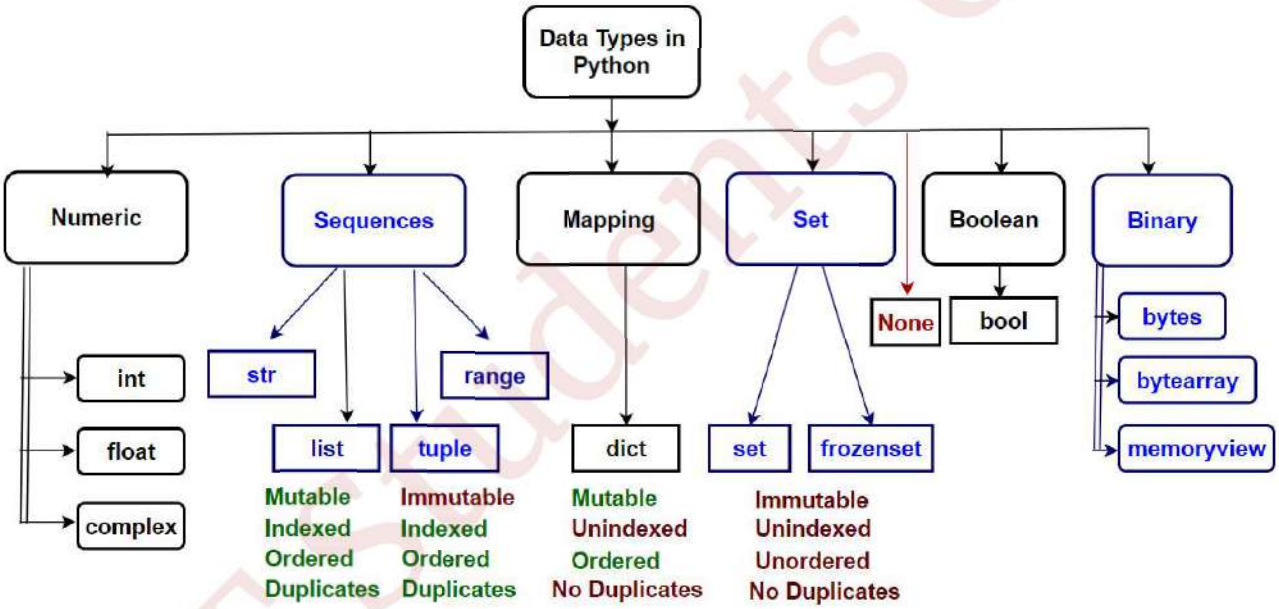
Data Types in Python

Data types are the classification of data items. A Data type represents a kind of value to perform an operation on a particular data. Python has several built-in data types. These data types are used to store and manipulate different types of data in Python.

- **Data types** in Python are actually **classes.**
- **Variables** are **instances** or **objects** of those classes.
- No need to specify a datatype while declaring a variable.
- Python automatically sets a datatype based on the value we assign to a variable.

Python data types are classified into **Primitive** and **Non-Primitive** data types.
Primitive Data types: Numerics (int, float, complex), Character set (str), Boolean, Binary, and None.
Non Primitive Data types: Sequences (str, list, tuple), Mapping (dict), and Sets (set)



type() function is used to find the data type of any variable.

- **Syntax:**

  type(variable name)

- **Ex:** **type(50)**
  Output: **<class 'int'>**

1. Python Numeric Data Type is used to hold numeric values. Python supports integers, floating-point, and complex numbers. Integers are whole numbers, while floating-point numbers have decimal points.

- **int** - holds signed integers of non-limited length. (Ex: 10)
- **float-** holds floating precision numbers and it's accurate up to 15 decimal places. (Ex: 3.14)
- **complex**- holds complex numbers. (Ex: 5 +7j)

Example:

```
#Data type of integer, float and complex numbers
a=70          # variable with integer value
b=70.2345     # variable with float value
c=70+5j       # variable with complex value
print("Data type of",a, " is ", type(a))
print("Data type of",b, " is ", type(b))
print("Data type of",c, " is ", type(c))
```

Output:

Data type of 70 is <class 'int'>
Data type of 70.2345 is <class 'float'>
Data type of (70+5j) is <class 'complex'>

2. Python String Data Type

A string is a **sequence** of one or more characters enclosed in a **single quote, double-quote, or triple-quote.** Multi-line strings are enclosed in triple quotes.
In python, there is no character data type. A character in Python is a string of length one. It is represented by **str** class.

