Father of Python **Guido van Rossum**, and released in **1991**.

**ASCII values** 0-->48, A-->65, a-->97

to **get** the ASCII value use "**ord**(a)”--> will get the output(97)

**Numeric Types**

* **Int**, or integer, is a whole number, positive or negative, without decimals, of unlimited length.
* **Float**, or "floating point number" is a number, positive or negative, containing one or more decimals.
* Float can also be scientific numbers with an "e" to indicate the power of 10.
* **Complex** numbers are written with a "j" as the imaginary part:
* Python has a **built-in module** called **random** that can be used to generate random numbers: **random.randrange() 🡪 this is used for generating OTP**

**Boolean:** The **bool()** function allows you to evaluate any value, and give you True or False in return(Syntax: print(bool(x))

* Almost any value is evaluated to **True if it has some sort** of content.
* **Any string is True**, except **empty** strings.
* Any number is True, **except 0**.
* Any **list**, tuple, **set**, and **dictionary** are True, except empty ones.
* Check if an object is an integer or not: (syntax: print(isinstance(x, int))

**Python Collections (Arrays)**

There are **four collection** **data types** in the Python programming language:

* **List** is a collection which is **ordered and changeable**. Allows **duplicate** members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is **ordered and unchangeable**. Allows **duplicate** members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is **unordered** and unindexed. **No duplicate** members.
* [**Dictionary**](https://www.w3schools.com/python/python_dictionaries.asp) is a collection which is **unordered and changeable**. **No duplicate** members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

**LIST:**

* **List method(pop**->will delete**, delete** will delete the list **& clear** will clear the elements & makes the list empty.

**List Comprehension:**

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

**List Sort Descending**

To sort descending, use the keyword argument reverse = True:

**Syntax**: thislist.sort(reverse = True)

\*\*\***Diff between Array & List**\*\*\*\*

* In **List** we can have **different elements**(int, strin, float) and in **array** only same type of elements are allowed.
* Also for array we have to import array module and for list directly we can define.

**Set:** In set it **removes** the duplicate values at the time of **printing** the output(syntax: **set is define is {}**)

The only diff b/w **list** and **set** is list can have **duplicate** values whereas **set** cannot.

🡪 To **add** one item to a set use **the add()** method. (syntax: xyz.add("apple")

* To **add** items from another set into the current set, use the **update()** method.
* The object in the update() method does not have be a **set**, it can be **any iterable object** (tuples, lists, dictionaries et,).
* Diff b/w **remove() & discard()** methods are **discard will not raise** an error if the item doesn’t exist in set.
* Sets are *unordered*, so when using the **pop() method**, you do not know which **item that gets removed.**
* The **intersection\_update()** method will keep only the items that are **present in both sets**. (Syntax: x.intersection\_update(y))
* The **intersection()** method will return a ***new* set**, that only contains the items that are present in **both sets**.
* The **symmetric\_difference\_update()** method will keep only the **elements that are NOT** present in both sets. (Syntax: x.symmetric\_difference\_update(y))

**Dictionary:**

* There is also a method called **get()** that will give you the same result: **Get the value** of the "model" key: x = thisdict.get("model")
* The **keys()** method will **return a list of all the keys** in the dictionary. (syntax: xyz.keys())
* The **values()** method will return a **list of all the values** in the dictionary. Syntax: xyz.values()
* The **items()** method will return **each item in** a dictionary, **as tuples in a list**.
* The **popitem()** method removes the **last inserted item** (in versions **before 3.7**, a random item is removed instead):
* The **del** keyword removes the item with the specified **key name**: Syn del thisdict["model"]
* The **del k**eyword can also delete the dictionary completely: syn del thisdict

**Imp \*\* “““ ””” Triple** Double/single quotes are used to add multiline comments

**Casting:** defining (integer, string, float) specifically.

**X= str(‘3’), Y= int(3), z= float(3) 🡪 output will be (3, 3, 3.0)**

* To **get** the type of **data type** use **🡪 type(x), type(y)**

**SLICING/INDEXING:**

* Specify the **start index** and the **end** index, **separated** by a **colon**, to return a part of the string.
* [2:5]

