**Beginners**

**List**: it has []

i.e : fruits = ["apple", "banana", "cherry"]

fruits[0] to retrieve info out of list at index 0 which here is “apple”

fruits[0] = "orange" # it will replace "apple" with "orange"

fruits.append("kiwi") # will add kiwi into the list (in the last place after cherry)

**Dictionary**: it has {}

dictionary has key and value (if the key and value information is so long use the indentation as below)

list only has simple values (no keys.)

programming\_dictionary = {

"Bug": "An error in a program that prevents the program from running as expected.",

"Function": "A piece of code that you can easily call over and over again."

}

retrieving item from dictionary (you will need to provide key)

print(programming\_dictionary["Bug"])

adding key: value to existing dictionary

programming\_dictionary["Loop"] = "The action of doing something over and over again"

Loop through dictionary

for key in programming\_dictionary:

print(key) # it will print all the keys

print(programming\_dictionary[key]) # it will print all the values

Nesting Dictionaries and Lists

ie : dict1 = {key1: [list1], key2 : {dict2}}

**Functions**: three types

1. Normal Function : def my\_function() :
2. Function with the input : def my\_function(Parameter):
3. Function with the output : def my\_function():

return 3\*2

""" Take first and last name and convert it to Title Case""" # this is doc string (when you hover over the function name it shows this info) , it has to be written right after function name like below

def format\_name(f\_name, l\_name):

  """ Take first and last name and convert it to Title Case""" # this is doc string (when you hover over the function name it shows this info)

  if f\_name == "" or l\_name == "":

    return "You didn't provide valid inputs." # (multiple return statemnents)

  formated\_f\_name = f\_name.title()

  formated\_l\_name = l\_name.title()

  return f"Result: {formated\_f\_name} {formated\_l\_name}"

**Scope:**

Local scope : when you create any variable inside the function , you can only access that variable from inside the function and if you try to access that variable from outside of the function you will get an error message.

################### Scope ####################

enemies = 1

def increase\_enemies():

  enemies = 2

  print(f"enemies inside function: {enemies}")

increase\_enemies()

print(f"enemies outside function: {enemies}")

here output would be  
enemies inside function: 2

enemies outside function: 1

Global scope: when you define variable outside of the function, it is accessible from inside as well as outside of function and thus name global.

so if you want to modify the global variable from inside the function, that you need to declare that it is global variable and then you can modify it (But avoid doing this as it creates more errors !!).

# modifying global variable from inside the function

enemies = 1

def increase\_enemies():

  global enemies

  enemies += 2

  print(f"enemies inside function: {enemies}")

increase\_enemies()

print(f"enemies outside function: {enemies}")

################################################################################################

**Intermediate**

**Object oriented programming (OOP):**

* it is much better approach then the Procedural programming. it splits the large tsk into a smaller feasible pieces which are call modules. it uses classes and Object method. all the new era’s programming languages are OOP.
* object = it is combination of attributes and methods
* attributes = they are variable but attached to specific object
* methods= they are functions but attached to specific object
* but you can generate many objects from the same blueprint, and this blueprint is called class.
* car = CarBlueprint()
* here car is an object and CarBlueprint() is class

**Creating your own class and objects:**

* you can create class by simply putting keyword “class” in from of class name.
* class User:

    print("this is a user class")

* initialize = when object is being initialize we can put some predefine attributes to it using initializing.
* class Car:

    def \_\_init\_\_(self) :

* so this \_\_init\_\_ function will be called every time you create an object from class Car and whatever you define under this class will run.

**Modules/ Packages:**

* Aliasing the module :
  + import turtle as t
  + so if the name of the module is very long and you do not keep writing it again and again then use aliasing which makes our code much simpler.
* Installing Modules:
  + not all the modules can be simply imported, some of them needs to be installed first before we can import them in our code.
  + you can download any package from here : [PyPI · The Python Package Index](https://pypi.org/)

**Tuple:**

* Tuple is datatype in python
* my\_tuple = (1, 2, 3) : it has round brackets, and each item is separated by comma
* it is very much similar to list
* the difference is that tuple is stone carved so that the values inside tuple cannot be change (immutable)
* my\_tuple[0] = 5 : this operation is not possible as we cannot change the tuple values afterwards.

**Higher-order function:**

* it is function that takes another function as argument.
* def add(n1, n2):
* return n1 + n2
* def calculator(n1, n2, func):
* result = func(n1, n2)
* print(result)
* so here calculator is higher order function which uses add function as one of it’s argument.
* notice when we pass function as argument we do not use () besides it, it is just name of function.

**Class-Inheritance :**

* a class can inherit functions and attributes of another class.
* class Fish(Animal):
* def \_\_init\_\_(self):
* super().\_\_init\_\_()
* here Fish class inherits from Animal class.
* super = it refers to the super class which here is Animal
* super().\_\_init\_\_() : it initialize everything from super class
* Fish is subclass here.

**Slicing:**

* if you want to get certain items from the list, then use slicing
* piano\_keys = ["a", "b", "c", "d", "e", "f", "g"]
* piano\_new\_keys = piano\_keys[2:5]
* here we will get [“c’, “d”, “e”] in piano\_new\_keys

**How to open, read and write to files using “with” keyword :**

* file = open("my\_file.txt")  
  contents = file.read()  
  print(contents)
* with method:
* with open("my\_file.txt", mode="w") as file:  
   file.write("added new text !!")

**Pandas Library:**

* super useful for the data analysis
* data\_frame = entire table of the data
* series = single column out of the data table
* print(data["temp"]) # pandas is intelligent enough to know the heading of the column and return the data underneath it >>>> super useful

**List Comprehension:**

* allows you to create new\_list from existing\_list in just one line of code.
* # syntax is as below  
  new\_list = [new\_item for item in existing\_list]
* example is as below
* numbers = [1, 2, 3]  
  new\_number\_list = [n+1 for n in numbers]
* it also allows you to add the if condition
* names = ["Viral", "Mikin", "Amrut", "Panth", "alex", "ami"]  
  five\_letter\_name = [name for name in names if len(name) == 5 ]
* output: ['Viral', 'Mikin', 'Amrut', 'Panth']

**Dictionary Comprehension:**

* it is same as list comprehension
* syntax: new\_dict= {new\_key:new\_value for item in list}  
  syntax: new\_dict= {new\_key:new\_value for (key,value) in dict.item() }  
  syntax: new\_dict= {new\_key:new\_value for (key,value) in dict.item() if test }
* with pandas we can also create a new dict.
* new\_dict = {new\_key:new\_value for (index, row) in df.iterrows()}

**TKinter:**

* helps you create the GUI
* things you can use within TKinter are : Label, Button, Entry (input box), Text, Spinbox, Scale, Checkbutton, Radiobutton, Listbox etc.
* there are three layouts : pack(), place(), grid()

**Advance Python Arguments:**

* def my\_function (a, b, c): here a , b and c are ***simple arguments*** , and when we call the function we provide these arguments.
* def my\_function (a=1, b=2, c=3): functions with the ***default arguments.***
* def add(\*args): it takes unlimited arguments.
* def add(\*args):  
   for n in args:  
   print(n)  
  add(1, 2, 3, 4, 5, 6)
* \*args : unlimited positional arguments.
* \*\*kwargs : it is unlimited keyword arguments

**Errors and Exception:**

* when error happens the entire code breaks down and nothing after that error line gets executed.
* try : something that might cause exception
* except : do this if there was an exception
* else : do this if no exception
* finally : do this no matter what
* raise : you can raise error manually

**JSON :**

* json.dump() : to write to a json file
* json.load() : to read from json file
* json.update(): to update json data

**API :**

* 1xx = Hold on
* 2xx = Here you go, successful
* 3xx = Go Away
* 4xx = You screwed up (user did something wrong)
* 5xx = I screwed up (server-side problem)
* [HTTP Status Codes — httpstatuses.com](https://httpstatuses.com/)