String Assignment using **single quotes ‘ ‘ :**
Syntax: var = ‘ string ‘ **Ex:** lang = ‘Python’

String Assignment using double **quotes “ “ :**
Syntax: var = “ string “ **Ex:** lang = “Python”

String Assignment using **triple quotes ‘ ‘ ‘ or “ “ “ :**
Syntax1: var = ‘’’ string ‘’’ **Ex:** lang = ‘’’Python’’’

Triple quotes are used to assign a **multi-line strin**g to a variable.
Syntax2: var = “”” multi-line string “””
Ex:

```
>>> lang = """ Python is both Structured and
... Object Oriented Programming language """
>>> print(lang)
 Python is both Structured and
Object Oriented Programming language
>>>
```

Ex: or “Python” or ‘’’Python’’’

Access String Characters in Python

We can access the characters in a string in three ways.

- Indexing: Each character in a string is indexed starting from 0.
  var[index]

- Negative Indexing: The index of the last character of a string is -1, the second from the last is -2, and so on.
  var[-index]

- Slicing: Access a range of characters in a string by using the slicing operator colon :
  var[startIndex : endIndex-1]

Example:

```
# string data type  (str)
name = 'CIT'
city = "Guntur"
address = '''777, First line,
Guntur 522001'''
# using , to concatenate 2 or more strings
print(name, city, address)
#using + to concatenate 2 or more strings
print("I study at " + name + " " +city)

print("t" in city)  #True
print(city[0])      #G by accessing string using index
print("I study at %s in %s" % (name,city))
print("I study at {}s in {}".format(name,city))
```

Output:

```
True
CIT Guntur 777, First line,
Guntur 522001
I study at CIT Guntur

True
G
I study at CIT in Guntur
I study at CITs in Guntur
```

Character Sets in Python

The Python character set is a valid set of characters recognized by the Python language. These are the characters we can use during writing a script in Python. Python supports all **ASCII / Unicode** characters that include:

- **Alphabets:** All 52 capital **A-Z (65-90**) and small **a-z (97-122**) alphabets.
- **Digits:** All digits **0-9 (48-57).**
- **Special Symbols:** Python supports all kind of special symbols like, **~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > , < \ |** .
- **White Spaces:** White spaces like tab space(9), blank space(32), newline(10), and carriage return(13) .
- **Other:** All ASCII and UNICODE characters are supported by Python which constitutes the Python character set.

Note: See ASCII table for all ASCII codes in Reference section at the end of this document.

Every character in Python language has its equivalent **ASCII** (American Standard Code for Information Interchange) value. ASCII character set is a subset of **UNICODE** (ex: **UTF-8**, UTF-16, or UTF-32 namely **U**nicode **T**ransformation **F**ormat). Python defaults to using UTF-8.

➔ **chr()** function converts a given integer (0–255) to its ASCII equivalent character string.
- Ex: ch1=**chr(65)**, here ch1 contains ASCII character capital A
- Ex: ch2=**chr(97)**, here ch2 contains ASCII character small a

➔ **ord()** function converts a given character to its ASCII equivalent integer (0–255)
- Ex: a1=**ord('Z'**), where a1 will be assigned the ASCII value 90.

---

3. Python List Data Type - The list is a collection of multiple data elements of same or different data types. A list is ordered, mutable, indexed and allows duplicate elements.

- The list is a **versatile data type** exclusive to Python.
- The list is an **ordered** data **sequence** written in square brackets **[ ]** separated by commas **,** .

List Properties

A. **Ordered -** The list items have a **defined order** and the order will not change. If you add new items to a list, the new items will be placed at the end of the list.
B. **Mutable -** The list is **changeable.** We can change, add, and remove items in a list after its creation.
C. **Indexed -** The list items are **indexed**, the 1st item has index [0], the 2nd item has index [1] and so on.
D. **Allows Duplicates -** Since lists are indexed, lists can have items with the same value.

Method-1:

Syntax: listvariable = [value-1, value-2, . . . value-n]

Method-2:

👉 **list() as a Constructor**

We can use the **list()** constructor to create a list in Python.

Syntax: listvariable = list ((value-1, value-2, . . . value-n)) #notice the 2 parantheses

Accessing Elements in List:

Each element in a List has an index. We can access any element of a List by its index position.

