**Conditionals Logic**

is\_old=True

is\_licensed = True

if is\_old:

do something

elif is\_licensed:

do something

else:

do something

* We use colon (:) to start a condition.
* After indentation all code us belongs to conditional.
* Conditional work on True and False.
* if is\_old: (if this is True it will run the code underneath it with indentation)
* If the conditions are not True in if and elif it will the code underneath the else.
* We can use “and” and “or” keywords in conditional
* if have two condition we can write “and” between the conditions. If all conditions are true run its code.
* if have two condition we can write “or” between the conditions. If one condition is true run its code.

if is\_old and is\_license:

print("You are old enough to drive and you have license")

else:

print("You are not of age")

**Indentation**

* It is not for styling in python.
* Indentation use for hierarchy of code parent to child.
* Spaces makes changes through program.

**Truthy and Falsy**

* Python can convert strings into boolean if use in conditional it return boolean True or False they are called Truthy and Falsy.
* All values considered Truthy except the following:
  + None
  + 0.0
  + Fraction(0,1)
  + 0j
  + [] empty list
  + 0
  + Decimal(0)
  + {} empty dictionary
  + () empty tuple
  + ‘’ empty strings
  + b’’ empty byte
  + set() empty set
  + range(0) empty range
  + obj.\_\_bool\_\_() return false
  + obj.\_\_lenn\_\_() return 0
* The Truthy value will satisfy the check performed by ‘if’ and ‘while’ statements. We use Truthy and Falsy to differentiate bool values True and False.

password=’123’

user\_name=’John’

if password and user\_name:

print(“Register Successfully”)

* Above code means if user given values and strings are not empty the Register.

**Ternary Operator**

* They are call Conditional Expressions.
* These are same as conditional.
* It is a shortcut of ‘if’ conditional.

is\_friend = True

can\_message = ‘message’ if is\_friend else ‘not allowed’

print(can\_message)

**Short Circuiting**

* It will done using ‘or’ keyword
* In ‘or’ if one condition is true it will run it is short circuiting in conditional.

**Logical Operators**

* These works in conditionals
* They return returns True or False.
* < less than
* > greater than
* == comparison
* >= greater than and equal to
* <= less than equal to
* != not equal to
* and
* or
* not() it is a function

**is vs ==**

* Using ==
* True == 1 True
* ‘’ == 1 False
* [] == 1 False
* 10 == 10.0 True
* [] == [] True
* If two different types then check by Truthy and Falsy
* If same type check values by comparison == they are totally same.
* ‘1’ == 1 is false because now given a string value in it
* Using is
* True is 1 False
* ‘’ is 1 False
* [] is 1 False
* 10 is 10.0 False
* is keyword check the place in the memory if value is same it returns true.
* It gives false on data structures even the items in the data structure are same because they are placing in different memory.
* **==** check the values
* **is** check the place in the memory and the exact thing.

**Loops**

* Run line of codes over and over.

for item in [1,2,3]:

print(item)

* It iterates list, tuples, sets, dictionary.
* We can create nested loops.

**Iterable**

* Iterable can be list, dictionary, tuple, set and string.
* It iterates (one by one check each item in the collection).
* In dictionary we have three methods to iterate item.

for item in user.items():

for item in user.values():

for item in user.keys():

* We can do unpacking to print key and values

for key,value in user.items()

**Range Function (range()) in loop**

* It return the sequence of integer.
* print(range(0,100))

for item in range(0,100):

print(item)

* It return 0 to 100 so we can say that how many times loop should run.
* If doesn’t need a variable in loop only to use range we can use \_ .

for \_ in range(0,100):

print(\_)

* If want to reverse range can use range?(10,0,-1)
* range(start,stop,step) but start from higher
* we can create list using range

for \_ in range(2):

print(list(range(0,10)))

**While Loop**

* while loop runs infinite loops.
* It can be dangerous it crash program.
* Sow we use break key word.

i = 0

while i <50:

print(i)

i += 1

# break

else:

print("done work")

* When while complete then else will work.
* If it is a break in while else will not work.
* When use while or for:
  + It depends on code.
  + While loops are flexible.
  + For loops are simpler.
  + While loop we have to create extra variable, use a break.
  + For loop is simple but while loop use when do not know how many times it run.
  + We can use while loop for input.
  + We can use conditional to break.