**Strip() Method:** Strip() method removes the whitespaces either from beginning or end (syntax xyz.strip())

**Replace()**: The replace() method replaces a string with another string: (syntax xyz.replace(‘H’, ‘J’))

**Split()**: The split() method returns a list where the text between the specified separator becomes the list items. (syntax🡪 xyz.split(‘,’)

**Escape Character**: To insert characters that are illegal in a string, use an escape character(\).

* txt = "We are the so-called \"Vikings\" from the north."

**Vairable**

* A variable **name** can only contain **alpha-numeric** characters and **underscores** (A-z, 0-9, and \_ )
* **Variable** names are **case-sensitive** (age, Age and AGE are three different variables)
* **Illegal** variable names:

2myvar = "John"  
my-var = "John"  
my var = "John"

* **To use Global variable** inside function have to define global variable inside the function using global keyword.
* A variable created inside a function belongs to the local scope/Variable of that function, and can only be used inside that/any inside function. A variable created inside a function is available inside that function:
* The local variable can be accessed from a function within the function:
* **Naming Variable**: If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function):
* The function will print the local x, and then the code will print the global x:

1. Instance variable

2. Class Variable

**Methods**

1. Class Method--> for using class varaible class method is called

2. Instance Method --> For using instance varailbe we use instance method

3. Static Method--> if u have to do extra with the class go with the static method

**IF Else Loops:**

* This technique(short hand IF & Else) is known as **Ternary Operators**, or **Conditional Expressions**.

**Lambda/Anonymous Function:**

* A **lambda function** can take any **number of arguments**, but can only have **one expression**. (syn: lambda arguments : expression

x = lambda a : a + 10  
print(x(5)) #here a=5

* **Use lambda functions** when an anonymous function is required for a **short period of time**.

**Class**

**\_\_Init\_\_() Function:** All classes have a function called **\_\_init\_\_(),** which is always **executed w**hen the class is **being initiated**.

**Inheritance:** The child's \_\_init\_\_() function **overrides** the inheritance of the parent's \_\_init\_\_() function, To keep the inheritance of the parent's \_\_init\_\_() function, add a call to the parent's \_\_init\_\_() function:

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

**Self Parameter:** The **self parameter** is a reference to the **current instance** of the class, and is used to **access variables** that belong to the class.

**Method**: If you add a method in the child class with the same name as a function in the parent class, the inheritance of the parent method will be overridden.

**Dir() inside Module**: There is a built-in function to list all the function names (or variable names) in a module. The dir() function:

syntax of printing all variables and function names of the "mymodule" module? Print(dir(modulename))

**File Handling:**

* Syntax for file opening “f = open("demofile.txt", "rt")”
* r is for read and t=text which is not required as the file will be in text mode always
* "r" - Read - Default value. Opens a file for reading, error if the file does not exist
* "a" - Append - Opens a file for appending, also creates the file if it does not exist and this will append in last/end line if the file already exists.
* "w" - Write - Opens a file for writing, creates the file if it does not exist, and over writes the content if available in existing file.
* "x" - Create - Creates the specified file, returns an error if the file exists
* Can only read part of character using syn: print(f.read(**5**))
* Also only lines can be read using syn: print(f.read(readline))

List Comprehension

tuples = [t for t in tuples if t] # this will remove the empty tuples in the list

also [x for x in list if x%2!=0] x!=[]) # this will prevent element which will not divide by 2 != will not print the rhs part.

Separator and end can be called in print statement or with return function [syn: sep=’@’ end= ‘ ’]

Write a Python program to print the documents (syntax, description etc.) of Python built-in function(s).

*Sample function*: abs()

Code: print(abs.\_\_doc\_\_)