Syntax: listname[index]

- **Indexing:** The list elements are indexed starting from 0; which means, the first item in the list is at index 0.
- **Negative Indexing:** Python also supports negative indexing. Negative indexing starts with -1 at the last element in a list. We can use negative indexing without knowing the length of the list to access the last item.

z =	[3,	7,	4,	2]
index	0	1	2	3
negative index	-4	-3	-2	-1

Example-1: Basic list

```
# list data type
#list of integers
marks = [50,60,70]
print(marks)

#list of strings
subjects = ["English","Maths","Programming"]
print(subjects)

#list of both integers and strings
address = [245,"Amaravathi Rd","Guntur",522001]
```

```
print(address)
#index starts with 0, increments by 1 and prints a single element
print(address[2]) #this will print "Guntur" from list address
```

Output:

[50, 60, 70]
['English', 'Maths', 'Programming']
[245, 'Amaravathi Rd', 'Guntur', 522001]
Guntur

Example-2: Advanced list

```
# list items are ordered.
dept_names = ["Software", "Physics", "Arts"]
dept_codes = [101, 102, 103]
status = [True, False, False]

print(dept_names)
print(dept_codes)
print(status)

# list elements can be different data types
mult_list = ["CIT", 2023, True, 99.77, "2nd Sem"]
print(mult_list)

# len() to find number of elements in a list
print("Length of list",len(dept_names))

# Find the data type of list varaible
print("Data type is",type(dept_names))

# list() as constructor
subjects = list(("English","Chemistry","Maths","Python")) # notice 2
parantheses
print(subjects)
```

Output:

['Software', 'Physics', 'Arts']
[101, 102, 103]
[True, False, False]

['CIT', 2023, True, 99.77, '2nd Sem']
Length of list 3
Data type is <class 'list'>
['English', 'Chemistry', 'Maths', 'Python']

Example-3: Accessing

```
# Define a list
z = [3, 7, 4, 2]
# Access the first item of a list at index 0
print(z[0])
print(z[2])
# Access the lst item of a list at index -1
print(z[-1])
```

Output:
3
4
2

4. Python Tuple Data type - The tuple is a collection of many data items of same or different data type. A tuple is **ordered, immutable, indexed**, and **allows duplicate items.**

- The tuple is an **ordered sequence** of data written in parentheses **( )** separated by commas **','**

Tuple Properties

A. **Ordered -** The tuple items have a **defined order** and the order will not change. If you add new items to a list, the new items will be placed at the end of the list.
B. **Immutable -** The list is **immutable.** That means data in a tuple is **write-protected.** We cannot change, add or remove items after the tuple has been created.
C. **Indexed -** Tuple items are **indexed**, the first item has an index [0], the second item has an index [1], and so on.
D. **Allows Duplicates.** Since lists are indexed, lists can have items with the same value

Method-1: tuple with many items

Syntax: setvariable = (value-1, value-2, . . . value-n)

Method-2: tuple with ONE item

Syntax: tuplevariable = (value-1,) #must use , after the element

Method-3:

👉 **tuple( ) as a Constructor**

We can use the **tuple( )** constructor to create a list in Python.

Syntax: tuplevariable = tuple ((value-1, value-2, . . . value-n)) #notice the 2 parantheses

Example-1: Basic tuple

```
# tuple data type
#tuple of integers
marks = (50,60,50)
print(marks)

#tuple of strings
subjects = ("English","Maths","Programming")
print(subjects)

#tuple of both integers and strings
address = (245,"Amaravathi Rd","Guntur",522001)
print(address)

#index starts with 0, increments by 1 and prints a single element
print(address[2]) #this will print "Guntur" from tuple address
```

Output:

(50, 60, 50)
('English', 'Maths', 'Programming')
(245, 'Amaravathi Rd', 'Guntur', 522001)
Guntur

Example-2: Advanced tuple

```
# tuple items are ordered.
dept_names = ("Software", "Physics", "Arts")
dept_codes = (101, 102, 103)
status = (True, False, False)

print(dept_names)
print(dept_codes)
print(status)
```

```
# tuple elements can be different data types
mult_tuple = ("CIT", 2023, True, 99.77, "2nd Sem")
print(mult_tuple)

# len() to find number of elements in a tuple
print("Length of tuple",len(dept_names))

# Find the data type of tuple varaible
print("Data type is",type(dept_names))

# tuple() as constructor
subjects = tuple(("English","Chemistry","Maths","Python")) # notice 2
parantheses
print(subjects)

# tuple with single item and a comma
subjects = ("Python",)   # a tuple
print("Single element",type(subjects))

# NOT tuple with single item WITHOUT comma
subjects = ("Python")   # just a string
print("Single element",type(subjects))
```