**Break, Continue, Pass**

* Break will stop the operation and exit the loop or condition.
* Continue take the loop or conditional to the start skipping the code after continue.
* Pass – do nothing it is a placeholder in a loop or conditional. If thinking on some code and you want not to break the program use pass.

**First GUI program**

Display the image below to the right hand side where the 0 is going to be ' ', and the 1 is going to be '\*'. This will reveal an image!

picture = [

[0,0,0,1,0,0,0],

[0,0,1,1,1,0,0],

[0,1,1,1,1,1,0],

[1,1,1,1,1,1,1],

[0,0,0,1,0,0,0],

[0,0,0,1,0,0,0]

]

for image in picture:

for pixel in image:

if pixel:

print('\*', *end*='') # by default print function gives new line but end will define what we want after printing

else:

print(' ',*end*='')

print('')

**Functions**

* Functions are actions perform on computer.

def say\_hello():

print(“hello)

* We have to write function to call it

say\_hello()

* In order to take an action must write a bracket in the end of function name.
* Keep the code dry – Don’t repeat yourself.
* We to define functions before calling it because python interpreter run line by line.

**Arguments and Parameters**

* We can give variable to function when defining it those variables are called parameters like def hi(a,b):
* When variable use in calling function they are called arguments.
* Calling function also known as invoking function.
* These are called positional parameters and arguments. Position matters when calling function.

**Keyword Arguments**

* When calling a function we can write values in function.

say\_hello(name=”Raheel”,age=30)

**Default Parameters**

* The values will place when defining functions.
* If I run another function and forget to give arguments it will take default parameters

def say\_hello(name=”Raheel”,age=30)

print(f’my name is {name} and age is {age}’)

* say\_hello() # when call without arguments it will take default one
* say\_hello(“Raheel 2”,”40”) # it will take given values

**return concept**

def sum(num1,num2):

num1+num2

print(sum(4,5))

* Above code will give None because nothing is printing so we use return keyword to return something.
* So we do:

def sum(num1,num2):

return num1+num2

print(sum(4,5))

* Function either modifies our program or return something.
* Function should do something very well.
* we can create nested functions

def sum(*num1*,*num2*):

def another\_sum(*n1*,*n2*):

return n1+n2

return another\_sum(num1,num2)

total = sum(10,20)

print(total)

* Return keyword automatically exit the function.

**Methods vs Functions**

* Methods has to own by something.
* Functions are built-in and self define.
* We can create methods by Classes

**Docstring**

* Information about functions can write in docstring

def some\_function():

‘’’

Information about function

‘’’’

* We can see the information of function by hovering on the name of function or use help function like help(somefunction()) and dunder method print(somefunction.\_\_doc\_\_)

**Clean Code**

* Break the code.
* Less line functions are good functions.

**args and kwargs**

* \*args it can accept as many arguments just defining one parameter.
* It gives tuples of arguments.
* \*\*kwargs takes as many keyword arguments when calling functions.
* Kwargs returns dictionary.
* Rule of defining them:
* params
* \*args
* default params
* \*\*kwargs

def some\_function(\**args*,\*\**kwargs*):

total = 0

for items in kwargs.values():

total += items

return sum(args) + total

print(some\_function(1,2,3,4,5, *num1* =5,*num2*=10))

**Scope**

* Scope is what variable do have access to
* When variable is not in the scope it give an error.
* Python has function scope.
* Any variable not in function is called global scope.
* Functions have its own scope.
* Scope rules:
* Start with local scope – In function variable are local.
* Is their parent local.
* Check global scope.
* Built-In function

**Global Keyword**

* Parameter consider in local scope as a local variable.
* If create global variable and want use it in function without new creating new one.
* We use global keyword to use global variables in function.
* It is a bad practice to use global variables in function.
* We can use dependency injection.

**Nonlocal**

**# Scope - what variables do I have access to?**

def outer():

x = "local"

def inner():

nonlocal x # it refering parent variable x

x = "nonlocal" # and changing value of parent local so the x value change from local to non local

print("inner:", x)

inner()

print("outer:", x)

outer()

#1 - start with local

#2 - Parent local?

#3 - global

#4 - built in python functions

**Why do we use scope**

It is easy if everything take as global but machine not have infinite power, memory so programmers need to think how to use scope to reduce power and memory of a machine.