Python

Indentation in python means it is a block of code.

Comment is indicated with #

Multi line comment within “”” …… “””” block

Python doesn’t have a command to declare variable.

X = ‘hello’

type(x) – gives <class ‘str’>

variable are case sensitive

variable must start with letter or \_ ( no numbers or special char )

multiple variable can be defined in single line

x, y, z = "Orange", "Banana", "Cherry"

multiple variable with same value

x = y = z = “Orange”

upack a collection

fruits = [“orange”,”banana”,”apple”]

x, y, z = fruits

Concat : +

Global variables which are created outside of a function can be used by the function.

Create a global variable by declaring it as global

Data type –

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

Typecast

X = “5”

int(X) -- gives 5

float("3") – 3.0

mutlti line sentence is within three quotes ( double / single. )

a = """Hi my name is techno

I like to program"""

a = '''Hi my name is techno

I like to program'''

Single char access. – print(a[1])

Looping in string –

for x in “banana”:

print(x)

length of a string - len()

check a string -- “substr” in “main str”

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

Not - “substr” in “main str”

b = "Hello, World!" //slice from 2 to 5  
print(b[2:5]) //llo

print(b[:5]) starts from 0 print(b[5:]) starts from 5 till end

From: "o" in "World!" (position -5)

Uppercase – upper() lowercase – lower()

Remove space – strip()

replace()

split(“,”)

Insert number into a string

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

or

myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

If statement

if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

List:

mylist = ["apple", "banana", "cherry"]

Use  list() constructor when creating a new list

thislist = list(("apple", "banana", "cherry"))

if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

thislist[1] = "blackcurrant"

thislist = ["apple", "banana", "cherry"]

thislist[1:2] = ["blackcurrant", "watermelon"]

['apple', 'blackcurrant', 'watermelon', 'cherry']

thislist.insert(2, "watermelon") – inserts as a 3rd element

thislist.append("orange")

thislist.remove("banana")

thislist.pop(1) or del thislist[0]

del thislist. //entire list deleted

thislist.clear() //empties the list

thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)

thislist = ["apple", "banana", "cherry"]  
thistuple = ("kiwi", "orange")  
thislist.extend(thistuple)  
print(thislist)

while i < len(thislist):  
  print(thislist[i])  
  i = i + 1

Comprehension:

newlist = [x for x in fruits if x != "apple"]

ewlist = [x if x != "banana" else "orange" for x in fruits]

thislist.sort()

thislist.sort(reverse = True) //descending

customize sort-

def myfunc(n):  
  return abs(n - 50)  
  
thislist = [100, 50, 65, 82, 23]  
thislist.sort(key = myfunc)  
print(thislist)

thislist.reverse() // reverses the sort order

mylist = thislist.copy() //copy a list

or

mylist = list(thislist)

joining 2 lists – append(), extend() or just + list1 + list2

Tuple - A tuple is a collection which is ordered ( has a specific order )and **unchangeable ( item cant be added or deleted)**.

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

create a tuple with only one item, you must add a comma after the item.

Tuple can have different data types.

Can use tuple() constructor to make a tuple

You are allowed to add tuples to tuples. If you need to change the tuple first convert it list and then edit and convert it back to tuple

thistuple = ("apple", "banana", "cherry")  
y = ("orange",)  
thistuple += y  
  
print(thistuple)

unpack a tuple –

fruits = ("apple", "banana", "cherry")  
  
(green, yellow, red) = fruits  
  
print(green)  
print(yellow)  
print(red)

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")  
  
(green, yellow, \*red) = fruits

creates a list called red with value [cherry", "strawberry", "raspberry"]

Add two tuples tuple3 = tuple1 + tuple2

Multiply a tuple tuple1\*2

Set – they are unordered and unchangeable and cannot have duplicates.

Sets are used to store multiple items in a single variable.

thisset = {"apple", "banana", "cherry"}

Use set() constructor to make a set.

Add an item to a set using the add().