Output:

('Software', 'Physics', 'Arts')
(101, 102, 103)
(True, False, False)
('CIT', 2023, True, 99.77, '2nd Sem')
Length of tuple 3
Data type is <class 'tuple'>
('English', 'Chemistry', 'Maths', 'Python')
Single element <class 'tuple'>
Single element <class 'str'>

5. Python Dictionary

Dictionary is used to store data values in **key : value** pairs and can be **referenced** by using the **key** name.

Dictionary is a collection of Ordered, Mutable, Unindexed and does NOT allow duplicates.

- Dictionaries are written within curly braces **{ }** in the form of **key : value.**
- Dictionary is useful to access a large amount of data efficiently.

Dictionary Properties:

A. **Ordered -** The dictionary items have a defined order and the order will not change.
B. **Mutable or changeable -** Dictionaries are changeable. We can change, add or remove items after the dictionary has been created.
C. **Not Indexed -** The dictionary elements are NOT indexed; rather, the elements are **referenced** by using the **key** name.
D. **No Duplicates -** Dictionaries cannot have two items with the same key. Duplicate key:value will overwrite existing values.

Method-1:

Syntax: dictvariable = { key-1:value-1, key-2:value-2, . . . key-n:value-n }

Method-2:

👉 **dict() as a Constructor**

We can use the **dict()** constructor to create a set in Python.

Syntax: dictvariable = set ((key-1:value-1, key-2:value-2, . . . key-n:value-n))
#notice the 2 parantheses

len() - len(dictionary) will result in the number of items in the dictionary.
type() - finds data type of the object which is dict

Example-1: Basic Dictionary

```
#dictionary variable
a = {1:"Abdul",2:"Kalam", "age":60}

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

Output:

Abdul
Kalam
60

Example-2: Advanced Dictionary

```
#dictionary variable
cars = {
  "brand": "Hyundai",
```

```
  "model": "Creta",
  "year": 2023
}
# prints all dictionary
print(cars)
# prints selected key's value
print(cars["brand"])

cars = {
  "brand": "Hyundai",
  "model": "Creta",
  "year": 2023,
  "year": 2015
}
# prints selected key's value
print(cars["year"])

# len() to find number of elements in a dict
print("Length of dict",len(cars))

# Find the data type of dictionary varaible
print("Data type is",type(cars))
```

Output:

{'brand': 'Hyundai', 'model': 'Creta', 'year': 2023}
Hyundai
2015
Length of dict 3
Data type is <class 'dict'>

6. Python Set Data type - Sets are used to store multiple items of different data types in a single variable.

Set is a collection of data that is unordered, unchangeable, unindexed, and duplicate values are not allowed.

The data items in sets are enclosed in curly braces **{ }** and separated by commas ‘,’.

Set Properties:

A. **Unordered -** Set items do not have a defined order. They can appear in a different order each time we access them. They cannot be referred by index or key.
B. **Unimmutable or Unchangeable -** Set items cannot be changed after the set has been created. However, we can remove existing items and add new items.

C. **Unindexed -** Set items are not indexed. So, we can't access set items by an index.
D. **No Duplicates allowed -** Sets cannot have two items with the same value.

Note: The values **True** and **1** are the same in sets, and are treated as duplicates.

Method-1:

Syntax: setvariable = { value-1, value-2, . . . value-n }

Method-2:

👉 **set() as a Constructor**

We can use the **set()** constructor to create a set in Python.

Syntax: setvariable = set ((value-1, value-2, . . . value-n)) #notice the 2 parantheses

Example-1: set basics

```
# set of integers
marks = {50,60,50,40}
print(marks)

# set of string
subjects = {"English","English","Maths","English","Programming","Maths"}
print(subjects)

# set of both integers and strings
address = {245,"Amaravathi Rd","Guntur",522001}
print(address)
```

Output:

{40, 50, 60}
{'English', 'Programming', 'Maths'}
{522001, 'Amaravathi Rd', 245, 'Guntur'}

Example-2: set advanced

```
# set items are unordered. The items will appear in a random order.
# Rerun the program to see the change in results

dept_names = {"Software", "Physics", "Arts"}
dept_codes = {101, 102, 103}
status = {True, False, False}
```

