```
#!/usr/bin/python3

# syntax.py by Kumar Aakash

# Some Notes :
# Everything IN Python 3 is an object variable functions and even code
# Every object has an ID and a type and a value
# ID is a unique idetifier that identifies a unique object (cannor change for asingle object)
# Type is the CLASS ofthe object (canno change for teh life)
# Value is teh data contained inside teh object.
# Since every thing is an object their can be function calls even using Varible as objects eg var.xmfnc1()

# mutable objects can change their value immutable objects cannot
# variables in python are immutable obects though they might seem to be changinh values
# actually they do not
# Variables in Python are actually refrences to objects and whenver a new value is assigned to a variable
# the refrence is changed to a diffrent address
# If we reasssign the old value of teh variable again the refrence changes to the old refrence again
# thus python vriables are immutable objects.
# Most duncdatmental types in Python are immutable
# ists dictionaries and other objects are mutable

# simple class definition
class Egg:
    # constructor of a class
    # it is usually a good idea to anme an objct variable as
    # _Varname reminds that this is an object variable.
    # Although we can use these variables directly (outside the class)
    # it is always a good idea to write getters and setters for each
    # class variable (private access)
    def __init__(self, kind="fried "):
        self._kind = kind

    # Function definition of a class
    def Whatkind(self):
        return self._kind + "egg"

# Another moe better way of creating object variables is using
# keyword arguements or hashmaps.

class Egg2:
    def __init__(self, **kwargs):
        self.variables = kwargs
        print(self.variables)

    def getVariable(self, varname):
        return self.variables[varname]

    def setVariables(self, varname, value = None):
```

```
50            self.variables[varname] = value;
51
52        def describe(self):
53            print("We are talking about an egg from a ",self.getVariable(
              'type'),
54                  "which is ",self.variables['time'], "days old and is ",
                    self.variables['kind'])
55
56    class Animal:
57        def talk(self):
58            print("Animal Talking")
59        def walk(self):
60            print("Animal Walking")
61
62    class Dog(Animal):
63        #@Override
64        def talk(self):
65            print("Dog Talking")
66        def walk(self):
67            print("Dog Walking")
68
69    class Cat(Animal):
70        #@Override
71        def walk(self):
72            super().walk()
73            print("Cat is walking")
74
75    class Monkey(Animal):
76        pass
77
78    # Polymorphism is something that Python is exceptionally
79    # good at.
80
81    class Duck:
82        def quack(self):
83            print("Dicks Quack")
84
85        def walk(self):
86            print("Ducks can both walk and swim")
87
88    class Hen:
89        def quack(self):
90            print("hens don't quack")
91
92        def walk(self):
93            print("Hens can walk and Fly")
94
95    # though by the name it expects a Hen but both of the hen and
96    # the duck can be supplied without an error to this function
97    # becoz both are implementing teh same interface and have the same
98    # fnctions.
99    def some_func(Hen):
100       print("Here expecting a hen")
101
```

```
102  # Creating Generator Classes:
103  class inclusive_Range:
104      def __init__(self, *args):
105
106          nm = len(*args)
107          if nm < 1:
108              raise TypeError("Atlest one arguement is required")
109          elif nm == 1:
110              self.start = 0
111              self.end = args[0]
112              self.step=1
113          elif nm == 2:
114              (self.start, self.end) = args
115              self.step = 1
116          elif nm == 3:
117              (self.start, self.end, self.step) = args
118          else:
119              raise TypeError ("TOo msny atguements")
120
121      def __itr__(self):
122          i = self.start
123          while(i <= self.end):
124              yield i
125              i += 1
126
127  #--------------------
128
129  # main method
130  def main():
131
132      #Inheritance in Python
133
134      for i in inclusive_range(0, 25, 5):
135          print(i)
136
137      inclusive_range()
138
139      inclusive_range(1,2,3,4,5,6)
140
141      bruno = Dog()
142      mycat = Cat()
143
144      bruno.walk()
145      bruno.talk()
146
147      mycat.walk()
148      mycat.talk()
149
150      pukpuk = Hen()
151      chapchap = Duck()
152
153      # Runtime Polymorphism only teh implementation of same interface is
154      # required to call the same function of any two objects.