Add any other iterable item ( like set, list, tuple…)

thisset = {"apple", "banana", "cherry"}  
mylist = ["kiwi", "orange"]  
  
thisset.update(mylist)  
  
print(thisset)

remove(), discard(), pop(), clear() and del for deletion of set

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

x.intersection\_update(y)

z = x.intersection(y) // present in both sets

x.symmetric\_difference\_update(y)

z = x.symmetric\_difference(y) //not present in both the sets

Dictionaries: Key Value pair , cannot have duplicate keys

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

print(thisdict["brand"])

x = thisdict.get("model")

x = thisdict.keys(). //list of keys

x = thisdict.values() // list of values

x = thisdict.items(). // list of key:value pair

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

thisdict.update({"year": 2020}) //update

thisdict["color"] = "red" //Add

thisdict.pop("model") // removes item model

thisdict.popitem() // removes last item

Can have nested dictionaries

If –

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

if a > b: print("a is greater than b")

short hand : print("A") if a > b else print("B")

Functions:

def MyFunction():

print(‘Inside function’)

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil", "Refsnes")

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

Can have recursive function

Lambda :

A lambda function is a small anonymous function.

lambda arguments: expression

x = lambda a : a + 10  
print(x(5))

Classes/Objects:

class Person:  
  def \_\_init\_\_(self, name, age): // executed when class is initiated  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36) // object creation  
  
print(p1.name)  
print(p1.age)

Inheritance:

Parent class –

class Person:  
  def \_\_init\_\_(self, fname, lname):  
    self.firstname = fname  
    self.lastname = lname  
  
  def printname(self):  
    print(self.firstname, self.lastname)  
  
#Use the Person class to create an object, and then execute the printname method:  
  
x = Person("John", "Doe")  
x.printname()

child class:

class Student(Person):

pass

x = Student("Mike", "Olsen")  
x.printname()

Child’s \_\_init\_\_() overrides parents \_\_init\_\_() function.

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)

Inherits parents class method and properties.

Iterator:

mytuple = ("apple", "banana", "cherry")  
myit = iter(mytuple)  
  
print(next(myit))  
print(next(myit))  
print(next(myit))

Modules:

A file containing a set of functions you want to include in your application. It can be done using the command import

import mymodule

import mymodule as mx

Date:

import datetime  
  
x = datetime.datetime.now()  
  
print(x.year)  
print(x.strftime("%A"))

JSON:

import json  
  
# some JSON:  
x =  '{ "name":"John", "age":30, "city":"New York"}'  
  
# parse x:  
y = json.loads(x)  
  
# the result is a Python dictionary:  
print(y["age"])

import json  
  
# a Python object (dict):  
x = {  
  "name": "John",  
  "age": 30,  
  "city": "New York"  
}  
  
# convert into JSON:  
y = json.dumps(x)  
  
# the result is a JSON string:  
print(y)

Try … Except ( Error handling )

try:  
  print(x)  
except:  
  print("An exception occurred")

try:  
  print(x)  
except:  
  print("Something went wrong")  
finally:  
  print("The 'try except' is finished")

Input :

username = raw\_input("Enter username:")  
print("Username is: " + username)

Open a file:

f = read(“myfile.txt”,”r”)

f = open("demofile.txt", "r")  
print(f.read()) # reads the content of the file and prints it

print(f.read(**5**)) # returns only first 5 character from the file

print(f.readline()) # gives first line and if you print readline() agaian it gives the 2nd line

f.close() #closes the file

f = open("demofile2.txt", "a") //open file in append mode  
f.write("Now the file has more content!")  
f.close()

f = open("demofile3.txt", "w") //write mode it overwrites the file  
f.write("Woops! I have deleted the content!")

f = open("myfile.txt", "x") // creates a new file

import os  
os.remove("demofile.txt") //delete file

os.rmdir(“myfolder”) //delete complete folder

os.path.exists(“demofile.txt”) //checks if the file exists

Python MySQL

import mysql.connector

mydb = mysql.connector.connect(  
  host="localhost",  
  user="*yourusername*",  
  password="*yourpassword*"  
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = ("John", "Highway 21")  
mycursor.execute(sql, val)  
 **mydb.commit()**  
print(mycursor.rowcount, "record inserted.")

mycursor = mydb.cursor()  
  
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"  
val = [  
  ('Peter', 'Lowstreet 4'),  
  ('Amy', 'Apple st 652'),  
  ('Hannah', 'Mountain 21'),  
  ('Michael', 'Valley 345'),  
  ('Sandy', 'Ocean blvd 2'),  
  ('Betty', 'Green Grass 1'),  
  ('Richard', 'Sky st 331'),  
  ('Susan', 'One way 98'),  
  ('Vicky', 'Yellow Garden 2'),  
  ('Ben', 'Park Lane 38'),  
  ('William', 'Central st 954'),  
  ('Chuck', 'Main Road 989'),  
  ('Viola', 'Sideway 1633')  
]  
  
mycursor.executemany(sql, val)  
  
mydb.commit()

myresult = mycursor.fetchall()  
  
for x in myresult:  
  print(x)

myresult = mycursor.fetchone() // fetches first record