```
print(dept_names)
print(dept_codes)
print(status)

# set elements can be different data types
mult_set = {"CIT", 2023, True, 99.77, "2nd Sem"}
print(mult_set)

# len() to find number of elements in a set
print("Length of set",len(dept_names))

# Find the data type of set varaible
print("Data type is",type(dept_names))

# set() as constructor
subjects = set(("English","Chemistry","Maths","Python")) # notice 2
parantheses
print(subjects)
```

Output:

{'Physics', 'Software', 'Arts'}
{101, 102, 103}
{False, True}
{True, 99.77, 2023, 'CIT', '2nd Sem'}
Length of set 3
Data type is <class 'set'>
{'Maths', 'English', 'Python', 'Chemistry'}

Data Type	Ordered?	Mutable (changeable)?	Indexed?	Duplicates allowed?
list	Ordered	Mutable (changeable)	Indexed	Allows Duplicate members
tuple	Ordered	Immutable (unchangeable)	Indexed	Allows Duplicate members
dict	Ordered (>=Py3.7)	Mutable (changeable)	Unindexed but uses Key	No Duplicates members
set	unordered	Immutable (unchangeable) However, add & remove of members are possible	Unindexed	No Duplicates members

7. None Data Type

- None data type is an object of NoneType class
- None is used to define a null value or no value at all.
- None can only be equal to None.
- None can be assigned to any variable, but new NoneType objects cannot be created.

Syntax:

```
var = None
```

1. None is not the same as False.
2. None is the same as 0.
3. None is not considered an empty string.
4. Returns False if we compare None with anything, except while comparing it to None itself.
5. A variable can be returned to its initial, empty state by being assigned the value of None.
6. None keyword provides support for both **is** and **==** operators.

8. Python Boolean Data Type

Boolean is a data type that has one of two possible values, **True** or **False.** They are mostly used in creating the control flow of a program using conditional statements.

Syntax:

```
var1 = True
var2 = False
```

Example: Boolean

```
# boolean (bool) data type in Python.
x = True
y = False
# Prints boolean result
print("x is ", x)
print("y is ", y)
print()
# Data type of True and False
print("Data type of 'True':", type(x))
print("Data type of 'False':", type(x))
```

Output:

```
x is  True
y is  False
Data type of 'True': <class 'bool'>
Data type of 'False': <class 'bool'>
```

9. Python Binary Data Type

Following are the 3 binary data types in Python.

- **bytes**
- **bytearray**
- **memoryview**

	bytes	bytearray	memoryview
Definition	**bytes** and **bytearray** are used to manipulate binary data in python. They are often used to **send data over networks** in byte streams **for compatibility and reliability** without depending on network to decode data.		**memoryview** is a buffer protocol that accesses objects of **bytes** and **bytearrays**.
Syntax Data type	var_b = b"string" A string preceded by b	var_barr = b"string" A string preceded by b	var_mv = memoryview(byte object var)
Syntax Function	bytes(source, encoding, errors) Returns byte object. encoding - utf8, ascii error - ignore, replace strict	bytearray(source, encoding, errors) Returns byte array of objects. encoding - utf8, ascii error - ignore, replace strict	
Properties	• **bytes()** function returns objects that are immutable or cannot be changed. • Returns small integers in the range 0 <= x < 256 and print as ASCII characters when displayed.	• **bytearray()** function returns objects that are mutable or can be changed. • Returns small integers in the range 0 <= x < 256 and print as ASCII characters when displayed.	**memoryview** can access the memory of other binary objects (**bytes, bytearray**) without copying the actual data.
Example Data type	b1=b'First bytes' b2=b"Second bytes" b3=b'''Third bytes'''	barr1=b'First bytes' barr2=b"Second bytes" barr3=b'''Third bytes'''	mv1=memoryview(b1) mv2=memoryview(barr2)

Example Function	`>>> b="Hello"`
	`>>> result = bytes(b,"ascii")`
	`>>> print(result)`
	`b'Hello'`
		`>>> b="Hello"`
		`...`
		`>>> result = bytearray(b,"ascii")`
		`...`
		`>>> print(result)`
		`...`
		`bytearray(b'Hello')`	

Note: A byte is a memory space that consists of 8 bits of binary digits (0 or 1). Python uses the UTF-8 Unicode character set. Each character (alphabet, number, or symbol) in a string occupies 1 byte of memory space.

More Data Type Conversion Functions

➔ **chr()** function converts a given integer (0–255) to its ASCII equivalent character string.
- Ex: ch1=**chr(65)**, here ch1 contains ASCII character capital A
- Ex: ch2=**chr(97)**, here ch2 contains ASCII character small a

➔ **ord()** function converts a given character to its ASCII equivalent integer (0–255)
- Ex: a1=**ord('Z'**), where a1 will be assigned the ASCII value 90.

➔ **complex()** function is used to print a complex number with the value ***real + imag*j*** or convert a string or number to a complex number. Ex:
- c1=complex(3,4)
- print(c1) outputs: (3+4j)

➔ **hex()** function converts an integer number (of any size) to a lowercase hexadecimal string prefixed with "0x".
- Ex: i_to_h=hex(255), i_to_h contains '0xff' and
- Ex: i_to_h=hex(16), i_to_h contains '0x10'

➔ **oct()** function converts an integer number (of any size) to an octal string prefixed with "0o" using.
- Ex: v1=oct(8), where v1 contains '0o10' and
- Ex: v2=oct(16), where v2 contains '0o20'

Escape Sequences in Python

To insert characters that are illegal in a string, use an escape character.
An escape character is a backslash \ followed by the character you want to insert.

Escape Sequence	Description	Example
\n	New line	print("Hello \nCIT") Output: Hello CIT
\t	Horizontal Tab	print("Hello \tCIT") Output: Hello CIT
\v	Vertical Tab	print("Hello \vCIT") Output: Hello CIT
\b	Backspace	print("Hello \bCIT") Output: HelloCIT
\r	Carriage Return	print("Hello \rCIT") Output: Hello CIT
\f	Form Feed	To start on next page
\'	Single Quote	print("I\'m at CIT") Output: I'm at CIT
\"	Double Quote	print("College is \"CIT\" ") Output: College is "CIT"
****	Backslash	print("Name: \\ name \\") Output: Name: \ name \
\ooo	A backslash followed by three integers will result in a octal value	txt = "\110\145\154\154\157" print(txt) Output: Hello
\xhh	A backslash followed by an 'x' and a hex number represents a hex value	txt = "\x48\x65\x6c\x6c\x6f" print(txt) Output: Hello

Using Functions and Modules in Python

Definition of a Function:

- A function is a block of code used to perform a specific task.
- A complex problem can be divided into **smaller** functions.
- We can define many functions, but a function **runs only when it is called**.
- We can pass data into functions as **arguments**, also known as **parameters**.
- Functions are **reusable**; write the code once, and use it many times.

Python has 3 types of functions: **Built-in** functions (Standard library), **Built-in Module** functions, and **User-defined** functions

1. **Built-in functions** (**Standard library)** are the **built-in** Python functions that can be used directly in our program.
   - **pow() -** returns the power of a number
   - **abs()** - returns a positive number
   - **round() -** returns a rounded precision number
   - **>>> dir(__builtins__)** - displays all built-in functions in Python

   ➔ We should supply a value as an **argument** to a function. Ex: round(9.27,1) the first arguments is required and the second argument is optional.

```
>>> pow(3,2)
9
>>> round(9.37)
9
>>> round(9.37,2)
9.37
>>> abs(-7)
7
>>> abs((2-7)+1)
4
>>>
```

First, the expressions are evaluated and the result is used as input to function. And then the function executes. Here in abs() function, first (2-7)+1 is evaluated to -4 and then abs(-4) results to 4.

2.1 Built-in Module functions - Python has built-in modules that provide several functions. These **modules need to be imported** using **'import'** into our program to use the functions in those modules. Few such modules are,

- Math Module
- cMath Module
- Random Module
- Requests Module
- Statistics Module

Syntax:

Import moduleName
var = moduleName.func-1(values)

➔ **Note:** When we use 'import' a module, we must use the . member ship operator to use its function.

- **>>> dir(math)** - displays all built-in functions in math module

Example:

Note: To use a functions from a module - write the name of a module, followed by a dot . membership operator and the name of the function. Example: **math.sqrt(81)**

```
import math
st = math.sqrt(81)
s = math.sin(0)
c = math.cos(0)
print(st)
print(s)
print(c)

Output:
9.0
0.0
1.0
```

2.2 Built-in Module functions using 'from' clause - the **functions within the modules can directly be imported** before their use using '**from**' and '**import**' keyword.

Syntax:

from moduleName import func-1,func-2, … func-n
var-1 = func-1(values)
var-2 = func-2(values)

Example:

➔ **Note:** When we use 'from' and 'import', we can directly call function name WITHOUT using the . membership operator.