155
```

```
156         some_func(pukpuk)
157         some_func(chapchap)
158
159         smEgg2 = Egg2(kind = 'fried', type='Chicken', time=2)
160         smEgg2.describe()
161         # Simple assignment
162         a, b = 0,1
163
164         # For and while Loops
165         while a < b:
166             print("In a while ... ")
167             a += 1
168
169         # For loop introduces an iterator and can work with containers
170         # aggregator
171         fh = open("lines.txt")
172         for i in fh.readlines():
173             print(i)
174
175         # print() adds an extra newline but to prevent that we
176         # can use the function end = ''
177         # For loops can be used with
178         for i in "Somestring":  #strings
179             print(i)
180
181         for i in (1, 2, 3, 4, 5):   #tuple
182             print(i)
183
184         for i in [1, 2, 3, 4, 5]:   #lists
185             print(i)
186
187         # For loops may sometimes need an enumerator or a
188         # couter variable to keep track of what is going on
189         # this feature is also present in python
190         for index, i in enumerate("This is a string"):
191             print(index ,i)
192             if index == 3: print("@ third index ", i)
193             if i == 'a': print("A is found at ", index)
194
195         # Break and Continue in for
196         for i in "This is a String":
197             if i == 's': continue
198             print(i)
199             #if i == 'n': break
200         else:
201             print("Action performaed af ter the loop is \n"
202                   "done ... only executed if the loop condition fails \n"
203                   "can be done by commenting out the break above \n")
204
205         # ranges are non inclusive the last item is not included
206         for i in range(0, 10): print(i)
207
208         # Divmod Function gives the result of both divison and modulo
209         # in the same operation
```

```
210
211         # SLICE operations
212         print ("Slice Operation")
213         l = [1 , 2,  3,  4,  5,  6,  7,  8,  9,  0,
214              11, 22, 33, 44, 55, 66, 77, 88, 99, 00,
215              111,222,333,444,555,666,777,888,999,000]
216
217         print(l[0:10])  # elements from 0 to 10
218         print(l[0:10:3]) # every third elemnts from 0 to 10
219         print(l[5:10])  # elements from 5 to 10
220         l[1:3] = [99 , 99, 99]
221         print (l)
222
223         # Regular Expressions in Python
224         import re
225         print("====REGEXES====\n")
226         rfh = open("raven.txt")
227         for line in rfh:
228             print(re.sub('len[a-z]{,}', "###", line))
229
230         # Can also be done with
231         # note that a new file handle is needed because
232         # teh previous one already reached teh EOF
233
234         print("******PART 2******")
235
236         pattern = re.compile('(len|Neverm)ore', re.IGNORECASE)
237         rfh2 = open("raven.txt")
238         for line1 in rfh2:
239             match1 = re.search(pattern, line1)
240             if match1:
241                 print(line1.replace(match1.group(), '###'))
242                 # or
243                 print(pattern.sub('###', line), end='')
244         print("========")
245
246         # Data Types
247         # there are two different types of numbers in Python
248         # integers and floats
249         x = 34.001
250         y = 34 / 5
251         z = 34 // 5   # ignores the deciaml part
252         z1 = round(34 / 5, 3)  # rounds teh result to given preision
253         print("A is a ", type(a), a)
254         print("X is a ", type(x), x)
255         print("Y is a ", type(y), y)
256         print("Z is a ", type(z), z)
257         print("Z1 is a ", type(z1), z1)
258
259         # Typecast in Python
260         m = int(2.334343) # Constructor for int clasa parameter passed is
            2.33
261         n = float(23)
262         print("m is a ", type(m), m)
```

```
263         print("n is a ", type(n), n)
264
265         # Strings are one of teh most strong features of Python
266         # Both double and simgle quotes strings are allowed
267         s = "This is a test string"
268         print(s)
269         s1 = "This is a string in \n 2 lines"
270         print(s1)
271         s2 = r"This is a raw string in \n 2 lines"
272         print(s2)
273         s3 = "This is a string with {} number inseted".format(n)
274         print(s3)
275         s4 = '''\
276         this is another way of
277         describbing a string where we can describe
278         a string line after line after
279         line it is a really great way of writing'''
280         print(s4)
281
282         # Tuples and Lists
283         p = {0, 1, 2} # a tuple is immutable cannot append or insert
284         print(type(p), p)
285
286         q = [{0,1}, {2,3}, {4, 5}] # a list of tuples
287         print(type(q), q)
288         q.append(5)
289         print("After appending", type(q), q)
290         q.insert(2, 10) # inserting 10 at location 2
291         print("After inserting", type(q), q)
292         # We can see the individual elements of a sequential type by doing
293         print("List Element", q[1])
294
295         # string is also a sequential type of data
296         str = 'qwerty'
297         print(str[1])
298         print(str[1:4])  # this is called slicing
299
300         # using the sequential data types as a loop works
301         # for tuple and lists too
302         for i in str:
303             print(i)
304
305         # Another aggregator type is called a DICTIONARY and is pretty
            similar
306         # to HASH in other languages like JAVA and C++
307         di = {1:"one" , 2:"two" , 3:"three", 4:"four", 5:"five"}
308         print(di)
309
310         # traversing a dictionary
311         for k in di:
312             print(k , di[k])
313         # notice that the print is in no
314         # particular order therefor to sort it according to keys
315         # we can have the sorted method called on the key object of
```

```
316         # dict
317 
318         # sorting by Keys
319         for k in sorted(di.keys()):
320             print(k, di[k])
321 
322         # sorting by Values
323         for k in sorted(di.values()):
324             print(k)
325 
326         # also support multiple kinds
327         d2 = dict(
328             one = 1 , two = 2 , three = 3
329             )
330 
331         d2['six'] = "Six"
332         for k in sorted(d2.keys()):
333             print(k, d2[k])
334 
335         # the "is" operator is used for comparing instances but = is
336         # used for comparing values ... if two variales (more like pointers)
337         # point to same location they can be compared using x is y
338         # they are immutable objects
339         # Mutable objects such as list and dict cannot be compared using
340         # IS operator.
341 
342         # True and False are immutable objects of the Class Bool
343 
344         # A conditional statement similar to swith case in Python is not
345         # present but that essentially is not a weakness in the language
346         # rather it is a different outlook of looking at things
347 
348         choices = dict(
349             one = "One",
350             two = 2,
351             three = "three",
352             four = 4
353         )
354         # So now instead of needing a special conruct we can
355         # easily select the value based on the key using the
356         # dictionary
357         print("=====")
358         print(choices['one'])
359         print(choices['two'])
360         print(choices['three'])
361         print(choices['four'])
362 
363         # However this might lead to an error if something that is not
364         # in the dict is searched, to overcome this problem a get method
365         # teh dictionary is provided
366         # will print Other as Nine is not present in key
367         print(choices.get('nine', 'other'))
368         print("=====")
369 
```

```
370         # creating object
371         smegg1 = Egg()
372         print("Kind Egg1 = {}".format(smegg1.Whatkind()))
373         smegg2 = Egg("scrambled")
374         print("Kind Egg2 = {}".format(smegg2.Whatkind()))
375
376         # multiple assignment
377         a, b = b, a     # Swap is simple
378         print("A = {} B = {}".format(a, b))
379
380         # conditional block
381         if a < b:
382             print("A is less")
383         elif a > b:
384             print("A is greater")
385         else:
386             print("Both are equal")
387
388         # conditional expression/value
389         s = "a less" if (a < b) else "a not less"
390         print(s)
391         print("This is the syntax.py file.")
392
393         # Function Calls
394         # takes default arguement
395         func()
396         # overwrites the default arguement
397         func(3)
398         # Passing 3 values to func with 4 args
399         func_none(1, 2, 3)
400         # Calling function with unknown number
401         # of arguements
402         func_ua(1,2,3,4,5,6,7,8,9,0)
403         # Functions can also be called using named (key value) pairs
404         func_kwa(one=1 , two=2, three=3)
405
406         # Exceptions Handling in Python
407         # Try Catch Else
408         try:
409             nfh = open("xlines.txt")
410         except IOError as e:
411             print("Exception Occured While Opoening teh File to read")
412             print("Error : ", e)
413         else:
414             for i in nfh.readlines():
415                 print(i)
416
417         # raising your own exceptions
418         try:
419             for i in readfile("files.doc"):
420                 print(i)
421         except IOError as ex:
422                 print("IO Exception Raised", ex)
423         except ValueError as err:
```

```
424                 print("Error : ", err)
425
426         # A generator function is any function that returns an iterator
            object
427         # inclusive_range() here returns an iterator object
428         for i in inclusive_range(25):
429             print(" ", i ,end='')
430
431     # ---------------------
432
433     # in iterator function similar to range but inclusive
434     def inclusive_range(*args):
435         if len(args) < 1:
436             raise TypeError("inclusive_range requires atleast one arguement")
437         elif len(args) == 1:
438             start = 0
439             end = args[0]
440             step = 1
441         elif len(args) == 2:
442             (start, end) = args
443             step = 1
444         elif len(args) == 3:
445             (start, end, step) = args
446         else:
447             raise TypeError("inclusive range can have maximum of three
                arguements")
448
449         i = start
450         while(i <= end):
451             yield i  # returns i but keeps execution inside the function body
452             i += step
453
454     # functio nreceiving key value pairs essentially dictionaries
455     # also known as key word arguements
456     def func_kwa(**kwargs):
457         print(kwargs)
458         print(kwargs['one'])
459         print(kwargs['two'])
460         print(kwargs['three'])
461
462     # All kinds of arguements can also be mixed barring one small restriction
463     # teh order should be Number Args -> Tuple Args -> Dict Args
464
465     # Function with unknown number of arguements
466     def func_ua(a1, a2, a3, *args):
467         print(a1);
468         print(a2);
469         print(a3);
470         print(*args);
471
472     # functions which have x number of args
473     # can be called with y number of arguements
474     # if x-y arguements are asssigned to None
475
```

```
476   def func_none(a1, a2, a3, a4 = None):
477       print(a1);
478       print(a2);
479       print(a3);
480       print(a4);
481
482   # function which does not have any lines in it
483   def test_pass():
484       pass   # essentially a NO Operation statement
485              # makes ur fnction syntactically correct.
486
487   # returns the filehandle iterator
488   def readfile(filename):
489       if(filename.endswith(".txt")):
490           smfh = open(filename)
491           return smfh.readlines()
492       else:
493           raise ValueError("Filename must end in a txt")
494
495   # example for function definition
496   def func(a = 7):
497       print("We are n func")
498       for i in range(a, 10):
499           print(i)
500       print("Leving the func")
501
502   # allows us to run the script in any order that we want and
503   # does not make it manadatory to define a function before use.
504   # or evn a definitio nf a main function
505   if __name__ == "__main__": main()
506   # or
507   # main() would also have th e same effect the above code is generall
508   # useful in modules where it specifies which function is going to
509   # execute for this module
510
```