Exercise 1 - Strings

Given the code below, insert the correct negative index on line 3 in order to get the last character in the string.

Exercise 1 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[-1])
```

Exercise 2 - Strings

Given the code below, insert the correct positive index on line 3 in order to return the comma character from the string.

Exercise 2 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[7])
```

Exercise 3 - Strings

Given the code below, insert the correct negative index on line 3 in order to return the **w** character from the string.

Exercise 3 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[-10])
```

Exercise 4 - Strings

Given the code below, insert the correct method on line 3 in order to return the index of the **B** character in the string.

Exercise 4 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.index("B"))
```

Exercise 5 - Strings

Given the code below, insert the correct method on line 3 in order to return the number of occurrences of the letter **o** in the string.

Exercise 5 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.count("o"))
```

Exercise 6 - Strings

Given the code below, insert the correct method on line 3 in order to convert all letters in the string to uppercase.

Exercise 6 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.upper())
```

Exercise 7 - Strings

Given the code below, insert the correct method on line 3 in order to get the index at which the substring **Bitcoin** starts.

Exercise 7 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.find("Bitcoin"))
```

Exercise 8 - Strings

Given the code below, insert the correct method on line 3 in order to check of the string starts with the letter **X**.

Exercise 8 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.startswith("X"))
```

Exercise 9 - Strings

Given the code below, insert the correct method on line 3 in order to convert all uppercase letters to lowercase and all lowercase letters to uppercase.

Exercise 9 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.swapcase())
```

Exercise 10 - Strings

Given the code below, insert the correct method on line 3 in order to remove all spaces (single Space characters from the keyboard) from the string.

Exercise 10 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.replace(" ", ""))
```

Exercise 11 - Strings

Given the code below, insert the correct method on line 3 in order to replace all the occurrences of letter **i** with the substring **btc**.

Exercise 11 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.replace("i", "btc"))
```

Exercise 12 - Strings

Given the code below, insert the correct method on line 3 in order to split the entire string in two parts, using the comma as a delimiter.

Exercise 12 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.split(","))
```

Exercise 13 - Strings

Given the code below, insert the correct method on line 3 in order to join the characters of the string using the **&** symbol as a delimiter.

Exercise 13 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print("&".join(my_string))
```

Exercise 14 - Strings

Given the code below, insert the correct code on line 4 in order to concatenate **my_string** with the following string:

```
my_other_string = "Poor guy!"
```

Exercise 14 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2  my_other_string = "Poor guy!"
3
4  print(my_string + my_other_string)
```

Exercise 15 - Strings

Given the code below, insert the correct method on line 3 in order to convert the first letter of each word in the string to uppercase.

Exercise 15 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string.title())
```

Exercise 16 - Strings

Given the code below, use string formatting with the **%** operator on line 3 to map the values of **2010**, **10k** and **Bitcoin** to the corresponding formatting operators.

Exercise 16 - Solution

```
1 my_string = "In %s, someone paid %s %s for two pizzas."
2
3 print(my_string % ("2010", "10k", "Bitcoin"))
```

Exercise 17 - Strings

Given the code below, use string formatting with the **format()** method on line 3 to map the values of **2010**, **10k** and **Bitcoin** to the corresponding formatting operators.

Exercise 17 - Solution

```
1 my_string = "In {}, someone paid {} {} for two pizzas."
2
3 print(my_string.format("2010", "10k", "Bitcoin"))
```

Exercise 18 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the substring **2010** from the string. Use positive indexes!

Exercise 18 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[3:7])
```

Exercise 19 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the substring **Bitcoin** from the string. Use negative indexes!

Exercise 19 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string[-23:-16])
```

Exercise 20 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the first 12 characters in the string. Use a single, positive index!

Exercise 20 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string[:12])
```

Exercise 21 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the last 9 characters of the string. Use a single, negative index!

Exercise 21 - Solution

```
1  my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3  print(my_string[-9:])
```

Instructions

Exercise 22 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the entire string in reversed order.

exercise.py

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[::-1])
```

Exercise 23 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return every 7th character of the string, starting with the first character.

The final result should be `I,n top`

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[0::7])
```

Exercise 23 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[::7])
```

Exercise 24 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the string except the first 10 characters. Use a single, positive index!

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[10::])
```

Exercise 24 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[10:])
```

Exercise 25 - Strings

Given the code below, use slicing and insert the correct code on line 3 in order to return the string except the last 4 characters. Use a single, negative index!

Exercise 25 - Solution

```
1 my_string = "In 2010, someone paid 10k Bitcoin for two pizzas."
2
3 print(my_string[:-4])
```

Exercise 26 - Numbers & Booleans

Given the code below, use the correct function on line 3 in order to convert **num1** from integer to float.

```
1  num1 = 25
2
3  num2 = float(num1)
4
5  print(type(num2))
```

Exercise 27 - Numbers & Booleans

Given the code below, use the correct function on line 3 in order to convert **num1** from float to integer.

```
1  num1 = 13.67
2
3  num2 = int(num1)
4
5  print(type(num2))
```

Exercise 28 - Numbers & Booleans

Given the code below, use the correct math operator in between **num1** and **num2** on line 4 in order to obtain the result shown.

```
1  num1 = 25
2  num2 = 8
3
4  num3 = num1 % num2
5
6  print(num3 == 1) #result is True
```

Exercise 29 - Numbers & Booleans

Given the code below, use the correct math operator in between **num1** and **num2** on line 4 in order to obtain the result shown.

```
1  num1 = 10
2  num2 = 3
3
4  num3 = num1 num2
5
6  print(num3 == 250 * 4) #result is True
```

Exercise 29 - Solution

```
1  num1 = 10
2  num2 = 3
3
4  num3 = num1 ** num2
5
6  print(num3 == 250 * 4) #result is True
```

Exercise 30 - Numbers & Booleans

Given the code below, use the correct math operator in between **num1** and **num2** on line 4 in order to obtain the result shown.

```
1  num1 = 5
2  num2 = 2
3
4  num3 = num1 // num2
5
6  print(num3 == 5 % 3) #result is True
```

Exercise 31 - Numbers & Booleans

Given the code below, use the correct function on line 3 in order to obtain the distance between **num1** and 0.

```
1  num1 = -11
2
3  num2 =
4
5  print(num2 == 11) #should result in True
```

Exercise 31 - Solution

```
1  num1 = -11
2
3  num2 = abs(num1)
4
5  print(num2 == 11) #should result in True
```

Exercise 32 - Numbers & Booleans

Given the code below, use the correct function on line 4 in order to raise **num1** to the power of **num2**.

```
1  num1 = 10
2  num2 = 5
3
4  num3 = num1 ** num2
5
6  print(num3 == 100000)
```

Exercise 32 - Solution

```
1  num1 = 10
2  num2 = 5
3
4  num3 = pow(num1, num2)
5
6  print(num3 == 100000)
```

Exercise 33 - Numbers & Booleans

Given the code below, use the correct logical operator in between the two expressions on line 1 in order for the final result to be **False**.

```
1  result = ((25 % 7 + 10 / 2) % 3 == 0)  == ((abs(-19) / 2 - 2) > 9)
2
3  print(result) #should return False
```

Exercise 33 - Solution

```
1  result = ((25 % 7 + 10 / 2) % 3 == 0) and ((abs(-19) / 2 - 2) > 9)
2
3  print(result) #should return False
```

Exercise 34 - Numbers & Booleans

Given the code below, use the correct logical operator in between the two expressions on line 1 in order for the final result to be **True**.

```
1  result = (min(pow(2, abs(3)), 9) == 3 ** 2 - 1) or (66 % 20 + 2 > 2 ** 3)
2
3  print(result) #should return True
```

Exercise 34 - Solution

```
1  result = (min(pow(2, abs(3)), 9) == 3 ** 2 - 1) or (66 % 20 + 2 > 2 ** 3)
2
3  print(result) #should return True
```

Exercise 35 - Numbers & Booleans

Given the code below, use the correct function on line 1 (for which the argument sits inside the parentheses) in order for the final result to be **False**.

```
1 result = (150 % (10 ** 2 / 2))
2
3 print(result) #should return False
```

Exercise 35 - Solution

```
1 result = bool(150 % (10 ** 2 / 2))
2
3 print(result) #should return False
```

Exercise 36 - Lists

Given the code below, use the correct method on line 3 in order to find out the number of elements in **my_list**.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 char = len(my_list)
4
5 print(char)
```

Exercise 36 - Solution

```
1 my_list = [10, 10.5, 20, 30, "Python", "Java", "Ruby"]
2
3 char = len(my_list)
4
5 print(char)
```

Exercise 37 - Lists

Given the code below, use the correct code on line 3 in order to remove the element of **my_list** located at index 5.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_list.
4
5 print(my_list)
```

Exercise 37 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_list.pop(5)
4
5 print(my_list)
```

Exercise 38 - Lists

Given the code below, use the correct method on line 3 in order to add the element **'C++'** at the end of **my_list**.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_list.
4
5 print(my_list)
```

Exercise 38 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_list.append('C++')
4
5 print(my_list)
```

C++ is string so quotes are required

Exercise 39 - Lists

Given the code below, use the correct method on line 3 in order to remove the element **30** from **my_list**.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_list.
4
5 print(my_list)
```

Exercise 39 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_list.remove(30)
4
5 print(my_list)
```

```
my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']

my_list.pop(3)

print(my_list)
```

Exercise 40 - Lists

Given the code below, use the correct method on line 3 in order to return the index of the element **10.5** in **my_list**.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  index = my_list.
4
5  print(index)
```

Exercise 40 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  index = my_list.index(10.5)
4
5  print(index)
```

Exercise 41 - Lists

Given the code below, use the correct method on line 3 in order to insert the element **77** at index 4 in **my_list**.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_list.
4
5  print(my_list)
```

Exercise 41 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_list.insert(4, 77)
4
5  print(my_list)
```

This line inserts the value 77 at index 4 in the list my_list

(Insert keyword)

Exercise 42 - Lists

Given the code below, use the correct method on line 3 in order to concatenate **my_list** with **[100, 101, 102]**, by adding the latter at the end of **my_list**.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_list.
4
5  print(my_list)
```

Exercise 42 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_list.extend([100, 101, 102])
4
5  print(my_list)
```

Exercise 43 - Lists

Given the code below, use the correct method on line 3 in order to find out how many times does the element **20** occur in **my_list**.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  howmany = my_list.
4
5  print(howmany)
```

Exercise 43 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  howmany = my_list.count(20)
4
5  print(howmany)
```

Exercise 44 - Lists

Given the code below, use the correct function on line 3 in order to sort the elements of **my_list** in ascending order.

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 asc =
4
5 print(asc)
```

Exercise 44 - Solution

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 asc = sorted(my_list)
4
5 print(asc)
```

Exercise 45 - Lists

Given the code below, use the correct function (and argument) on line 3 in order to sort the elements of **my_list** in descending order.

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 asc =
4
5 print(asc)
```

Exercise 45 - Solution

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 asc = sorted(my_list, reverse = True)
4
5 print(asc)
```

Exercise 46 - Lists

Given the code below, use the correct function on line 3 in order to find out the smallest number in **my_list**.

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 small =
4
5 print(small)
```

Exercise 46 - Solution

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 small = min(my_list)
4
5 print(small)
```

Exercise 47 - Lists

Given the code below, use the correct function on line 3 in order to find out the largest number in **my_list**.

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 large =
4
5 print(large)
```

Exercise 47 - Solution

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 large = max(my_list)
4
5 print(large)
```

Exercise 48 - Lists

Given the code below, use the correct function on line 3 in order to get the sum of all the elements of **my_list**.

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 add =
4
5 print(add)
```

Exercise 48 - Solution

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 add = sum(my_list)
4
5 print(add)
```

Exercise 49 - Lists

Given the code below, use the correct method on line 3 in order to delete all the elements from **my_list** and obtain an empty list.

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 my_list.
4
5 print(my_list)
```

Exercise 49 - Solution

```
1 my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3 my_list.clear()
4
5 print(my_list)
```

Exercise 50 - Lists

Given the code below, use the correct operators and parentheses on line 3 in order to add the elements of **[30.01, 30.02, 30.03]** to **my_list** and multiply the resulting list by **2**.