```
from math import sqrt, sin, cos
st = sqrt(81)
s = sin(0)
c = cos(0)
print(st)
print(s)
print(c)
```

```
Output:
9.0
0.0
1.0
```

- **dir(module name)** function lists all the functions of a module

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh
', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign
', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1'
, 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot
', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lg
amma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi
', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
'tau', 'trunc', 'ulp']
```

3. User-defined functions - We can also create our own functions based on our requirements.

Syntax: Python Function Declaration
def function_name(arguments): ** # function body** ** return**

Here,

- **def** - keyword used to declare a function
- **function_name -** any name given to the function
- **arguments** (optional) - any value passed to function
- **return** (optional) - returns value from a function

Example:

```
# User defined function
# declaration & definition
def welcome():
    print("Welcome to Python at CIT")
# function call
welcome()
```

Output:

Welcome to Python at CIT

Reference

1. **Table-1** is the list of builtin functions in Python for your quick reference.
   a. You can also obtain this list at Python Interactive Shell using IDLE
   b. **Syntax: dir(class name)** lists all builtin standard library functions in Python
   c. **>>> dir(__builtins__)**

Table-1: Python Built-In Functions		
S.No	**Function**	**Description**
1	**abs()**	Returns the absolute value of a number
2	**all()**	Returns True if all items in an iterable object are true
3	**any()**	Returns True if any item in an iterable object is true
4	**ascii()**	Returns a readable version of an object. Replaces none-ascii characters with escape character
5	**bin()**	Returns the binary version of a number
6	**bool()**	Returns the boolean value of the specified object
7	**bytearray()**	Returns an array of bytes
8	**bytes()**	Returns a bytes object
9	**callable()**	Returns True if the specified object is callable, otherwise False
10	**chr()**	Returns a character from the specified Unicode code.
11	**classmethod()**	Converts a method into a class method
12	**compile()**	Returns the specified source as an object, ready to be executed
13	**complex()**	Returns a complex number
14	**delattr()**	Deletes the specified attribute (property or method) from the specified object
15	**dict()**	Returns a dictionary (Array)
16	**dir()**	Returns a list of the specified object's properties and methods
17	**divmod()**	Returns the quotient and the remainder when argument1 is divided by argument2
18	**enumerate()**	Takes a collection (e.g. a tuple) and returns it as an enumerate object
19	**eval()**	Evaluates and executes an expression
20	**exec()**	Executes the specified code (or object)
21	**filter()**	Use a filter function to exclude items in an iterable object
22	**float()**	Returns a floating point number
23	**format()**	Formats a specified value

24	**frozenset()**	Returns a frozenset object
25	**getattr()**	Returns the value of the specified attribute (property or method)
26	**globals()**	Returns the current global symbol table as a dictionary
27	**hasattr()**	Returns True if the specified object has the specified attribute (property/method)
28	**hash()**	Returns the hash value of a specified object
29	**help()**	Executes the built-in help system
30	**hex()**	Converts a number into a hexadecimal value
31	**id()**	Returns the id of an object
32	**input()**	Allowing user input
33	**int()**	Returns an integer number
34	**isinstance()**	Returns True if a specified object is an instance of a specified object
35	**issubclass()**	Returns True if a specified class is a subclass of a specified object
36	**iter()**	Returns an iterator object
37	**len()**	Returns the length of an object
38	**list()**	Returns a list
39	**locals()**	Returns an updated dictionary of the current local symbol table
40	**map()**	Returns the specified iterator with the specified function applied to each item
41	**max()**	Returns the largest item in an iterable
42	**memoryview()**	Returns a memory view object
43	**min()**	Returns the smallest item in an iterable
44	**next()**	Returns the next item in an iterable
45	**object()**	Returns a new object
46	**oct()**	Converts a number into an octal
47	**open()**	Opens a file and returns a file object
48	**ord()**	Convert an integer representing the Unicode of the specified character
49	**pow()**	Returns the value of x to the power of y
50	**print()**	Prints to the standard output device
51	**property()**	Gets, sets, deletes a property
52	**range()**	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)
53	**repr()**	Returns a readable version of an object
54	**reversed()**	Returns a reversed iterator

