

```

1  # import is used to make specialty functions available
2  # These are called modules
3  import random
4  import sys
5  import os
6
7  # Hello world is just one line of code
8  # print() outputs data to the screen
9  print("Hello World")
10
11  '''
12  This is a multi-line comment
13  '''
14
15  # A variable is a place to store values
16  # Its name is like a label for that value
17  name = "Derek"
18  print(name)
19
20  # A variable name can contain letters, numbers, or _
21  # but can't start with a number
22
23  # There are 5 data types Numbers, Strings, List, Tuple, Dictionary
24  # You can store any of them in the same variable
25
26  name = 15
27  print(name)
28
29  # The arithmetic operators +, -, *, /, %, **, //
30  # ** Exponential calculation
31  # // Floor Division
32  print("5 + 2 =", 5+2)
33  print("5 - 2 =", 5-2)
34  print("5 * 2 =", 5*2)
35  print("5 / 2 =", 5/2)
36  print("5 % 2 =", 5%2)
37  print("5 ** 2 =", 5**2)
38  print("5 // 2 =", 5//2)
39
40  # Order of Operation states * and / is performed before + and -
41
42  print("1 + 2 - 3 * 2 =", 1 + 2 - 3 * 2)
43  print("(1 + 2 - 3) * 2 =", (1 + 2 - 3) * 2)
44
45  # A string is a string of characters surrounded by " or '
46  # If you must use a " or ' between the same quote escape it with \
47  quote = "\"Always remember your unique,\""
48
49  # A multi-line quote
50  multi_line_quote = ''' just
51  like everyone else'''
52
53  print(quote + multi_line_quote)
54
55  # To embed a string in output use %s
56  print("%s %s %s" % ('I like the quote', quote, multi_line_quote))
57
58  # To keep from printing newlines use end=""
59  print("I don't like ",end="")
60  print("newlines")
61
62  # You can print a string multiple times with *
63  print('\n' * 5)
64
65  # LISTS -----
66  # A list allows you to create a list of values and manipulate them
67  # Each value has an index with the first one starting at 0
68
69  grocery_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']

```

```
70 print('The first item is', grocery_list[1])
71
72 # You can change the value stored in a list box
73 grocery_list[0] = "Green Juice"
74 print(grocery_list)
75
76 # You can get a subset of the list with [min:up to but not including max]
77
78 print(grocery_list[1:3])
79
80 # You can put any data type in a a list including a list
81 other_events = ['Wash Car', 'Pick up Kids', 'Cash Check']
82 to_do_list = [other_events, grocery_list]
83
84 print(to_do_list)
85
86 # Get the second item in the second list (Boxes inside of boxes)
87 print(to_do_list[1][1])
88
89 # You add values using append
90 grocery_list.append('onions')
91 print(to_do_list)
92
93 # Insert item at given index
94 grocery_list.insert(1, "Pickle")
95
96 # Remove item from list
97 grocery_list.remove("Pickle")
98
99 # Sorts items in list
100 grocery_list.sort()
101
102 # Reverse sort items in list
103 grocery_list.reverse()
104
105 # del deletes an item at specified index
106 del grocery_list[4]
107 print(to_do_list)
108
109 # We can combine lists with a +
110 to_do_list = other_events + grocery_list
111 print(to_do_list)
112
113 # Get length of list
114 print(len(to_do_list))
115
116 # Get the max item in list
117 print(max(to_do_list))
118
119 # Get the minimum item in list
120 print(min(to_do_list))
121
122 # TUPLES -----
123 # Values in a tuple can't change like lists
124
125 pi_tuple = (3, 1, 4, 1, 5, 9)
126
127 # Convert tuple into a list
128 new_tuple = list(pi_tuple)
129
130 # Convert a list into a tuple
131 # new_list = tuple(grocery_list)
132
133 # tuples also have len(tuple), min(tuple) and max(tuple)
134
135 # DICTIONARY or MAP -----
136 # Made up of values with a unique key for each value
137 # Similar to lists, but you can't join dicts with a +
138
```

```
139 super_villains = {'Fiddler' : 'Isaac Bowin',
140                   'Captain Cold' : 'Leonard Snart',
141                   'Weather Wizard' : 'Mark Mardon',
142                   'Mirror Master' : 'Sam Scudder',
143                   'Pied Piper' : 'Thomas Peterson'}
144
145 print(super_villains['Captain Cold'])
146
147 # Delete an entry
148 del super_villains['Fiddler']
149 print(super_villains)
150
151 # Replace a value
152 super_villains['Pied Piper'] = 'Hartley Rathaway'
153
154 # Print the number of items in the dictionary
155 print(len(super_villains))
156
157 # Get the value for the passed key
158 print(super_villains.get("Pied Piper"))
159
160 # Get a list of dictionary keys
161 print(super_villains.keys())
162
163 # Get a list of dictionary values
164 print(super_villains.values())
165
166 # CONDITIONALS -----
167 # The if, else and elif statements are used to perform different
168 # actions based off of conditions
169 # Comparison Operators : ==, !=, >, <, >=, <=
170
171 # The if statement will execute code if a condition is met
172 # White space is used to group blocks of code in Python
173 # Use the same number of proceeding spaces for blocks of code
174
175 age = 30
176 if age > 16 :
177     print('You are old enough to drive')
178
179 # Use an if statement if you want to execute different code regardless
180 # of whether the condition ws met or not
181
182 if age > 16 :
183     print('You are old enough to drive')
184 else :
185     print('You are not old enough to drive')
186
187 # If you want to check for multiple conditions use elif
188 # If the first matches it won't check other conditions that follow
189
190 if age >= 21 :
191     print('You are old enough to drive a tractor trailer')
192 elif age >= 16:
193     print('You are old enough to drive a car')
194 else :
195     print('You are not old enough to drive')
196
197 # You can combine conditions with logical operators
198 # Logical Operators : and, or, not
199
200 if ((age >= 1) and (age <= 18)):
201     print("You get a birthday party")
202 elif (age == 21) or (age >= 65):
203     print("You get a birthday party")
204 elif not(age == 30):
205     print("You don't get a birthday party")
206 else:
207     print("You get a birthday party yeah")
```

```

208
209 # FOR LOOPS -----
210 # Allows you to perform an action a set number of times
211 # Range performs the action 10 times 0 - 9
212 for x in range(0, 10):
213     print(x , ' ', end="")
214
215 print('\n')
216
217 # You can use for loops to cycle through a list
218 grocery_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']
219
220 for y in grocery_list:
221     print(y)
222
223 # You can also define a list of numbers to cycle through
224 for x in [2,4,6,8,10]:
225     print(x)
226
227 # You can double up for loops to cycle through lists
228 num_list = [[1,2,3],[10,20,30],[100,200,300]];
229
230 for x in range(0,3):
231     for y in range(0,3):
232         print(num_list[x][y])
233
234 # WHILE LOOPS -----
235 # While loops are used when you don't know ahead of time how many
236 # times you'll have to loop
237 random_num = random.randrange(0,100)
238
239 while (random_num != 15):
240     print(random_num)
241     random_num = random.randrange(0,100)
242
243 # An iterator for a while loop is defined before the loop
244 i = 0;
245 while (i <= 20):
246     if(i%2 == 0):
247         print(i)
248     elif(i == 9):
249         # Forces the loop to end all together
250         break
251     else:
252         # Shorthand for i = i + 1
253         i += 1
254         # Skips to the next iteration of the loop
255         continue
256
257     i += 1
258
259 # FUNCTIONS -----
260 # Functions allow you to reuse and write readable code
261 # Type def (define), function name and parameters it receives
262 # return is used to return something to the caller of the function
263 def addNumbers(fNum, sNum):
264     sumNum = fNum + sNum
265     return sumNum
266
267 print(addNumbers(1, 4))
268
269 # Can't get the value of rNum because it was created in a function
270 # It is said to be out of scope
271 # print(sumNum)
272
273 # If you define a variable outside of the function it works every place
274 newNum = 0;
275 def subNumbers(fNum, sNum):
276     newNum = fNum - sNum

```

```
277     return newNum
278
279 print(subNumbers(1, 4))
280
281 # USER INPUT -----
282 print('What is your name?')
283
284 # Stores everything typed up until ENTER
285 name = sys.stdin.readline()
286
287 print('Hello', name)
288
289 # STRINGS -----
290 # A string is a series of characters surrounded by ' or "
291 long_string = "I'll catch you if you fall - The Floor"
292
293 # Retrieve the first 4 characters
294 print(long_string[0:4])
295
296 # Get the last 5 characters
297 print(long_string[-5:])
298
299 # Everything up to the last 5 characters
300 print(long_string[:-5])
301
302 # Concatenate part of a string to another
303 print(long_string[:4] + " be there")
304
305 # String formatting
306 print("%c is my %s letter and my number %d number is %.5f" % ('X', 'favorite', 1, .14))
307
308 # Capitalizes the first letter
309 print(long_string.capitalize())
310
311 # Returns the index of the start of the string
312 # case sensitive
313 print(long_string.find("Floor"))
314
315 # Returns true if all characters are letters ' isn't a letter
316 print(long_string.isalpha())
317
318 # Returns true if all characters are numbers
319 print(long_string.isalnum())
320
321 # Returns the string length
322 print(len(long_string))
323
324 # Replace the first word with the second (Add a number to replace more)
325 print(long_string.replace("Floor", "Ground"))
326
327 # Remove white space from front and end
328 print(long_string.strip())
329
330 # Split a string into a list based on the delimiter you provide
331 quote_list = long_string.split(" ")
332 print(quote_list)
333
334 # FILE I/O -----
335
336 # Overwrite or create a file for writing
337 test_file = open("test.txt", "wb")
338
339 # Get the file mode used
340 print(test_file.mode)
341
342 # Get the files name
343 print(test_file.name)
344
345 # Write text to a file with a newline
```

```

346 test_file.write(bytes("Write me to the file\n", 'UTF-8'))
347
348 # Close the file
349 test_file.close()
350
351 # Opens a file for reading and writing
352 test_file = open("test.txt", "r+")
353
354 # Read text from the file
355 text_in_file = test_file.read()
356
357 print(text_in_file)
358
359 # Delete the file
360 os.remove("test.txt")
361
362 # CLASSES AND OBJECTS -----
363 # The concept of OOP allows us to model real world things using code
364 # Every object has attributes (color, height, weight) which are object variables
365 # Every object has abilities (walk, talk, eat) which are object functions
366
367 class Animal:
368     # None signifies the lack of a value
369     # You can make a variable private by starting it with __
370     __name = None
371     __height = None
372     __weight = None
373     __sound = None
374
375     # The constructor is called to set up or initialize an object
376     # self allows an object to refer to itself inside of the class
377     def __init__(self, name, height, weight, sound):
378         self.__name = name
379         self.__height = height
380         self.__weight = weight
381         self.__sound = sound
382
383     def set_name(self, name):
384         self.__name = name
385
386     def set_height(self, height):
387         self.__height = height
388
389     def set_weight(self, height):
390         self.__height = height
391
392     def set_sound(self, sound):
393         self.__sound = sound
394
395     def get_name(self):
396         return self.__name
397
398     def get_height(self):
399         return str(self.__height)
400
401     def get_weight(self):
402         return str(self.__weight)
403
404     def get_sound(self):
405         return self.__sound
406
407     def get_type(self):
408         print("Animal")
409
410     def toString(self):
411         return "{} is {} cm tall and {} kilograms and says {}".format(self.__name, self.__height, self.__weight, self.__sound)
412
413 # How to create a Animal object
414 cat = Animal('Whiskers', 33, 10, 'Meow')

```

```
415
416 print(cat.toString())
417
418 # You can't access this value directly because it is private
419 #print(cat.__name)
420
421 # INHERITANCE -----
422 # You can inherit all of the variables and methods from another class
423
424 class Dog(Animal):
425     __owner = None
426
427     def __init__(self, name, height, weight, sound, owner):
428         self.__owner = owner
429         self.__animal_type = None
430
431         # How to call the super class constructor
432         super(Dog, self).__init__(name, height, weight, sound)
433
434     def set_owner(self, owner):
435         self.__owner = owner
436
437     def get_owner(self):
438         return self.__owner
439
440     def get_type(self):
441         print ("Dog")
442
443     # We can overwrite functions in the super class
444     def toString(self):
445         return "{} is {} cm tall and {} kilograms and says {}. His owner is {}".format(self.ge
446
447     # You don't have to require attributes to be sent
448     # This allows for method overloading
449     def multiple_sounds(self, how_many=None):
450         if how_many is None:
451             print(self.get_sound)
452         else:
453             print(self.get_sound() * how_many)
454
455 spot = Dog("Spot", 53, 27, "Ruff", "Derek")
456
457 print(spot.toString())
458
459 # Polymorphism allows use to refer to objects as their super class
460 # and the correct functions are called automatically
461
462 class AnimalTesting:
463     def get_type(self, animal):
464         animal.get_type()
465
466 test_animals = AnimalTesting()
467
468 test_animals.get_type(cat)
469 test_animals.get_type(spot)
470
471 spot.multiple_sounds(4)
```