```
1  my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3  add = my_list [30.01, 30.02, 30.03] 2
4
5  print(add)
```

Exercise 50 - Solution

```
1  my_list = [10, 10.5, 20, 30, 25.6, 19.25, 11.01, 29.99]
2
3  add = (my_list + [30.01, 30.02, 30.03]) * 2
4
5  print(add)
```

Exercise 51 - Lists

Given the code below, use the correct code on line 3 in order to return the element **20** from **my_list** based on its index.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  element =
4
5  print(element)
```

Exercise 51 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  element = my_list[2]
4
5  print(element)
```

Index -> syntax list

Exercise 52 - Lists

Given the code below, use the correct code on line 3 in order to return the element **Java** from **my_list** based on its index (negative).

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 element =
4
5 print(element)
```

Exercise 52 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 element = my_list[-2]
4
5 print(element)
```

Exercise 53 - Lists

Given the code below, use the correct code on line 3 in order to return a slice made of **[30, 'Python', 'Java']** from **my_list** based on positive indexes.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice =
4
5 print(my_slice)
```

Exercise 53 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice = my_list[3:6]
4
5 print(my_slice)
```

Exercise 54 - Lists

Given the code below, use the correct code on line 3 in order to return a slice made of **[30, 'Python', 'Java']** from **my_list** based on negative indexes.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice =
4
5 print(my_slice)
```

Exercise 54 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice = my_list[-4:-1]
4
5 print(my_slice)
```

Exercise 55 - Lists

Given the code below, use the correct code on line 3 in order to return **my_list** except the first 3 elements, using a single, positive index.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice =
4
5 print(my_slice) #[30, 'Python', 'Java', 'Ruby']
```

Exercise 55 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice = my_list[3:]
4
5 print(my_slice) #[30, 'Python', 'Java', 'Ruby']
```

Exercise 56 - Lists

Given the code below, use the correct code on line 3 in order to return **my_list** except the last 4 elements, using a single, negative index.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice =
4
5 print(my_slice) #[10, 10.5, 20]
```

Exercise 56 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice = my_list[:-4]
4
5 print(my_slice) #[10, 10.5, 20]
```

Exercise 57 - Lists

Given the code below, use the correct code on line 3 in order to return **my_list** except the first 3 elements, using a single, negative index.

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice =
4
5 print(my_slice) #[30, 'Python', 'Java', 'Ruby']
```

Exercise 57 - Solution

```
1 my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3 my_slice = my_list[-4:]
4
5 print(my_slice) #[30, 'Python', 'Java', 'Ruby']
```

Exercise 58 - Lists

Given the code below, use the correct code on line 3 in order to return **my_list** except the last 2 elements, using a single, positive index.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_slice =
4
5  print(my_slice) #[10, 10.5, 20, 30, 'Python']
```

Exercise 58 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_slice = my_list[:5]
4
5  print(my_slice) #[10, 10.5, 20, 30, 'Python']
```

Exercise 59 - Lists

Given the code below, use the correct code on line 3 in order to return every third element of **my_list** starting with the first element of the list.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_slice =
4
5  print(my_slice) #[10, 30, 'Ruby']
```

Exercise 59 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_slice = my_list[::3]
4
5  print(my_slice) #[10, 30, 'Ruby']
```

Exercise 60 - Lists

Given the code below, use the correct code on line 3 in order to return every fourth element of **my_list** starting with the last element of the list.

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_slice =
4
5  print(my_slice) #['Ruby', 20]
```

Exercise 60 - Solution

```
1  my_list = [10, 10.5, 20, 30, 'Python', 'Java', 'Ruby']
2
3  my_slice = my_list[::-4]
4
5  print(my_slice) #['Ruby', 20]
```

Exercise 61 - Sets

Given the code below, use the correct function on line 3 in order to convert **my_list** to a set.

```
1  my_list = [1, 2, 3, 4, 4, 5, 2, 9, 8, 8, 4, 3]
2
3  my_set =
4
5  print(type(my_set))
```

Exercise 61 - Solution

```
1  my_list = [1, 2, 3, 4, 4, 5, 2, 9, 8, 8, 4, 3]
2
3  my_set = set(my_list)
4
5  print(type(my_set))
```

Exercise 62 - Sets

Given the code below, use the correct function on line 3 in order to convert **my_list** to a frozen set.

```
1  my_list = [1, 2, 3, 4, 4, 5, 2, 9, 8, 8, 4, 3]
2
3  my_set =
4
5  print(type(my_set))
```

Exercise 62 - Solution

```
1  my_list = [1, 2, 3, 4, 4, 5, 2, 9, 8, 8, 4, 3]
2
3  my_set = frozenset(my_list)
4
5  print(type(my_set))
```

A frozenset in Python is an immutable set, meaning that once it is created, its contents cannot be changed. This is in contrast to a regular set, which is mutable and can be modified after creation

Exercise 63 - Sets

Given the code below, use the correct code on line 3 in order to verify if element **10** is in **my_set**.

```
1  my_set = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2
3  check =
4
5  print(check)
```

Exercise 63 - Solution

```
1  my_set = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2
3  check = 10 in my_set
4
5  print(check)
```

Exercise 64 - Sets

Given the code below, use the correct method on line 3 in order to add the element **19** to **my_set**.

```
1 my_set = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2
3 my_set.
4
5 print(my_set)
```

Exercise 64 - Solution

```
1 my_set = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2
3 my_set.add(19)
4
5 print(my_set)
```

Exercise 65 - Sets

Given the code below, use the correct method on line 3 in order to delete the element **9** from **my_set**.

```
1 my_set = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2
3 my_set.
4
5 print(my_set)
```

Exercise 65 - Solution

```
1 my_set = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2
3 my_set.remove(9)
4
5 print(my_set)
```

Exercise 66 - Sets

Given the code below, use the correct method on line 4 in order to find out the common elements of **my_set1** and **my_set2**.

```
1  my_set1 = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2  my_set2 = {12, 9, 4, 2, 0, 6}
3
4  common =
5
6  print(sorted(list(common)))
```

Exercise 66 - Solution

```
1  my_set1 = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2  my_set2 = {12, 9, 4, 2, 0, 6}
3
4  common = my_set1.intersection(my_set2)
5
6  print(sorted(list(common)))
```

Exercise 67 - Sets

Given the code below, use the correct method on line 4 in order to join the elements of **my_set1** and **my_set2**.

```
1  my_set1 = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2  my_set2 = {12, 9, 4, 2, 0, 6}
3
4  join =
5
6  print(sorted(list(join)))
```

Exercise 67 - Solution

```
1  my_set1 = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2  my_set2 = {12, 9, 4, 2, 0, 6}
3
4  join = my_set1.union(my_set2)
5
6  print(sorted(list(join)))
```

Exercise 68 - Sets

Given the code below, use the correct method on line 4 in order to find out the elements of **my_set2** that are not members of **my_set1**.

```
1  my_set1 = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2  my_set2 = {12, 9, 4, 2, 0, 6}
3
4  diff =
5
6  print(sorted(list(diff)))
```

Exercise 68 - Solution

```
1  my_set1 = {1, 4, 6, 5, 9, 0, 8, 3, 2, 7, 11}
2  my_set2 = {12, 9, 4, 2, 0, 6}
3
4  diff = my_set2.difference(my_set1)
5
6  print(sorted(list(diff)))
```

Exercise 69 - Tuples

Given the code below, use the correct method on line 3 in order to find out the number of elements in **my_tup**.

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3  number =
4
5  print(number)
```

Exercise 69 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia",
   "Slovenia", "Hungary")
2
3  number = len(my_tup)
4
5  print(number)
```

Exercise 70 - Tuples

Given the code below, use the correct method on line 3 in order to find out the index of **Slovakia** in **my_tup**.

```
1 my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3 index = my_tup.
4
5 print(index)
```

Exercise 70 - Solution

```
1 my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3 index = my_tup.index("Slovakia")
4
5 print(index)
```

Exercise 71 - Tuples

Given the code below, write code in order to perform tuple assignment on line 3 and obtain the results below.

```
1 my_tup = ("Romania", "Poland", "Estonia")
2
3
4
5 print(ro + ", " + po + ", " + es) #returns 'Romania, Poland, Estonia'
```

Exercise 71 - Solution

```
1 my_tup = ("Romania", "Poland", "Estonia")
2
3 (ro, po, es) = my_tup
4
5 print(ro + ", " + po + ", " + es) #returns 'Romania, Poland, Estonia'
```

In Python, you can perform tuple unpacking, which allows you to assign the elements of a tuple to individual variables. Here's how it works:

Tuple Assignment: In the line (ro, po, es) = my_tup, the tuple my_tup is being unpacked into three variables ro, po, and es. Each variable is assigned a corresponding element from the tuple based on its position.

Accessing Elements: After the unpacking, ro will contain the first element of my_tup, po will contain the second element, and es will contain the third element.

Exercise 72 - Tuples

Given the code below, use the correct method on line 3 in order to find out the last element of **my_tup** in alphabetical order.

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3  last =
4
5  print(last) #should return Slovenia
```

The max() function in Python returns the maximum element from a sequence, such as a list, tuple, or string. When used with a tuple like my_tup, it returns the element that comes last when sorted lexicographically.

Exercise 72 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3  last = max(my_tup)
4
5  print(last) #should return Slovenia
```

Exercise 73 - Tuples

Given the code below, use the correct method on line 3 in order to find out the number of occurrences of **Estonia** in **my_tup**.

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Estonia", "Romania", "Hungary", "Slovenia")
2
3  number = my_tup.
4
5  print(number)
```

Exercise 73 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Estonia", "Romania", "Hungary", "Slovenia")
2
3  number = my_tup.count("Estonia")
4
5  print(number)
```

Exercise 74 - Tuples

Given the code below, use the correct slice on line 3 in order to return all the elements of **my_tup**, except the first two of them, using a negative index.

```
1 my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3 my_slice =
4
5 print(my_slice)
```

Exercise 74 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3  my_slice = my_tup[-5:]
4
5  print(my_slice)
```

Exercise 75 - Tuples

Given the code below, use the correct slice on line 3 in order to return all the elements of **my_tup**, except the last three of them, using a negative index.

```
1 my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3 my_slice =
4
5 print(my_slice)
```

Exercise 75 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3  my_slice = my_tup[:-3]
4
5  print(my_slice)
```

Exercise 76 - Tuples

Given the code below, use the correct slice on line 3 in order to return all the elements of **my_tup**, except the first three of them, using a positive index.

```
1 my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3 my_slice =
4
5 print(my_slice)
```

Exercise 76 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia",
   "Slovenia", "Hungary")
2
3  my_slice = my_tup[3:]
4
5  print(my_slice)
```

Exercise 77 - Tuples

Given the code below, use the correct slice on line 3 in order to return all the elements of **my_tup**, except the last two of them, using a positive index.

```
1 my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia", "Slovenia", "Hungary")
2
3 my_slice =
4
5 print(my_slice)
```

Exercise 77 - Solution

```
1  my_tup = ("Romania", "Poland", "Estonia", "Bulgaria", "Slovakia",
   "Slovenia", "Hungary")
2
3  my_slice = my_tup[:5]
4
5  print(my_slice)
```

Exercise 78 - Ranges

Given the code below, use the correct argument(s) for the **range()** function on line 1 in order to return a range of consecutive integers from 0 to 9 inclusively. Use a single argument inside the parentheses of **range()**!

```
1 my_range =
2
3 print(list(my_range)) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Exercise 78 - Solution

```
1 my_range = range(10)
2
3 print(list(my_range)) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Exercise 79 - Ranges

Given the code below, use the correct argument(s) for the **range()** function on line 1 in order to return a range of consecutive integers from 0 to 9 inclusively. Use two arguments inside the parentheses of **range()**!

```
1 my_range =
2
3 print(list(my_range)) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Exercise 79 - Solution

```
1 my_range = range(0, 10)
2
3 print(list(my_range)) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Exercise 80 - Ranges

Given the code below, use the correct argument(s) for the **range()** function on line 1 in order to return a range of consecutive integers from 117 to 129 exclusively.

```
1 my_range =
2
3 print(list(my_range)) #[117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128]
```

Exercise 80 - Solution

```
1 my_range = range(117, 129)
2
3 print(list(my_range)) #[117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128]
```

Exercise 81 - Ranges

Given the code below, use the correct argument(s) for the **range()** function on line 1 in order to return **[10, 13, 16, 19]** when converted to a list.

```
1 my_range =
2
3 print(list(my_range)) #[10, 13, 16, 19]
```

Exercise 81 - Solution

```
1 my_range = range(10, 20, 3)
2
3 print(list(my_range)) #[10, 13, 16, 19]
```

Exercise 82 - Ranges

Given the code below, add the correct step as the third argument of the **range()** function on line 1 in order to return **[115, 120]** when converted to a list.

```
1 my_range = range(115, 125, )
2
3 print(list(my_range)) #[115, 120]
```

Exercise 82 - Solution

```
1 my_range = range(115, 125, 5)
2
3 print(list(my_range)) #[115, 120]
```

Exercise 83 - Ranges

Given the code below, add the correct step as the third argument of the **range()** function on line 1 in order to return **[-75, -60, -45, -30]** when converted to a list.

```
1 my_range = range(-75, -25, )
2
3 print(list(my_range)) #[-75, -60, -45, -30]
```

Exercise 83 - Solution

```
1 my_range = range(-75, -25, 15)
2
3 print(list(my_range)) #[-75, -60, -45, -30]
```

Exercise 84 - Ranges

Given the code below, add the correct step as the third argument of the **range()** function on line 1 in order to return **[-25, 5, 35, 65, 95, 125]** when converted to a list.

```
1 my_range = range(-25, 139, )
2
3 print(list(my_range)) #[-25, 5, 35, 65, 95, 125]
```

Exercise 84 - Solution

```
1 my_range = range(-25, 139, 30)
2
3 print(list(my_range)) #[-25, 5, 35, 65, 95, 125]
```

Exercise 85 - Ranges

Given the code below, write the correct range on line 1 in order to return **[-10]** when converted to a list.

```
1 my_range =
2
3 print(list(my_range)) #[-10]
```

Exercise 85 - Solution

```
1 my_range = range(-10,-9)
2
3 print(list(my_range)) #[-10]
```

Exercise 86 - Dictionaries

Given the code below, use the correct code on line 3 in order to return the value associated with key **4**. Do not use a method as a solution for this exercise!

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3 value =
4
5 print(value)
```

Exercise 86 - Solution

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3 value = crypto[4]
4
5 print(value)
```

Exercise 87 - Dictionaries

Given the code below, use the correct code on line 3 in order to return the value associated with key **4**. This time, use a method as a solution for this exercise!

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3 value =
4
5 print(value)
```

Exercise 87 - Solution

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3 value = crypto.get(4)
4
5 print(value)
```

Exercise 88 - Dictionaries

Given the code below, use the correct code on line 3 in order to update the value associated with key **4** to **"Cardano"**.

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3
4
5 print(crypto[4])
```

Exercise 88 - Solution

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3 crypto[4] = "Cardano"
4
5 print(crypto[4])
```

Exercise 89 - Dictionaries

Given the code below, use the correct code on line 3 in order to add a new key-value pair to the dictionary: **6: "Monero"**

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3
4
5  print(crypto[6])
```

Exercise 89 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
   "XRP"}
2
3  crypto[6] = "Monero"
4
5  print(crypto[6])
```

Exercise 90 - Dictionaries

Given the code below, use the correct code on line 3 in order to return the number of key-value pairs in the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  number =
4
5  print(number)
```

Exercise 90 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
   "XRP"}
2
3  number = len(crypto)
4
5  print(number)
```

Exercise 91 - Dictionaries

Given the code below, use the correct code on line 3 in order to delete the key-value pair associated with key **3**. Do not use a method as a solution for this exercise!

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3
4
5 print(crypto)
```

Exercise 91 - Solution

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3 del crypto[3]
4
5 print(crypto)
```

Exercise 92 - Dictionaries

Given the code below, use the correct code on line 3 in order to delete the key-value pair associated with key **3**. This time, use a method as a solution for this exercise!

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3
4
5 print(crypto)
```

Exercise 92 - Solution

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3 crypto.pop(3)
4
5 print(crypto)
```

Exercise 93 - Dictionaries

Given the code below, use the correct code on line 3 in order to verify that **7** is **not** a key in the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  check =
4
5  print(check)
```

Exercise 93 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  check = 7 not in crypto
4
5  print(check)
```

Exercise 94 - Dictionaries

Given the code below, use the correct method on line 3 in order to delete all the elements in the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  crypto.
4
5  print(crypto)
```

Exercise 94 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  crypto.clear()
4
5  print(crypto)
```

Exercise 95 - Dictionaries

Given the code below, use the correct code on line 3 in order to get a list of tuples, where each tuple represents a key-value pair in the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  result =
4
5  print(list(result))
```

Exercise 95 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
   "XRP"}
2
3  result = crypto.items()
4
5  print(list(result))
```

Exercise 96 - Dictionaries

Given the code below, use the correct function on line 3 in order to get the sum of all the keys in the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  add =
4
5  print(add)
```

Exercise 96 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
   "XRP"}
2
3  add = sum(crypto)
4
5  print(add)
```

Exercise 97 - Dictionaries

Given the code below, use the correct method on line 3 in order to get a list of all the values in the dictionary.

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3 val =
4
5 print(list(val))
```

Exercise 97 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3  val = crypto.values()
4
5  print(list(val))a
```

Exercise 98 - Dictionaries

Given the code below, use the correct function on line 3 in order to get the smallest key in the dictionary.

```
1 crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3 key =
4
5 print(key)
```

Exercise 98 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
"XRP"}
2
3  key = min(crypto)
4
5  print(key)
```

Exercise 99 - Dictionaries

Given the code below, use the correct method on line 3 in order to get a list of all the keys in the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3  keys =
4
5  print(list(keys))
```

Exercise 99 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
   "XRP"}
2
3  keys = crypto.keys()
4
5  print(list(keys))
```

Exercise 100 - Dictionaries

Given the code below, use the correct method on line 3 in order to return and remove an arbitrary key-value pair from the dictionary.

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2
3
4
5  print(len(crypto))
```

Exercise 100 - Solution

```
1  crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5:
   "XRP"}
2
3  crypto.popitem()
4
5  print(len(crypto))
```

Exercise 101 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a string.

```
1  value = 10
2
3  conv =
4
5  print(type(conv))
```

Exercise 101 - Solution

```
1  value = 10
2
3  conv = str(value)
4
5  print(type(conv))
```

Exercise 102 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to an integer.

```
1  value = "10"
2
3  conv =
4
5  print(type(conv))
```

Exercise 102 - Solution

```
1  value = "10"
2
3  conv = int(value)
4
5  print(type(conv))
```

Exercise 103 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a floating-point number.

```
1  value = 10
2
3  conv =
4
5  print(type(conv))
```

Exercise 103 - Solution

```
1  value = 10
2
3  conv = float(value)
4
5  print(type(conv))
```

Exercise 104 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a list.

```
1 value = "Hello!"
2
3 conv =
4
5 print(type(conv))
```

Exercise 104 - Solution

```
1 value = "Hello!"
2
3 conv = list(value)
4
5 print(type(conv))
```

Exercise 105 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a tuple.

```
1 value = [1, 2, 3, 10, 20, 30]
2
3 conv =
4
5 print(type(conv))
```

Exercise 105 - Solution

```
1 value = [1, 2, 3, 10, 20, 30]
2
3 conv = tuple(value)
4
5 print(type(conv))
```

Exercise 106 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a frozen set.

```
1 value = (10, 20, 40, 10, 25, 30, 45)
2
3 conv =
4
5 print(type(conv))
```

Exercise 106 - Solution

```
1 value = (10, 20, 40, 10, 25, 30, 45)
2
3 conv = frozenset(value)
4
5 print(type(conv))
```

Exercise 107 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a binary representation.

```
1 value = 10
2
3 conv =
4
5 print(conv)
```

Exercise 107 - Solution

```
1 value = 10
2
3 conv = bin(value)
4
5 print(conv)
```

Exercise 108 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** to a hexadecimal representation.

```
1 value = 10
2
3 conv =
4
5 print(conv)
```

Exercise 108 - Solution

```
1 value = 10
2
3 conv = hex(10)
4
5 print(conv)
```

Exercise 109 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** from binary to decimal notation.

```
1 value = '0b1010'
2
3 conv =
4
5 print(conv)
```

Exercise 109 - Solution

```
1 value = '0b1010'
2
3 conv = int(value, 2)
4
5 print(conv)
```

Exercise 110 - Data Type Conversions

Given the code below, use the correct function on line 3 in order to convert **value** from hexadecimal to decimal notation.

```
1  value = '0xa'
2
3  conv =
4
5  print(conv)
```

Exercise 110 - Solution

```
1  value = '0xa'
2
3  conv = int(value, 16)
4
5  print(conv)
```

Exercise 111 - Conditionals

Considering the code below, write code that prints out **True!** if **x** has 50 characters or more.

```
1  x = "The days of Python 2 are almost over. Python 3 is the king now."
2
3
```

Exercise 111 - Solution

```
1  x = "The days of Python 2 are almost over. Python 3 is the king now."
2
3  if len(x) >= 50:
4      print("True!")
```

Exercise 112 - Conditionals

Considering the code below, write code that prints out **True!** if **x** is a string and the first character in the string is **T**.

```
x = "The days of Python 2 are almost over. Python 3 is the king now."


```

Exercise 112 - Solution

```
x = "The days of Python 2 are almost over. Python 3 is the king now."

if type(x) is str and x.startswith("T"):
    print("True!")
```

Exercise 113 - Conditionals

Considering the code below, write code that prints out **True!** if at least one of the following conditions occurs:

- the string contains the character **z**
- the string contains the character **y** at least twice

```
x = "The days of Python 2 are almost over. Python 3 is the king now."


```

Exercise 113 - Solution

```
x = "The days of Python 2 are almost over. Python 3 is the king now."

if "z" in x or x.count("y") >= 2:
    print("True!")
```

Exercise 114 - Conditionals

Considering the code below, write code that prints out **True!** if the index of the first occurrence of letter **f** is less than 10 and prints out **False!** if the same condition is not satisfied.

```
1  x = "The days of Python 2 are almost over. Python 3 is the king now."
2
3
```

Exercise 114 - Solution

```
1  x = "The days of Python 2 are almost over. Python 3 is the king now
2
3  if x.index("f") < 10:
4      print("True!")
5  else:
6      print("False!")
```

Exercise 115 - Conditionals

Considering the code below, write code that prints out **True!** if the last 3 characters of the string are all digits and prints out **False!** otherwise.

Hint! *Use the appropriate method to check if the requested string slice contains digits only.*

```
1  x = "The days of Python 2 are almost over. Python 3 is the king now."
2
3
```

Exercise 115 - Solution

```
1  x = "The days of Python 2 are almost over. Python 3 is the king now."
2
3  if x[-3:].isdigit():
4      print("True!")
5  else:
6      print("False!")
```

Exercise 116 - Conditionals

Considering the code below, write code that prints out **True!** if **x** has at least 8 elements and the element positioned at index 6 is a floating-point number and prints out **False!** otherwise.

```
x = [115, 115.9, 116.01, ["length", "width", "height"], 109, 115, 119.5, ["length", "width", "height"]]


```

Exercise 116 - Solution

```
x = [115, 115.9, 116.01, ["length", "width", "height"], 109, 115, 119.5, ["length", "width", "height"]]

if len(x) >= 8 and type(x[6]) is float:
    print("True!")
else:
    print("False!")
```

Exercise 117 - Conditionals

Considering the code below, write code that prints out **True!** if the second string of the first list in **x** ends with the letter **h** and the first string of the second list in **x** also ends with the letter **h**, and prints out **False!** otherwise.

```
x = [115, 115.9, 116.01, ["length", "width", "height"], 109, 115, 119.5, ["length", "width", "height"]]


```

Exercise 117 - Solution

```
x = [115, 115.9, 116.01, ["length", "width", "height"], 109, 115, 119.5, ["length", "width", "height"]]

if x[3][1].endswith("h") and x[7][0].endswith("h"):
    print("True!")
else:
    print("False!")
```

Exercise 118 - Conditionals

Considering the code below, write code that prints out **True!** if one of the following two conditions are satisfied and prints out **False!** otherwise.

- the third string of the first list in **x** ends with the letter **h**
- the second string of the second list in **x** also ends with the letter **h**

```
x = [115, 115.9, 116.01, ["length", "width", "height"], 109, 115, 119.5, ["length", "width", "height"]]


```

Exercise 118 - Solution

```
x = [115, 115.9, 116.01, ["length", "width", "height"], 109, 115, 119.5, ["length", "width", "height"]]

if x[3][2].endswith("h") or x[7][1].endswith("h"):
    print("True!")
else:
    print("False!")
```

Exercise 119 - Conditionals

Considering the code below, write code that prints out **True!** if the largest value among the first 3 elements of the list is less than or equal to the smallest value among the next 3 elements of the list. Otherwise, print out **False!**

Hint! *Use the appropriate slices to make the comparison.*

```
x = [115, 115.9, 116.01, 109, 115, 119.5, ["length", "width", "height"]]


```

Exercise 119 - Solution

```
x = [115, 115.9, 116.01, 109, 115, 119.5, ["length", "width", "height"]]

if max(x[:3]) <= min(x[3:6]):
    print("True!")
else:
    print("False!")
```

Exercise 120 - Conditionals

Considering the code below, write code that prints out **True!** if **115** appears at least once inside the list or if it is the first element in the list. Otherwise, print out **False!**

Hint! *Use the appropriate method to check if* ***115*** *is the first element in the list.*

```
x = [115, 115.9, 116.01, 109, 115, 119.5, ["length", "width", "height"]]


```

Exercise 120 - Solution

```
x = [115, 115.9, 116.01, 109, 115, 119.5, ["length", "width", "height"]]

if x.count(115) >= 1 or x.index(115) == 0:
    print("True!")
else:
    print("False!")
```

Exercise 121 - Conditionals

Considering the code below, write code that prints out **True!** if the value associated with key number **5** is **Perl** or the number of key-value pairs in the dictionary divided by 5 returns a remainder less than 2. Otherwise, print out **False!**

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}


```

Exercise 121 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if x[5] == "Perl" or len(x) % 5 < 2:
    print("True!")
else:
    print("False!")
```

Exercise 122 - Conditionals

Considering the code below, write code that prints out **True!** if **3** is a key in the dictionary and the smallest value (alphabetically) in the dictionary is **C#**. Otherwise, print out **False!**

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}


```

Exercise 122 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if 3 in x and sorted(x.values())[0] == "C#":
    print("True!")
else:
    print("False!")
```

Exercise 123 - Conditionals

Considering the code below, write code that prints out **True!** if the last character of the largest (alphabetically) value in the dictionary is **n**. Otherwise, print out **False!**

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}


```

Exercise 123 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if sorted(x.values())[-1][-1] == "n":
    print("True!")
else:
    print("False!")
```

Exercise 124 - Conditionals

Considering the code below, write code that prints out **True!** if the largest key in the dictionary divided by the second largest key in the dictionary returns a remainder equal to the smallest key in the dictionary. Otherwise, print out **False!**

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}


```

Exercise 124 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if sorted(x.keys())[-1] % sorted(x.keys())[-2] == sorted(x.keys())[0]:
    print("True!")
else:
    print("False!")
```

Exercise 125 - Conditionals

Considering the code below, write code that prints out **True!** if the sum of all the keys in the dictionary is less than the number of characters of the string obtained by concatenating the values associated with the first 5 keys in the dictionary. Otherwise, print out **False!**

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}


```

Exercise 125 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if sum(x) < len(x[1] + x[2] + x[3] + x[4] + x[5]):
    print("True!")
else:
    print("False!")
```

Exercise 126 - Conditionals

Considering the code below, write code that prints out **True!** if the 3rd element of the first range is less than 2, prints out **False!** if the 5th element of the first range is 5, and prints out **None!** otherwise.

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]


```

Exercise 126 - Solution

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]

if x[0][2] < 2:
    print("True!")
elif x[0][4] == 5:
    print("False!")
else:
    print("None!")
```

Exercise 127 - Conditionals

Considering the code below, write code that prints out **True!** if the 3rd element of the 3rd range is less than 6, prints out **False!** if the 1st element of the second range is 5, and prints out **None!** otherwise.

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]


```

Exercise 127 - Solution

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]

if x[2][2] < 6:
    print("True!")
elif x[1][0] == 5:
    print("False!")
else:
    print("None!")
```

Exercise 128 - Conditionals

Considering the code below, write code that prints out **True!** if the last element of the first range is greater than 3, prints out **False!** if the last element of the second range is less than 9, and prints out **None!** otherwise.

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]


```

Exercise 128 - Solution

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]

if x[0][-1] > 3:
    print("True!")
elif x[1][-1] < 9:
    print("False!")
else:
    print("None!")
```

Exercise 129 - Conditionals

Considering the code below, write code that prints out **True!** if the length of the first range is greater than or equal to 5, prints out **False!** if the length of the second range is 4, and prints out **None!** otherwise.

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]


```

Exercise 129 - Solution

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]

if len(x[0]) >= 5:
    print("True!")
elif len(x[1]) == 4:
    print("False!")
else:
    print("None!")
```

Exercise 130 - Conditionals

Considering the code below, write code that prints out **True!** if the sum of all the elements of the first range is greater than the sum of all the elements of the third range, prints out **False!** if the largest element of the second range is greater than the largest element of the third range, and prints out **None!** otherwise.

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]


```

Exercise 130 - Solution

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]

if sum(x[0]) > sum(x[2]):
    print("True!")
elif max(x[1]) > max(x[2]):
    print("False!")
else:
    print("None!")
```

Exercise 131 - Conditionals

Considering the code below, write code that prints out **True!** if the largest element of the first range minus the second element of the 3rd range is equal to the first element of the first range, prints out **False!** if the length of the first range minus the length of the 2nd range is equal to the first element of the 3rd range, prints out **Maybe!** if the sum of all the elements of the 3rd range divided by 2 returns a remainder of 0, and prints out **None!** otherwise.

```
1  x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]
2
3
```

Exercise 131 - Solution

```
1  x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]
2
3  if max(x[0]) - x[2][1] == x[0][0]:
4      print("True!")
5  elif len(x[0]) - len(x[1]) == x[2][0]:
6      print("False!")
7  elif sum(x[2]) % 2 == 0:
8      print("Maybe!")
9  else:
10     print("None!")
```

Exercise 132 - Conditionals

Considering the code below, write code that prints out **True!** if the sum of the last 3 elements of the first range plus the sum of the last 3 elements of the 3rd range is equal to the sum of the last 3 elements of the 2nd range, and prints out **False!** if the length of the first range times 2 is less than the sum of all the elements of the 3rd range.

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]


```

Exercise 132 - Solution

```
x = [list(range(5)), list(range(5,9)), list(range(1,10,3))]

if sum(x[0][-3:]) + sum(x[2][-3:]) == sum(x[1][-3:]):
    print("True!")
elif len(x[0]) * 2 < sum(x[2]):
    print("False!")
```

Exercise 133 - Conditionals

Considering the code below, write code that prints out **True!** if the 2nd character of the value at key 1 is also present in the value at key 4, and prints out **False!** otherwise.

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}


```

Exercise 133 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if x[1][1] in x[4]:
    print("True!")
else:
    print("False!")
```

Exercise 134 - Conditionals

Considering the code below, write code that prints out **True!** if the second to last character of the value at key 3 is the first character of the value at key 5, and prints out **False!** otherwise.

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}
```

Exercise 134 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if x[3][-2] == x[5][0]:
    print("True!")
else:
    print("False!")
```

Exercise 135 - Conditionals

Considering the code below, write code that prints out **True!** if the number of characters of the smallest value in the dictionary is equal to the number of occurrences of letter **a** in the value at key 3, and prints out **False!** otherwise.

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}
```

Exercise 135 - Solution

```
x = {1: "Python", 2: "Java", 3: "Javascript", 4: "Ruby", 5: "Perl", 6: "C#", 7: "C++"}

if len(min(x.values())) == x[3].count("a"):
    print("True!")
else:
    print("False!")
```

Exercise 136 - Loops

Write a **for** loop that iterates over the **x** list and prints out all the elements of the list.

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3
```

Exercise 136 - Solution

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3  for i in x:
4      print(i)
```

Exercise 137 - Loops

Write a **for** loop that iterates over the **x** list and prints out the remainders of each element of the list divided by 3.

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3
```

Exercise 137 - Solution

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3  for i in x:
4      print(i % 3)
```

Exercise 138 - Loops

Write a **for** loop that iterates over the **x** list and prints out all the elements of the list in reversed order and multiplied by 10.

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3
```

Exercise 138 - Solution

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3  for i in sorted(x, reverse = True):
4      print(i * 10)
```

Exercise 139 - Loops

Write a **for** loop that iterates over the **x** list and prints out all the elements of the list divided by 2 and the string **Great job!** after the list is exhausted.

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3
```

Exercise 139 - Solution

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3  for i in x:
4      print(i / 2)
5  else:
6      print("Great job!")
```

Exercise 140 - Loops

Write a **for** loop that iterates over the **x** list and prints out the index of each element.

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3
```

Exercise 140 - Solution

```
1  x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
2
3  for i in x:
4      print(x.index(i))
```

Exercise 141 - Loops

Write a **while** loop that prints out the value of **x** squared while **x** is less than or equal to 5. Be careful not to end up with an infinite loop!

```
1  x = 0
2
3
```

Exercise 141 - Solution

```
1  x = 0
2
3  while x <= 5:
4      print(x ** 2)
5      x = x + 1
```

Exercise 142 - Loops

Write a **while** loop that prints out the value of **x** times 10 while **x** is less than or equal to 4 and then prints out **Done!** when **x** becomes larger than 4. Be careful not to end up with an infinite loop!

```
1  x = 0
2
3
```

Exercise 142 - Solution

```
1  x = 0
2
3  while x <= 4:
4      print(x * 10)
5      x = x + 1
6  else:
7      print("Done!")
```

Exercise 143 - Loops

Write a **while** loop that prints out the value of **x** plus 10 while **x** is less than or equal to 15 and the remainder of **x** divided by 5 is 0. Be careful not to end up with an infinite loop!

```
1  x = 10
2
3
```

Exercise 143 - Solution

```
1  x = 10
2
3  while x <= 15 and x % 5 == 0:
4      print(x + 10)
5      x = x + 1
```

Exercise 144 - Loops

Write a **while** loop that prints out the absolute value of **x** while **x** is negative. Be careful not to end up with an infinite loop!

```
x = -7


```

Exercise 144 - Solution

```
x = -7

while x < 0:
    print(abs(x))
    x = x + 1
```

Exercise 145 - Loops

Write a **while** loop that prints out the value of **x** times **y** while **x** is greater than or equal to 5 and less than 10, and prints out the result of **x** divided by **y** when **x** becomes 10. Be careful not to end up with an infinite loop!

```
x = 5
y = 2


```

Exercise 145 - Solution

```
x = 5
y = 2

while x >= 5 and x < 10:
    print(x * y)
    x = x + 1
else:
    print(x / y)
```

Exercise 146 - Loops

Write code that will iterate over the **x** list and multiply by 10 only the elements that are greater than 20 and print them out to the screen.

Hint: use nesting!

```
x = [10, 12, 13, 14, 17, 19, 21, 22, 25]
```

Exercise 146 - Solution

```
x = [10, 12, 13, 14, 17, 19, 21, 22, 25]

for i in x:
    if i > 20:
        print(i * 10)
```

Exercise 147 - Loops

Write code that will iterate over the **x** and **y** lists and multiply each element of **x** with each element of **y**, also printing the results to the screen.

Hint: use nesting!

```
x = [2, 4, 6]
y = [5, 10]
```

Exercise 147 - Solution

```
x = [2, 4, 6]
y = [5, 10]

for i in x:
    for j in y:
        print(i * j)
```

Exercise 148 - Loops

Write code that will iterate over the **x** and **y** lists and multiply each element of **x** with each element of **y** that is less than 12, also printing the results to the screen.

Hint: use nesting!

```
1  x = [2, 4, 6, 8]
2  y = [5, 10, 15, 20]
3
4
```

Exercise 148 - Solution

```
1  x = [2, 4, 6, 8]
2  y = [5, 10, 15, 20]
3
4  for i in x:
5      for j in y:
6          if j < 12:
7              print(i * j)
```

Exercise 149 - Loops

Write code that will iterate over the **x** and **y** lists and multiply each element of **x** that is greater than 5 with each element of **y** that is less than 12, also printing the results to the screen.

Hint: use nesting!

```
1  x = [2, 4, 6, 8]
2  y = [5, 10, 15, 20]
3
4
```

Exercise 149 - Solution

```
1  x = [2, 4, 6, 8]
2  y = [5, 10, 15, 20]
3
4  for i in x:
5      for j in y:
6          if i> 5 and j < 12:
7              print(i * j)
```

Exercise 150 - Loops

Write code that will iterate over the **x** and **y** lists and multiply each element of **x** with each element of **y** that is less than or equal to 10, also printing the results to the screen. For **y**'s elements that are greater than 10, multiply each element of **x** with **y** squared.

Hint: use nesting!

```
1  x = [2, 4, 6, 8]
2  y = [5, 10, 15, 20]
3
4
```

Exercise 150 - Solution

```
1  x = [2, 4, 6, 8]
2  y = [5, 10, 15, 20]
3
4  for i in x:
5      for j in y:
6          if j <= 10:
7              print(i * j)
8          else:
9              print(i * j ** 2)
```

Exercise 151 - Loops

Write code that will print out each character in **x** doubled if that character is also inside **y**.

Hint: use nesting!

```
1  x = "cryptocurrency"
2  y = "blockchain"
3
4
```

Exercise 151 - Solution

```
1  x = "cryptocurrency"
2  y = "blockchain"
3
4  for i in x:
5      if i in y:
6          print(i * 2)
```

Exercise 152 - Loops

Write code that will iterate over the range generated by **range(9)** and for each element that is between 3 and 7 inclusively print out the result of multiplying that element by the second element in the same range.

Hint: use nesting!

```
1 my_range = range(9)
2
3
```

Exercise 152 - Solution

```
1 my_range = range(9)
2
3 for i in my_range:
4     if 3 <= i <= 7:
5         print(i * my_range[1])
```

Exercise 153 - Loops

Write code that will iterate over the range starting at 1, up to but not including 11, with a step of 2, and for each element that is between 3 and 8 inclusively print out the result of multiplying that element by the last element in the same range. For any other element of the range (outside [3-8]) print **Outside!**

Hint: use nesting!

```
1
```

Exercise 153 - Solution

```
1 for i in range(1,11,2):
2     if 3 <= i <= 8:
3         print(i * range(1,11,2)[-1])
4     else:
5         print("Outside!")
```

Exercise 154 - Loops

Write code that will iterate over the range starting at 5, up to but not including 25, with a step of 5, and for each element that is between 10 and 21 inclusively print out the result of multiplying that element by the second to last element of the same range. For any other element of the range (outside [10-21]) print **Outside!** Finally, after the entire range is exhausted print out **The end!**

Hint: use nesting!

Exercise 154 - Solution

```
for i in range(5,25,5):
    if 10 <= i <= 21:
        print(i * range(5,25,5)[-2])
    else:
        print("Outside!")
else:
    print("The end!")
```

Exercise 155 - Loops

Write a **while** loop that prints out the value of **x** times 11 while **x** is less than or equal to 11. When x becomes equal to 10, print out **x is 10!** Be careful not to end up with an infinite loop!

```
x = 5
```

Exercise 155 - Solution

```
x = 5

while x <= 11:
    if x == 10:
        print("x is 10!")
        x = x + 1
    else:
        print(x * 11)
        x = x + 1
```

Exercise 156 - Loops

Insert a **break** statement where necessary in order to obtain the following result:

```
1
1
100
20
10
```

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
        print(i)
    print(j)
```

Exercise 156 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
            break
        print(i)
    print(j)
```

Exercise 157 - Loops

Insert a **break** statement where necessary in order to obtain the following result:

```
1
10
20
2
10
```

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
        print(i)
    print(j)
```

Exercise 157 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
        print(i)
        break
    print(j)
```

Exercise 158 - Loops

Insert a **break** statement where necessary in order to obtain the following result:

```
1
1
100
10
```

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
        print(i)
    print(j)
```

Exercise 158 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            break
            print(i * j)
        print(i)
    print(j)
```

Exercise 159 - Loops

Insert a **continue** statement where necessary in order to obtain the following result:

```
1
1
100
20
200
100
```

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
        print(i)
    print(j)
```

Exercise 159 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
            continue
        print(i)
    print(j)
```

Exercise 160 - Loops

Insert a **continue** statement where necessary in order to obtain the following result:

```
1
1
100
100
```

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
        print(i)
    print(j)
```

Exercise 160 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            continue
            print(i * j)
        print(i)
    print(j)
```

Exercise 161 - Exceptions

Fix the code below so that it doesn't generate a **SyntaxError**.

Hint! The result should be `20 200`

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0
            print(i * j)
```

Exercise 161 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
```

Exercise 162 - Exceptions

Fix the code below so that it doesn't generate a **SyntaxError**.

Hint! The result should be `20 200`

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print i * j
```

Exercise 162 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
```

Exercise 163 - Exceptions

Fix the code below so that it doesn't generate a **NameError**.

Hint! The result should be `20 200`

```
x = [1, 2]
y = [10, 100]

for i in x:
    for i in y:
        if i % 2 == 0:
            print(i * j)
```

Exercise 163 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
```

Exercise 164 - Exceptions

Fix the code below so that it doesn't generate a **TypeError**.

Hint! The result should be `20 200`

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if y % 2 == 0:
            print(i * j)
```

Exercise 164 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(i * j)
```

Exercise 165 - Exceptions

Fix the code below so that it doesn't generate an **IndexError**.

Hint! The result should be `200 200`

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(x[1] * y[2])
```

Exercise 165 - Solution

```
x = [1, 2]
y = [10, 100]

for i in x:
    for j in y:
        if i % 2 == 0:
            print(x[1] * y[1])
```

Exercise 166 - Exceptions

Add the necessary clause(s) to the code below so that in case the **ZeroDivisionError** exception is raised then the program prints out **Zero!** to the screen.

```
try:
    print(25 % 0)

```

Exercise 166 - Solution

```
try:
    print(25 % 0)
except ZeroDivisionError:
    print("Zero!")
```

Exercise 167 - Exceptions

Add the necessary clause(s) to the code below so that in case the code under **try** raises no exceptions then the program prints out the result of the math operation and the string **Clean!** to the screen.

```
try:
    print(25 % 5 ** 5 + 5)
except:
    print("Bug!")

```

Exercise 167 - Solution

```
try:
    print(25 % 5 ** 5 + 5)
except:
    print("Bug!")
else:
    print("Clean!")
```

Exercise 168 - Exceptions

Add the necessary clause(s) to the code below so that no matter if the code under **try** raises any exceptions or not, then the program prints out the string **Result!** to the screen.

```
try:
    print(25 % 0 ** 5 + 5)
except:
    print("Bug!")

```

Exercise 168 - Solution

```
try:
    print(25 % 0 ** 5 + 5)
except:
    print("Bug!")
finally:
    print("Result!")
```

Exercise 169 - Exceptions

Add the necessary clause(s) to the code below so that in case the code under **try** raises the **ZeroDivisionError** exception then the program prints out the string **Zero!** to the screen; additionally, if the code under **try** raises the **IndexError** exception then the program prints out the string **Index!** to the screen.

```
x = [1, 9, 17, 32]

try:
    print(x[3] % 3 ** 5 + x[4])

```

Exercise 169 - Solution

```
x = [1, 9, 17, 32]

try:
    print(x[3] % 3 ** 5 + x[4])
except ZeroDivisionError:
    print("Zero!")
except IndexError:
    print("Index!")
```

Exercise 170 - Exceptions

Add the necessary clause(s) to the code below so that in case the code under **try** raises no exceptions then the program prints out the result of the math operation and the string **Clean!** to the screen. If the code under **try** raises the **ZeroDivisionError** exception then the program prints **Zero!** to the screen. Ultimately, regardless of the result generated by the code under **try**, the program should print out **Finish!** to the screen.

```
try:
    print(25 % 5 ** 5 + 5)

```

Exercise 170 - Solution

```
try:
    print(25 % 5 ** 5 + 5)
except ZeroDivisionError:
    print("Zero!")
else:
    print("Clean!")
finally:
    print("Finish!")
```

Exercise 171 - Functions

Implement a function called **my_func()** that simply prints out **Hello Python!** to the screen and call the function.

Exercise 171 - Solution

```
1  def my_func():
2      print("Hello Python!")
3
4  my_func()
```

Exercise 172 - Functions

Implement a function called **my_func()** that creates a variable **add** which stores the result of adding 10 and 20, and prints out the value of **add**. Don't forget to also call the function!

Exercise 172 - Solution

```
def my_func():
    add = 10 + 20
    print(add)

my_func()
```

Exercise 173 - Functions

Implement a function called **my_func()** that takes a single parameter **x** and multiplies it with 10, also returning the result when the function is called.

```
result = my_func(7)
print(result)
```

Exercise 173 - Solution

```
def my_func(x):
    return x * 10

result = my_func(7)
print(result)
```

Exercise 174 - Functions

Implement a function called **my_func()** that takes two parameters **x** and **y** and divides **x** by **y**, also returning the result when the function is called.

```
1
2
3 result = my_func(38,19)
4 print(result)
```

Exercise 174 - Solution

```
1 def my_func(x, y):
2     return x / y
3
4 result = my_func(38,19)
5 print(result)
```

Exercise 175 - Functions

Implement a function called **my_func()** that takes 3 parameters **x**, **y** and **z** and raises **x** to the power of **y** then adds **z**, also returning the result when the function is called.

```
1
2
3 result = my_func(3,3,3)
4 print(result)
```

Exercise 175 - Solution

```
1 def my_func(x,y,z):
2     return x ** y + z
3
4 result = my_func(3,3,3)
5 print(result)
```

Exercise 176 - Functions

Implement a function called **my_func()** that takes a single parameter **x** and multiplies it with each element of **range(5)**, also adding each multiplication result to a new (initially empty) list called **my_new_list**. Finally, the list should be printed out to the screen after the function is called.

```
1
2
3 result = my_func(2)
4 print(result)
```

Exercise 176 - Solution

```
1 def my_func(x):
2     my_new_list = []
3     for i in range(5):
4         my_new_list.append(i * x)
5     return my_new_list
6
7 result = my_func(2)
8 print(result)
```

Exercise 177 - Functions

Implement a function called **my_func()** that takes a single parameter **x** (a string) and turns each character of the string to uppercase, also returning the result when the function is called.

```
1
2
3 result = my_func("Satoshi Nakamoto")
4 print(result)
```

Exercise 177 - Solution

```
1 def my_func(x):
2     return x.upper()
3
4 result = my_func("Satoshi Nakamoto")
5 print(result)
```

Exercise 178 - Functions

Implement a function called **my_func()** that takes a single parameter **x** (a list) and eliminates all duplicates from the list, also returning the result when the function is called.

```
1
2
3 result = my_func([11, 12, 13, 11, 15, 18, 18, 22, 20, 16, 12])
4 print(result)
```

Exercise 178 - Solution

```
1 def my_func(x):
2     return list(set(x))
3
4 result = my_func([11, 12, 13, 11, 15, 18, 18, 22, 20, 16, 12])
5 print(result)
```

Exercise 179 - Functions

Implement a function called **my_func()** that takes a single parameter **x** (a tuple) and for each element of the tuple that is greater than 4 it raises that element to the power of 2, also adding it to a new (initially empty) list called **my_new_list**. Finally, the code returns the result when the function is called.

```
1
2
3 result = my_func((2, 3, 5, 6, 4, 8, 9))
4 print(result)
```

Exercise 179 - Solution

```
1 def my_func(x):
2     my_new_list = []
3     for i in x:
4         if i > 4:
5             my_new_list.append(i ** 2)
6     return my_new_list
7
8 result = my_func((2, 3, 5, 6, 4, 8, 9))
9 print(result)
```

Exercise 180 - Functions

Implement a function called **my_func()** that takes a single parameter **x** (a dictionary) and multiplies the number of elements in the dictionary with the largest key in the dictionary, also returning the result when the function is called.

```
1
2
3 result = my_func({1: 3, 2: 3, 4: 5, 5: 9, 6: 8, 3: 7, 7: 0})
4 print(result)
```

Exercise 180 - Solution

```
1 def my_func(x):
2     return len(x) * sorted(x.keys())[-1]
3
4 result = my_func({1: 3, 2: 3, 4: 5, 5: 9, 6: 8, 3: 7, 7: 0})
5 print(result)
```

Exercise 181 - Functions

Implement a function called **my_func()** that takes a single positional parameter **x** and a default parameter **y** which is equal to 10 and multiplies the two, also returning the result when the function is called.

```
1
2
3 result = my_func(5)
4 print(result)
```

Exercise 181 - Solution

```
1 def my_func(x, y = 10):
2     return x * y
3
4 result = my_func(5)
5 print(result)
```

Exercise 182 - Functions

Implement a function called **my_func()** that takes a single positional parameter **x** and two default parameters **y** and **z** which are equal to 100 and 200 respectively, and adds them together, also returning the result when the function is called.

```
1
2
3 result = my_func(50)
4 print(result)
```

Exercise 182 - Solution

```
1 def my_func(x, y = 100, z = 200):
2     return x + y + z
3
4 result = my_func(50)
5 print(result)
```

Exercise 183 - Functions

Implement a function called **my_func()** that takes two default parameters **x** (a list) and **y** (an integer), and returns the element in **x** positioned at index **y**, also printing the result to the screen when called.

```
1
2
3 result = my_func(list(range(2,25,2)), 4)
4 print(result) #result should be 10
```

Exercise 183 - Solution

```
1 def my_func(x, y):
2     return x[y]
3
4 result = my_func(list(range(2,25,2)), 4)
5 print(result)
```

Exercise 184 - Functions

Implement a function called **my_func()** that takes a positional parameter **x** and a **variable-length** tuple of parameters and returns the result of multiplying **x** with the second element in the tuple, also returning the result when the function is called.

```
1
2
3 result = my_func(5, 10, 20, 30, 50)
4 print(result)
```

Exercise 184 - Solution

```
1 def my_func(x, *args):
2     return x * args[1]
3
4 result = my_func(5, 10, 20, 30, 50)
5 print(result)
```

Exercise 185 - Functions

Implement a function called **my_func()** that takes a positional parameter **x** and a **variable-length** dictionary of (keyword) parameters and returns the result of multiplying **x** with the largest value in the dictionary, also returning the result when the function is called.

```
1
2
3 result = my_func(10, val1 = 10, val2 = 15, val3 = 20, val4 = 25, val5 = 30)
4 print(result)
```

Exercise 185 - Solution

```
1 def my_func(x, **kwargs):
2     return x * sorted(kwargs.values())[-1]
3
4 result = my_func(10, val1 = 10, val2 = 15, val3 = 20, val4 = 25, val5 =
30)
5 print(result)
```

Exercise 186 - Functions

Add the correct line(s) of code inside the function in order to get **200** as a result of calling **my_func()** and have the result printed out to the screen.

```
1  var = 10
2
3  def my_func(x):
4
5
6  my_func(20)
```

Exercise 186 - Solution

```
1  var = 10
2
3  def my_func(x):
4      print(x * var)
5
6  my_func(20)
```

Exercise 187 - Functions

Add the correct line(s) of code inside the function in order to get **100** as a result of calling **my_func()** and have the result printed out to the screen.

```
1  var = 10
2
3  def my_func(x):
4      print(x * var)
5
6  my_func(20)
```

Exercise 187 - Solution

```
1  var = 10
2
3  def my_func(x):
4      var = 5
5      print(x * var)
6
7  my_func(20)
```

Exercise 188 - Functions

Make the necessary adjustment inside the function in order to get **120** as a result of calling **my_func()** and have the result printed out to the screen.

```
def my_func(x):
    print(x * var)
    var = 12

my_func(10)
```

Exercise 188 - Solution

```
def my_func(x):
    var = 12
    print(x * var)

my_func(10)
```

Exercise 189 - Functions

Add the necessary line of code inside the function in order to get **80** as a result of calling **my_func()** and have the result printed out to the screen.

```
var = 8

def my_func(x):
    print(x * var)
    var = 12

my_func(10)
```

Exercise 189 - Solution

```
var = 8

def my_func(x):
    global var
    print(x * var)
    var = 12

my_func(10)
```

Exercise 190 - Functions

Write code that will import only the **pi** variable from the **math** module and then it will format it in order to have only 4 digits after the floating point. Of course, print out the result to the screen using the **print()** function.

Exercise 190 - Solution

```
1  from math import pi
2
3  print("%.4f" % pi)
```

Exercise 191 - Files

Add the necessary code in between **print**'s parentheses in order to read the content of **test.txt** as a string and have the result printed out to the screen.

```
1  f = open("test.txt", "r")
2
3  print()
```

Exercise 191 - Solution

```
1  f = open("test.txt", "r")
2
3  print(f.read())
```

Exercise 192 - Files

Add the necessary code in between **print**'s parentheses in order to read the content of **test.txt** as a list where each element of the list is a row in the file, and have the result printed out to the screen.

```
1  f = open("test.txt", "r")
2
3  print()
```

Exercise 192 - Solution

```
1  f = open("test.txt", "r")
2
3  print(f.readlines())
```

Exercise 193 - Files

Add the necessary code on line 5 in order to bring back the cursor at the very beginning of **test.txt** before reading from the file once again.

```
1  f = open("test.txt", "r")
2
3  f.read()
4
5
6
7  print(f.read())
```

Exercise 193 - Solution

```
1  f = open("test.txt", "r")
2
3  f.read()
4
5  f.seek(0)
6
7  print(f.read())
```

Exercise 194 - Files

Add the necessary code on line 5 (in between the parentheses of **print()**) in order to get the current position of the cursor inside **test.txt** and have the result printed out to the screen.

```
1  f = open("test.txt", "r")
2
3  f.read(5)
4
5  print()
```

Exercise 194 - Solution

```
1  f = open("test.txt", "r")
2
3  f.read(5)
4
5  print(f.tell())
```

Exercise 195 - Files

Add the necessary code on line 5 (in between the parentheses of **print()**) in order to get the current mode in which **test.txt** is open (read, write etc.) and have the result printed out to the screen.

```
1  f = open("test.txt", "r")
2
3  f.read(5)
4
5  print()
```

Exercise 195 - Solution

```
1  f = open("test.txt", "r")
2
3  f.read(5)
4
5  print(f.mode)
```

Exercise 196 - Files

Add the necessary file access mode on line 1 in order to open **test.txt** for appending and reading at the same time.

```
1  f = open("test.txt", )
2
3  print(f.mode)
```

Exercise 196 - Solution

```
1  f = open("test.txt", "a+")
2
3  print(f.mode)
```

Exercise 197 - Files

Add the necessary code on lines 3 and 4 in order to write the string **python** to **test.txt** and have the result of reading the file printed out to the screen.

```
1 f = open("test.txt", "w")
2
3
4
5
6 f = open("test.txt", "r")
7
8 print(f.read())
```

Exercise 197 - Solution

```
1 f = open("test.txt", "w")
2
3 f.write("python")
4 f.close()
5
6 f = open("test.txt", "r")
7
8 print(f.read())
```

Exercise 198 - Files

Add the necessary code on lines 3 and 4 in order to write a list of strings **['python', ' ', 'and', ' ', 'java']** to **test.txt** and have the result of reading the file printed out to the screen.

```
1 f = open("test.txt", "w")
2
3
4
5
6 f = open("test.txt", "r")
7
8 print(f.read())
```

Exercise 198 - Solution

```
1 f = open("test.txt", "w")
2
3 f.writelines(['python', ' ', 'and', ' ', 'java'])
4 f.close()
5
6 f = open("test.txt", "r")
7
8 print(f.read())
```

Exercise 199 - Files

Add the necessary code starting at line 1 in order to write the string **python** and also close **test.txt** properly using the **with** statement.

```
1
2
3
4
5 f = open("test.txt", "r")
6
7 print(f.read())
```

Exercise 199 - Solution

```
1 with open("test.txt", "w") as f:
2     f.write("python")
3
4 f = open("test.txt", "r")
5
6 print(f.read())
```

Exercise 200 - Files

Add the necessary code on lines 4 and 5 in order to delete the entire content of **test.txt**.

How to solve a coding exercise:

```
1 with open("test.txt", "w") as f:
2     f.write("python")
3
4
5
6 f = open("test.txt", "r")
7
8 print(f.read())
```

Exercise 200 - Solution

```
1 with open("test.txt", "w") as f:
2     f.write("python")
3
4 f = open("test.txt", "r+")
5 f.truncate()
6
7 f = open("test.txt", "r")
8
9 print(f.read())
```

Exercise 201 - Regular Expressions

Write code on line 5 in order to match the word **Bitcoin** at the beginning of the string using the **match()** method.

```
1 import re
2
3 s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
      market cap of over $300B."
4
5 result =
6
7 print(result.group())
```

Exercise 201 - Solution

```
1 import re
2
3 s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
  of the current financial system. In 2017, the price of 1 BTC reached
  $20000, with a market cap of over $300B."
4
5 result = re.match("Bitcoin", s)
6
7 print(result.group())
```

Exercise 202 - Regular Expressions

Write code on line 5 in order to match the word **Bitcoin** at the beginning of the string using the **match()** method and ignoring the case. This way, no matter if you have **bitcoin** or **Bitcoin**, the match is done either way.

```
1 import re
2
3 s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
      market cap of over $300B."
4
5
6
7 print(result.group())
```

Exercise 202 - Solution

```
1 import re
2
3 s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
  of the current financial system. In 2017, the price of 1 BTC reached
  $20000, with a market cap of over $300B."
4
5 result = re.match("Bitcoin", s, re.I)
6
7 print(result.group())
```

Exercise 203 - Regular Expressions

Write code on line 5 in order to match the words **Bitcoin was** at the beginning of the string using the **match()** method. Use the dot (**.**) belonging to regex syntax in your solution.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group())
```

Exercise 203 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.match(r"B.{6} .{3}", s)
6
7  print(result.group())
```

Exercise 204 - Regular Expressions

Write code on line 5 in order to match the year **2009** in the string using the **search()** method. Use the **\d** in your solution.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 204 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"(\d{4})\s", s)
6
7  print(result.group(1))
```

Exercise 205 - Regular Expressions

Write code on line 5 in order to match the year **2017** in the string using the **search()** method. Use the **\d** in your solution.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 205 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"(\d{4}),", s)
6
7  print(result.group(1))
```

Exercise 206 - Regular Expressions

Write code on line 5 in order to match the date **Jan 3rd 2009** in the string using the **search()** method. Use the **\d** in your solution.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 206 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"(.{3}\s\d\w\w\s\d{4})\s", s)
6
7  print(result.group(1))
```

Exercise 207 - Regular Expressions

Write code on line 5 in order to match **BTC** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 207 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"([A-Z]{3})", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 208 - Regular Expressions

Write code on line 5 in order to match **1 BTC** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 208 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"([0-9]\s[A-Z]{3})", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 209 - Regular Expressions

Write code on line 5 in order to match **$20000** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 209 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"(\$\d{5}),", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 210 - Regular Expressions

Write code on line 5 in order to match **$300B** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 210 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"(\$\d{3}[A-Z])\.", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 211 - Regular Expressions

Write code on line 5 in order to match **market cap of** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 211 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.search(r"\s(.{6} .{3} .{2})\s", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result.group(1))
```

Exercise 212 - Regular Expressions

Write code on line 5 in order to match **184,073,529,068** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 212 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r"\$(\d{3},[0-9]{3},\d{3},[0-9]{3}),", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 213 - Regular Expressions

Write code on line 5 in order to match **10,259.02** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 213 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r"\$(\d{1,3},\d{1,3}\.\d{1,3}),", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 214 - Regular Expressions

Write code on line 5 in order to match **17,942,600 BTC** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 214 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r"\s([0-9]{2},[0-9]{3},[0-9]{3}\s.{3}),", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 215 - Regular Expressions

Write code on line 5 in order to match **24h: 0.10%** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 215 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r"\s(.{4}\s\d\.\d\d%)", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 216 - Regular Expressions

Write code on line 5 in order to match **Volume 24h: $15,670,986,269** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 216 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r"\.\d\d, (.{1,}:\s\$\d{2,},\d{2,},\d{2,},\d{2,}), ",
   s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 217 - Regular Expressions

Write code on line 5 in order to match **Circulating Supply: 17,942,600 BTC** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 217 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r"(\w+ \w+: \d{2}.+? [A-Z]{3}), ", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 218 - Regular Expressions

Write code on line 5 in order to match **259.02, V** in the string using the **search()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 218 - Solution

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume
   24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h:
   0.10%"
4
5  result = re.search(r",([0-9]{3}\.[0-9]{2},\s.)", s)
6
7  print(result.group(1))
```

```
1  import re
2
3  s = "Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result =
6
7  print(result.group(1))
```

Exercise 219 - Regular Expressions

Write code on line 5 in order to match all the **years** in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 219 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.findall(r"\s(\d{4})", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 220 - Regular Expressions

Write code on line 5 in order to match all the **numbers** (3, 2009, 2017 etc.) in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 220 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.findall(r"\d{1,}", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 221 - Regular Expressions

Write code on line 5 in order to match all the **three-letter words** in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 221 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.findall(r"\s(\w{3})\s", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 222 - Regular Expressions

Write code on line 5 in order to match all the **words starting with an uppercase letter** in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 222 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.findall(r"([A-Z]{1}.+?)\s", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 223 - Regular Expressions

Write code on line 5 in order to match all the **two-letter words starting with the letter o** in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 223 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
of the current financial system. In 2017, the price of 1 BTC reached
$20000, with a market cap of over $300B."
4
5  result = re.findall(r"\s(o.{1})\s", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 224 - Regular Expressions

Write code on line 5 in order to match all the **words that have at least 8 characters** in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 224 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.findall(r"\w{8,}", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 225 - Regular Expressions

Write code on line 5 in order to match all the **words starting with a or c and that have at least 3 letters** in the string using the **findall()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B."
4
5  result =
6
7  print(result)
```

Exercise 225 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B."
4
5  result = re.findall(r"\s([ac]\w{2,})\s", s)
6
7  print(result)
```

Exercise 226 - Regular Expressions

Write code on line 5 in order to replace all the **years** in the string with **XXXX** using the **sub()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 226 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B. Bitcoin, Market Cap:
   $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269,
   Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result = re.sub(r"\s\d{4}", " XXXX", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B. Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC,
       Change 24h: 0.10%"
4
5  result =
6
7  print(result)
```

Exercise 227 - Regular Expressions

Write code on line 5 in order to replace each **floating-point number** in the string (10,259.02 and 0.10) with a dot (**.**) using the **sub()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 227 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B. Bitcoin, Market Cap:
   $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269,
   Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result = re.sub(r"\d{1,},*\d*\.\d{1,}", ".", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B. Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC,
       Change 24h: 0.10%"
4
5  result =
6
7  print(result)
```

Exercise 228 - Regular Expressions

Write code on line 5 in order to replace all occurrences of **BTC** in the string with **Bitcoin** using the **sub()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 228 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
of the current financial system. In 2017, the price of 1 BTC reached
$20000, with a market cap of over $300B. Bitcoin, Market Cap:
$184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269,
Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result = re.sub(r"[A-Z]{3}", "Bitcoin", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B. Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC,
       Change 24h: 0.10%"
4
5  result =
6
7  print(result)
```

Exercise 229 - Regular Expressions

Write code on line 5 in order to replace all the **digits less than or equal to 5** in the string with **8** using the **sub()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 229 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
   of the current financial system. In 2017, the price of 1 BTC reached
   $20000, with a market cap of over $300B. Bitcoin, Market Cap:
   $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269,
   Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result = re.sub(r"[0-5]", "8", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B. Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC,
       Change 24h: 0.10%"
4
5  result =
6
7  print(result)
```

Exercise 230 - Regular Expressions

Write code on line 5 in order to replace all the **words starting with an uppercase letter *or* digits greater than or equal to 6** in the string with **W** using the **sub()** method.

Note! *There may be multiple ways to write the pattern. Write it however you want, what's important is the result and that gets verified by the Python 3 engine behind the exercise. If your pattern is correct and returns the expected string, then hitting* ***Check Solution*** *is going to verify if your code returns what it should.* ***Do not*** *use the words themselves as patterns for matching, since that would defeat the purpose of the exercise. Use the regex syntax you learned in the course and also Python's documentation found* ***here****. The solution that I provide after the exercise may be just one of the ways to solve the task.*

Advice! *If your solution works and it's different than the one I provide, then study and try to understand my solution as well.*

Exercise 230 - Solution

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure
of the current financial system. In 2017, the price of 1 BTC reached
$20000, with a market cap of over $300B. Bitcoin, Market Cap:
$184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269,
Circulating Supply: 17,942,600 BTC, Change 24h: 0.10%"
4
5  result = re.sub(r"[A-Z]\w{1,}|[6-9]", "W", s)
6
7  print(result)
```

```
1  import re
2
3  s = "Bitcoin was born on Jan 3rd 2009 as an alternative to the failure of the current financial system. In 2017, the price of 1 BTC reached $20000, with a
       market cap of over $300B. Bitcoin, Market Cap: $184,073,529,068, Price: $10,259.02, Volume 24h: $15,670,986,269, Circulating Supply: 17,942,600 BTC,
       Change 24h: 0.10%"
4
5  result =
6
7  print(result)
```

Exercise 231 - Classes

Write a class called **ClassOne** starting on line 1 containing:

- The **__init__** method with two parameters **p1** and **p2**. Define the corresponding attributes inside the **__init__** method.
- A method called **square** that takes one parameter **p3** and prints out the value of **p3** squared.

```
1  class
2
3  p = ClassOne(1, 2)
4  print(type(p))
```

Exercise 231 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10 print(type(p))
```

Exercise 232 - Classes

Considering the **ClassOne** class and the **p** object, write code on line 11 in order to access the **p1** attribute for the current instance of the class and print its value to the screen.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11
```

Exercise 232 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11 print(p.p1)
```

Exercise 233 - Classes

Considering the **ClassOne** class and the **p** object, write code on line 11 in order to call the **square()** method for the current instance of the class using **10** as an argument and print the result to the screen.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11
```

Exercise 233 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11 p.square(10)
```

Exercise 234 - Classes

Considering the **ClassOne** class and the **p** object, write code on line 11 in order to set the value of the **p2** attribute to **5** for the current instance of the class, **without** using a function.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11
12
13 print(p.p2)
```

Exercise 234 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11 p.p2 = 5
12
13 print(p.p2)
```

Exercise 235 - Classes

Considering the **ClassOne** class and the **p** object, write code on lines 11 and 12 in order to set the value of the **p2** attribute to **50** for the current instance of the class using a function, and then get the new value of **p2,** again using a function, and print it out to the screen as well.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
```

Exercise 235 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11 setattr(p, 'p2', 50)
12 print(getattr(p, 'p2'))
```

Exercise 236 - Classes

Considering the **ClassOne** class and the **p** object, write code on line 11 in order to check if **p2** is an attribute of **p**, using a function, also printing the result to the screen.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11
```

Exercise 236 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11 print(hasattr(p, 'p2'))
```

Exercise 237 - Classes

Considering the **ClassOne** class and the **p** object, write code on line 11 to check if **p** is indeed an instance of the **ClassOne** class, using a function, also printing the result to the screen.

exercise.py

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11
```

Exercise 237 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  p = ClassOne(1, 2)
10
11 print(isinstance(p, ClassOne))
```

Exercise 238 - Classes

Considering the **ClassOne** class, write code starting on line 9 to create a child class called **ClassTwo** that inherits from **ClassOne** and also has its own method called **times10()** that takes a single parameter **x** and prints out the result of multiplying **x** by 10.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9
10
11
12 y = ClassTwo(10, 20)
13 print(y.p1)
```

Exercise 238 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  class ClassTwo(ClassOne):
10     def times10(self, x):
11         print(x * 10)
12
13 y = ClassTwo(10, 20)
14 print(y.p1)
```

Exercise 239 - Classes

Considering the **ClassOne** and **ClassTwo** classes, where the latter is a child of the former, write code on line 15 in order to call the **times10()** method from the child class having **x** equal to 45, also printing the result to the screen.

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  class ClassTwo(ClassOne):
10     def times10(self, x):
11         return x * 10
12
13 obj = ClassTwo(15, 25)
14
15
```

Exercise 239 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  class ClassTwo(ClassOne):
10     def times10(self, x):
11         return x * 10
12
13 obj = ClassTwo(15, 25)
14
15 print(obj.times10(45))
```

Exercise 240 - Classes

Considering the **ClassOne** and **ClassTwo** classes, write code on line 13 to verify that **ClassTwo** is indeed a child of **ClassOne**, also printing the result to the screen.

exercise.py

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  class ClassTwo(ClassOne):
10     def times10(self, x):
11         return x * 10
12
13
```

Exercise 240 - Solution

```
1  class ClassOne(object):
2      def __init__(self, p1, p2):
3          self.p1 = p1
4          self.p2 = p2
5
6      def square(self, p3):
7          print(p3 ** 2)
8
9  class ClassTwo(ClassOne):
10     def times10(self, x):
11         return x * 10
12
13 print(issubclass(ClassTwo, ClassOne))
```

Exercise 241 - Other Concepts

Write a list comprehension on line 1 that will iterate over **range(1, 5)** and return a list of its elements.

```
1 cph =
2
3 print(cph)
```

Exercise 241 - Solution

```
1 cph = [i for i in range(1, 5)]
2
3 print(cph)
```

Exercise 242 - Other Concepts

Write a list comprehension on line 1 that will iterate over **range(1, 15, 5)** and return a list of its elements squared.

```
1 cph =
2
3 print(cph)
```

Exercise 242 - Solution

```
1 cph = [i ** 2 for i in range(1, 15, 5)]
2
3 print(cph)
```

Exercise 243 - Other Concepts

Write a list comprehension on line 1 that will iterate over **range(5, 25, 3)** and return a list of its elements squared only for the elements that are less than or equal to 16.

```
1  cph =
2
3  print(cph)
```

Exercise 243 - Solution

```
1  cph = [i ** 2 for i in range(5, 25, 3) if i <= 16]
2
3  print(cph)
```

Exercise 244 - Other Concepts

Write a dictionary comprehension on line 1 that will iterate over **range(9)** and return a dictionary of key-value pairs where the value is equal to the key times 3.

```
1  cph =
2
3  print(cph)
```

Exercise 244 - Solution

```
1  cph = {x: x * 3 for x in range(9)}
2
3  print(cph)
```

Exercise 245 - Other Concepts

Write a set comprehension on line 1 that will iterate over **range(10, 19)** and return a set of its elements divided by 2.5.

```
1  cph =
2
3  print(cph)
```

Exercise 245 - Solution

```
1  cph = {x / 2.5 for x in range(10, 19)}
2
3  print(cph)
```

Exercise 246 - Other Concepts

Write a lambda function on line 1 that takes two parameters **x** and **y** and multiplies **x** with **y**.

```
1 lam =
2
3 print(lam(2, 5))
```

Exercise 246 - Solution

```
1 lam = lambda x, y: x * y
2
3 print(lam(2, 5))
```

Exercise 247 - Other Concepts

Write a lambda function on line 1 that takes a list **list1** as a parameter, and multiplies each element of **range(1, 5)** with each element of **list1** using a list comprehension.

```
1 lam =
2
3 print(lam([1, 2]))
```

Exercise 247 - Solution

```
1 lam = lambda list1: [x * y for x in range(1, 5) for y in list1]
2
3 print(lam([1, 2]))
```

Exercise 248 - Other Concepts

Use the correct function from the **itertools** module on line 6, in between the parentheses of **list()**, in order to concatenate **list1** and **list2**.

exercise.py

```
1  import itertools
2
3  list1 = [1, 2, 3]
4  list2 = [4, 5]
5
6  result = list()
7
8  print(result)
```

Exercise 248 - Solution

```
1  import itertools
2
3  list1 = [1, 2, 3]
4  list2 = [4, 5]
5
6  result = list(itertools.chain(list1, list2))
7
8  print(result)
```

Exercise 249 - Other Concepts

Use the correct function from the **itertools** module with a **for** loop and a nested **if/else** block in order to return all the numbers starting at 20 and up to 31 with a step of 2. Be careful not to end up with an infinite loop!

```
1  import itertools
2
3
```

Exercise 249 - Solution

```
1  import itertools
2
3  for i in itertools.count(20, 2):
4      if i < 31:
5          print(i)
6      else:
7          break
```

Exercise 250 - Other Concepts

Use the correct function from the **itertools** module on line 5, in between the parentheses of **list()**, in order to return the elements for which the lambda function given as an argument returns **False**.

```
1  import itertools
2
3  lam = lambda x: x < 5
4
5  result = list( (lam, range(10)))
6
7  print(result)
```

Exercise 250 - Solution

```
1  import itertools
2
3  lam = lambda x: x < 5
4
5  result = list(itertools.filterfalse(lam, range(10)))
6
7  print(result)
```

Find()
Swapcase()
Startswith()
Replace(‘what’,‘with what’)
Split(‘,’)
print('&'.join(my_string))
print(my_string.title())
print(my_string.format('2010','10k','Bitcoin’))

```
print(my_string % ("2010", "10k", "Bitcoin"))
```

print(my_string[3:7])
print(my_string[-23:-16])
print(my_string[:12])
print(my_string[-9:])
print(my_string[::-1]) reverse string
print(my_string[::7])
print(my_string[10:])
print(my_string[:-4])
num3 = pow (num1,num2) (#Using pow() function to calculate num1 raised to the power of num2)
num3 = num1 ** num2

num1 = -11
num2 = abs(num1)
Num 1 == num2

When you apply the bool() function to an integer in Python, it converts the integer to a Boolean value. In Python, any numeric value that is not zero is considered True, and zero is considered False when converted to a Boolean.

my_list.remove('Java’)
my_list.pop(5)

my_list.add('Java’)
my_list.append(‘Java’)

index = my_list.index(10.5)
my_list.insert(4,77) at index 4, inserting element 77

my_list.extend([100,101,102]) ‘extend’ to add list to another list at end

howmany = my_list.count(20)

asc = sorted(my_list)
asc = sorted(my_list, reverse=True) to sort in descending order

small = min(my_list)
large = max(my_list)

add = sum(my_list) to get sum of all elements in list

my_list.clear() to delete all elements in list and obtain empty list

add = (my_list + [30.01, 30.02, 30.03])*2 add the elements of **[30.01, 30.02, 30.03]** to **my_list** and multiply the resulting list by **2**.

element = my_list[2] to get element at index 2
element = my_list.index(20) to get index of element 20
element = my_list[-2]
my_slice = my_list[-4:-1]

my_slice = my_list[:-4] except last 4
my_slice = my_list[-4:] except first 3

my_slice = my_list[::3] return every third element

---

return every fourth element of **my_list** starting with the last element of the list.
my_slice = my_list[::-4]
[::-4]: This slicing operation selects elements from the list starting from the end (indicated by the first :), moving backward (indicated by the negative step -1), and selecting every fourth element (indicated by the second :4).

---

my_frozenset = frozenset(my_set)
frozenset is an immutable collection of unique elements, similar to a set. The main difference between a frozenset and a regular set is that a frozenset is immutable, meaning once it is created, its elements cannot be changed or modified.

---

to verify if element **10** is in **my_set**.
check = 10 in my_set

---

my_set.add(19) list→ append, set→ add

append() and pop(): Only for lists.
add() and remove(): Only for sets.

find out the common elements of my_set1 and my_set2.
common = my_set1.intersection(my_set2)

in order to join the elements of **my_set1** and **my_set2**.
join = my_set1.union(my_set2)

to find out the elements of **my_set2** that are not members of **my_set1**.
diff = my_set2.difference(my_set1)

index = my_tup.index('Slovakia’) str type should add quotes

```
#tuple unpacking
my_tup = ("Romania", "Poland", "Estonia")
(ro,po,es) = my_tup
print(ro + ", " + po + ", " + es) #returns 'Romania, Poland, Estonia'
```

to find out the last element of **my_tup** in alphabetical order.
The max() function in Python returns the maximum element from a sequence, such as a list, tuple, or string. When used with a tuple like my_tup, it returns the element that comes last when sorted lexicographically
last = max(my_tup)

return all the elements of **my_tup**, except the first two of them, using a negative index

```
my_slice = my_tup[-5:]
```

return all the elements of **my_tup**, except the last three of them, using a negative index.

```
my_slice = my_tup[:-3]
```

```
my_range = range(-75, -25, 15 )
print(list(my_range))  # range takes difference between no.s, no negative sign for third element in range
Output: [-75, -60, -45, -30]
```

```
1.my_range = range(-25, 139, 30)
2.
3.print(list(my_range)) #[-25, 5, 35, 65, 95, 125]
```

return the value associated with key **4**

```
1.crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2.
3.value = crypto[4]
4.
5.print(value) # Stellar

6.value = crypto.get(4)   # now using method as solution.
```

update the value associated with key **4** to **"Cardano"**.
crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}

```
crypto[4] = "Cardano"
```

---

add a new key-value pair to the dictionary: **6: "Monero“**

```
1.crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2.
3.crypto[6] = "Monero"
4.
5.print(crypto[6])
```

---

return the number of key-value pairs in the dictionary.
crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}

```
number = len(crypto)

print(number)
```

in order to delete the key-value pair associated with key **3**

```
1.crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2.
3.del crypto[3]
4.
5.print(crypto)

6.crypto.pop(3)   # using method as a solution
```

in order to verify that **7** is **not** a key in the dictionary.

```
check = 7 not in crypto
```

delete all the elements in the dictionary.

```
crypto.clear()
```

1.in order to get a list of tuples, where each tuple represents a key-value pair in the dictionary.

```
crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2.
3.result = crypto.items()
4.
5.print(list(result))
```

.items() is a method in Python that is used to return a view object that displays a list of a dictionary's key-value tuple pairs. When called on a dictionary, it returns a view object that provides a dynamic view of the dictionary's entries.

in order to get the sum of all the keys in the dictionary

```
add = sum(crypto)
```

---

in order to get a list of all the values in the dictionary.

```
1.crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2.
3.val = crypto.values()
4.
5.print(list(val))
```

---

in order to get the smallest key in the dictionary.

```
key = min(crypto)
key = min(crypto.keys())# use both
```

---

in order to return and remove an arbitrary key-value pair from the dictionary.

```
1.crypto = {1: "Bitcoin", 2: "Ethereum", 3: "Litecoin", 4: "Stellar", 5: "XRP"}
2.
3.crypto.popitem()
4.
5.print(len(crypto))
```

```
1.value = '0b1010'
2.
3.conv = int(value, 2)
4.
5.print(conv)
```

#Converting binary to decimal notation
#Convert binary string to integer

```
1.value = '0xa'
2.
3.conv = int(value, 16)
4.
5.print(conv)
```

convert **value** from hexadecimal to decimal notation.

```
if (type(x) == str) and (x[0] == 'T’):
print("True!")
```

Or

```
1.if type(x) is str and x.startswith("T"):
2.print("True!")
```

Both methods can be used

```
if 'z' in x or x.count('y')>=2:
    print("True!")
```

```
1.if x.index("f") < 10:
2.print("True!")
3.else:
4.print("False!")
```

```
if x[-3:].isdigit():
    print("True!")
else:
    print("False!")
```

if the last 3 characters of the string are all digits

```
if len(x) >= 8 and type(x[6])==float:
    print("True!")else:    print("False!")
```

```
1.if x[3][1].endswith("h") and x[7][0].endswith("h"):
2.print("True!")
3.else:
4.print("False!")
```

Syntax to focus: endswith("h")

```
if x[5]=='perl' or len(x) % 5 <= 2:
    print('True!')
```

value associated with key number **5** is **Perl**

```
if (3 in x) and (min(x.values()) == 'C#'):
    print("True!")
```

Or

```
1.if 3 in x and sorted(x.values())[0] == "C#":
2.print("True!")
```

```
if max(x.values())[-1] == 'n’:
    print("True!")
```

last character of the largest (alphabetically) value in the dictionary is **n**

Or

```
1.if sorted(x.values())[-1][-1] == "n":
2.print("True!")
```

```
if ( sorted(x.keys())[-1] ) % ( sorted(x.keys())[-2] ) == (sorted(x.keys())[0]):    print("True!")
```

---

sum of all the keys in the dictionary is less than the number of characters of the string obtained by concatenating the values associated with the first 5 keys in the dictionary

```
if sum(x.keys()) < len((x[1] + x[2] + x[3] + x[4] + x[5])):
    print('True!')
```

- **elif** executes only if the preceding **if** statement is false.
- If the **if** statement is true, the corresponding **elif** and **else** statements are skipped.
- If none of the conditions are true, the **else** block (if present) executes.

In Python, the sorted() function can work with various iterable types, including lists, tuples, and strings. However, it cannot directly be applied to integers, as they are not iterable objects

---

Use keywords… in, not in, is, is not, endswith, startswith

```
x = "cryptocurrency“
y = "blockchain“
for i in x:
    if i in y:
        print(i*2)
```

```
1.for i in range(5,25,5):
2.if 10 <= i <= 21:
3.print(i * range(5,25,5)[-2])
4.else:
5.print("Outside!")
6.else:
7.print("The end!")
```

```
1.x = 5
2.
3.while x <= 11:
4.if x == 10:
5.print("x is 10!")
6.x = x + 1
7.else:
8.print(x * 11)
9.x = x + 1
```

```
try:
    print(25 % 0 ** 5 + 5)
except:
    print("Bug!")
finally:
    print("Result!")
```

```
def my_func(x):
    return x*10

result=my_func(7)
print(result)
```

```
1.def my_func(x):
2.my_new_list = []
3.for i in range(5):
4.my_new_list.append(i * x)
5.return my_new_list
6.
7.result = my_func(2)
8.print(result)
```

```
def my_func(x):
    return len(x) * sorted(x.keys())[-1]

result = my_func({1: 3, 2: 3, 4: 5, 5: 9, 6: 8, 3: 7, 7: 0})
print(result)
```

```
def my_func(x, **kwargs):
    return x * sorted(kwargs.values())[-1]

result = my_func(10, val1 = 10, val2 = 15, val3 = 20, val4 = 25, val5 = 30)
print(result)
```

```
def my_func(x, *args):
    return x * args[1]

result = my_func(5, 10, 20, 30, 50)
print(result)
```

```
var = 8

def my_func(x):
    global var
    print(x * var)
    var = 12

my_func(10)
```

.

```
from math import pi
print(f'{pi:.4f}')
```

```
f = open("test.txt", "r")

print(f.read())
```

```
f = open("test.txt", "r")

print(f.readlines())
```

To get output in list format

```
f = open("test.txt", "r")

f.read()

f.seek(0)

print(f.read())
```

bring back the cursor at the very beginning of **test.txt** before reading from the file once again.

```
f = open("test.txt", "r")

f.read(5)

print(f.tell())
```

in order to get the current position of the cursor inside test.txt

```
f = open("test.txt", "r")

f.read(5)

print(f.mode)
```

in order to get the current mode in which test.txt is open (read, write etc.)

```
f = open("test.txt", "a+")

print(f.mode)
```

order to open test.txt for appending and reading at the same time.

```
f = open("test.txt", "w")

f.write("python")
f.close()

f = open("test.txt", "r")

print(f.read())
```

write the string python to test.txt

```
f = open("test.txt", "w")

f.writelines(['python', ' ', 'and', ' ', 'java'])
f.close()

f = open("test.txt", "r")

print(f.read())
```

write a list of strings ['python', ' ', 'and', ' ', 'java'] to test.txt

```
with open("test.txt", "w") as f:
    f.write("python")

f = open("test.txt", "r")

print(f.read())
```

to write the string python and also close test.txt

```
with open("test.txt", "w") as f:
    f.write("python")

f = open("test.txt", "r+")
f.truncate()

f = open("test.txt", "r")

print(f.read())
```

order to delete the entire content of test.txt.