55	**round()**	Rounds a numbers
56	**set()**	Returns a new set object
57	**setattr()**	Sets an attribute (property/method) of an object
58	**slice()**	Returns a slice object
59	**sorted()**	Returns a sorted list
60	**staticmethod()**	Converts a method into a static method
61	**str()**	Returns a string object
62	**sum()**	Sums the items of an iterator
63	**super()**	Returns an object that represents the parent class
64	**tuple()**	Returns a tuple
65	**type()**	Returns the type of an object
66	**vars()**	Returns the __dict__ property of an object
67	**zip()**	Returns an iterator, from two or more iterators

2. **Table-2** is the list of external functions in '**math**' module for your quick reference.
   a. You can also obtain this list at Python Interactive Shell using IDLE
   b. **Syntax: dir(module name)** lists all functions in the module
   c. **>>> dir(math)**

Table-2: math Module Functions in Python		
S.No	**Function**	**Description**
1	**Method**	**Description**
2	math.acos()	Returns the arc cosine of a number
3	math.acosh()	Returns the inverse hyperbolic cosine of a number
4	math.asin()	Returns the arc sine of a number
5	math.asinh()	Returns the inverse hyperbolic sine of a number
6	math.atan()	Returns the arc tangent of a number in radians
7	math.atan2()	Returns the arc tangent of y/x in radians
8	math.atanh()	Returns the inverse hyperbolic tangent of a number
9	**math.ceil()**	Rounds a number up to the nearest integer
10	math.comb()	Returns the number of ways to choose k items from n items without repetition and order
11	math.copysign()	Returns a float consisting of the value of the first parameter and the sign of the second parameter
12	**math.cos()**	Returns the cosine of a number

13	math.cosh()	Returns the hyperbolic cosine of a number
14	math.degrees()	Converts an angle from radians to degrees
15	math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
16	math.erf()	Returns the error function of a number
17	math.erfc()	Returns the complementary error function of a number
18	**math.exp()**	Returns E raised to the power of x
19	math.expm1()	Returns Ex - 1
20	**math.fabs()**	Returns the absolute value of a number
21	**math.factorial()**	Returns the factorial of a number
22	**math.floor()**	Rounds a number down to the nearest integer
23	**math.fmod()**	Returns the remainder of x/y
24	math.frexp()	Returns the mantissa and the exponent, of a specified number
25	**math.fsum()**	Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)
26	math.gamma()	Returns the gamma function at x
27	**math.gcd()**	Returns the greatest common divisor of two integers
28	math.hypot()	Returns the Euclidean norm
29	**math.isclose()**	Checks whether two values are close to each other, or not
30	math.isfinite()	Checks whether a number is finite or not
31	math.isinf()	Checks whether a number is infinite or not
32	**math.isnan()**	Checks whether a value is NaN (not a number) or not
33	math.isqrt()	Rounds a square root number downwards to the nearest integer
34	math.ldexp()	Returns the inverse of math.frexp() which is x * (2**i) of the given numbers x and i
35	math.lgamma()	Returns the log gamma value of x
36	**math.log()**	Returns the natural logarithm of a number, or the logarithm of number to base
37	math.log10()	Returns the base-10 logarithm of x
38	math.log1p()	Returns the natural logarithm of 1+x
39	math.log2()	Returns the base-2 logarithm of x
40	math.perm()	Returns the number of ways to choose k items from n items with order and without repetition
41	**math.pow()**	Returns the value of x to the power of y
42	math.prod()	Returns the product of all the elements in an iterable

43	math.radians()	Converts a degree value into radians
44	math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
45	**math.sin()**	Returns the sine of a number
46	math.sinh()	Returns the hyperbolic sine of a number
47	math.sqrt()	Returns the square root of a number
48	**math.tan()**	Returns the tangent of a number
49	math.tanh()	Returns the hyperbolic tangent of a number
50	math.trunc()	Returns the truncated integer parts of a number
51	**Constant**	**Description**
52	**math.e**	Returns Euler's number (2.7182...)
53	math.inf	Returns a floating-point positive infinity
54	math.nan	Returns a floating-point NaN (Not a Number) value
55	**math.pi**	Returns PI (3.1415...)
56	math.tau	Returns tau (6.2831...)

ASCII Table

ASCII Table

Code	Char	Code	Char	Code	Char	Code	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL