DSO 545: Statistical Computing and Data Visualization

Abbass Al Sharif

Fall 2019

Lab 1: Python Variables and Lists

Contents

---

1. Python for Computations

1. Perform the some computations in Python using the arithematic signs $+, -, /, *, **$,

```
1
```

```
## 1
```

```
1 + 5
```

```
## 6
```

```
4/2
```

```
## 2.0
```

```
4**2
```

```
## 16
```

```
12%5
```

```
## 2
```

2. Variables in Python

2. Create three variables: one to store your height in meters (height), weight in kilograms (weight), and body mass index (bmi).

```
height = 1.80 #in meters
print(height)
```

```
## 1.8
```

```
height
```

```
## 1.8
```

```
weight  = 71 #in Kg
print(weight)
```

```
## 71
```

```
bmi = weight/(height**2)
print(bmi)
```

```
## 21.91358024691358
```

3. Using the code you created in the previous question, calculate the bmi for a person whose weight is 71Kg and height is 2m.

```
height = 2 #in meters
weight  = 71 #in Kg
bmi = weight/(height**2)
print(bmi)
```

```
## 17.75
```

4. Create a variable called x, and store a value of 5 in it.

```
x = 5
x
```

```
## 5
```

5. Update the stored value of the x variable that was defined in the previous question by incrementing it by 1.

```
x = x + 1
x
```

```
## 6
```

```
x = 5
x +=1 # similar to x = x + 1
x
```

```
## 6
```

```
x -= 1 #similar to x = x - 1
x
```

```
## 5
```

```
x *=2 #similar to x = x*2
x
```

```
## 10
```

6. Define three variables x, y, and z, and assign the values of 45, “Marshall”, and “DSO 545” to each.

```
x, y, z = 45, "Marshall", "DSO 545"

x
```

```
## 45
```

```
y
```

```
## 'Marshall'
```

```
z
```

```
## 'DSO 545'
```

3. Types of Data or Variables

7. What type of data structures do the x and bmi variables have? What about the type of the constant "DSO 545"?

```
type(x)
```

```
## <class 'int'>
```

```
type(bmi)
```

```
## <class 'float'>
```

```
type("DSO 545")
```

```
## <class 'str'>
```

8. Define a variable x with value of 2.5. Try to convert the data type of x to integer. Try to convert/cast it to a string.

```
x = 2.5
type(x)
```

```
## <class 'float'>
```

```
int(x)
```

```
## 2
```

```
type(x)

# we need to "override" the original content in object x
```

```
## <class 'float'>
```

```
x = int(x)
type(x)
```

```
## <class 'int'>
```

```
type(x)
```

```
## <class 'int'>
```

```
y= str(x)
y
```

```
## '2'
```

```
type(y)
```

```
## <class 'str'>
```

4. Python "List" Data Structure

9. Create a list of the following numbers: 1, 4, 9, 25, 36, 49, 64. Name this list `squares`.

```
squares = [1, 4, 9, 25, 36, 49, 64]
squares
```

```
## [1, 4, 9, 25, 36, 49, 64]
```

```
type(squares)
```

```
## <class 'list'>
```

10. The previous list was homogenous, i.e. all elements were of the same type. Create a mixed list of numbers and strings for the following: "one", 1, "four", 4, "nine", 9, "twenty five", 25. Name this list `mixed_list`.

```
mixed_list = ["one", 1, "four", 4, "nine", 9, "twenty five", 25]
mixed_list
```

```
## ['one', 1, 'four', 4, 'nine', 9, 'twenty five', 25]
```

```
type(mixed_list)
```

```
## <class 'list'>
```

11. Given that lists allow store to have any data type/structure, list's try to create a list of lists! Assign each string and numeric value in the previous question to a list, i.e. ['one', 1] etc. Store all of the sublists in one list and name it `list_of_lists`.

```
list_of_lists = [["one", 1], ["four", 4], ["nine", 9], ["twenty five", 25]]
list_of_lists
```

```
## [['one', 1], ['four', 4], ['nine', 9], ['twenty five', 25]]
```

```
type(list_of_lists)
```

```
## <class 'list'>
```

5. Accessing Elements of Lists in Python

12. Access and print the first element of the `squares` list defined earlier.

```
squares[0] # first element in the squares list
```

```
## 1
```

13. Access and print the third and sixth element in the `squares` list.

```
squares[2] # access the third element
```

```
## 9
```

```
squares[5] # access the sixth element
```

```
## 49
```

14. Access and print the last element in the list `squares`.

```
squares[-1] #access the last element in the list of squares
```

```
## 64
```

```
squares[-2] #access the second to last element in the list of squares
```

```
## 49
```

List Slicing: We can access a range of items in a list by using the "colon" slicing operator (:). The following rules apply to slicing (https://www.geeksforgeeks.org/python-list/):

List Slicing	Code
To print elements from beginning to a range	`[:Index]`
To print elements from specific `Index` till the end	`[Index:]`
To print elements within a range	`[Start Index:End Index]`
To print every other nth element in the list	`[::n]`
To print whole List in reverse order	`[::-1]`

15. Access and print the first 3 elements in the `squares` list.

```
squares[:3]
```

```
## [1, 4, 9]
```

16. Access and print the all elements in the `squares` list starting at 5th element.

```
squares[4:]
```

```
## [36, 49, 64]
```

17. Access and print the all elements in the `squares` list starting at 3rd element, and ending at the 5th element.

```
squares[2:5]
```

```
## [9, 25, 36]
```

18. Access and print every other 3rd element in the list.

```
squares[::3]
```

```
## [1, 25, 64]
```

19. Access and print all elements of the list in reverse order.

```
squares[::-1]
```

```
## [64, 49, 36, 25, 9, 4, 1]
```

20. Access and print every other (second) element of the list in reverse order:

```
squares[::-2]
```

```
## [64, 36, 9, 1]
```

21. Define a list that contains the elements: 30, 60, 70, 100, 120. Update the third element to be 95.

```
x = [30, 60, 70, 100, 120]
x
```

```
## [30, 60, 70, 100, 120]
```

```
x[3] = 95
x
```

```
## [30, 60, 70, 95, 120]
```

22. Update the 2nd, 3rd, and 4th elements in x to 55, 65, and 95.

```
x = [30, 60, 70, 100, 120]
x
```

```
## [30, 60, 70, 100, 120]
```

```
x[1:4] = [55,65,95]
x
```

```
## [30, 55, 65, 95, 120]
```

6. Iterating over a Python List

Method 1: Using a For loop

23. Use a for loop to print all the elements in `list = [1, 3, 5, 7, 9]`.

```
list = [1, 3, 5, 7, 9] #define the list

# Using for loop
for i in list:
    print(i)
```

```
## 1
## 3
## 5
## 7
## 9
```

24. Create a list (`degrees = [0, 10, 20, 40, 100]`). Print the following output:

```
## list element: 0
## list element: 10
## list element: 20
## list element: 40
## list element: 100
```

```
## The degrees list has 5 elements
```

Method #2: For loop and `range()`

25. Use the `range()` fucntion in a for loop to print numbers from 0 to 5.

```
for i in range(6):
    print(i)
```

```
## 0
## 1
## 2
## 3
## 4
## 5
```

26. Use the `range()` function to print all the elements in (`list = [1, 3, 5, 7, 9]`).

```
# Python code to iterate over a list
list = [1, 3, 5, 7, 9]

# getting length of list
length = len(list)

# Iterating the index
# same as 'for i in range(len(list))'
for i in range(length):
    print(list[i])
```

```
## 1
## 3
## 5
## 7
## 9
```

Method #3: Using list comprehension

27. Use a list comprehension to print all the elements of (`list = [1, 2, 3, 4, 5, 6, 7, 8, 9]`).

```
# Python code to iterate over a list
list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Using list comprehension
[print(i) for i in list]
```

```
## 1
## 2
## 3
## 4
## 5
## 6
## 7
## 8
## 9
## [None, None, None, None, None, None, None, None, None]
```

28. Use a list comprehension to create a list of all the squared values of the list (`list = [1, 2, 3, 4, 5, 6, 7, 8, 9]`).

```
# Square the numbers in the list using list comprehension

squared = [i**2 for i in list]
print(squared)

# Multiply every item in the list by 3
```

```
## [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
[i*3 for i in list]
```

```
## [3, 6, 9, 12, 15, 18, 21, 24, 27]
```

29. Use a list comprehension to print all the even numbers in the list (`list = [1, 2, 3, 4, 5, 6, 7, 8, 9]`).

```
list = [1, 2, 3, 4, 5, 6, 7, 8, 9]

[i for i in list if i%2==0]
```

```
## [2, 4, 6, 8]
```

7. Selected Methods for Python Lists

Methods that are available with list object in Python programming are tabulated below (https://www.programiz.com/python-programming/list):

Method	Description
append()	Add an element to the end of the list
extend()	Add all elements of a list to the another list
insert()	Insert an item at the defined index
remove()	Removes an item from the list
clear()	Removes all items from the list
index()	Returns the index of the first matched item
count()	Returns the count of number of items passed as an argument
sort()	Sort items in a list in ascending order
reverse()	Reverse the order of items in the list

30. Add the element 150 to the end of list (`x = [30, 60, 70, 100, 120]`).

```
x = [30, 60, 70, 100, 120]
x
```

```
## [30, 60, 70, 100, 120]
```

```
x.append(150)
x
```

```
## [30, 60, 70, 100, 120, 150]
```

31. Add the list `[170, 180, 190]` to the end of the list (`x = [30, 60, 70, 100, 120]`).

```
x = [30, 60, 70, 100, 120]
x
```

```
## [30, 60, 70, 100, 120]
```

```
x.extend([170, 180, 190])
x
```

```
## [30, 60, 70, 100, 120, 170, 180, 190]
```

32. Insert the element 50 to the list (`x = [30, 60, 70, 100, 120]`). Intset this element directly after 30.

```
x = [30, 60, 70, 100, 120]
x
```

```
## [30, 60, 70, 100, 120]
```

```
x.insert(1, 50) # first argument is index and second argument is the value to insert
x
```

```
## [30, 50, 60, 70, 100, 120]
```

33. Delete the element 50 from teh list defined in the previous question.

```
x = [30, 50, 60, 70, 100, 120]
x
```

```
## [30, 50, 60, 70, 100, 120]
```

```
x.remove(50) # remove the first occurance of element with value of 50
x
```

```
## [30, 60, 70, 100, 120]
```

34. What is the index position of the elemnt 70 in the list (`x = [30, 50, 60, 70, 100, 120]`).

```
x = [30, 50, 60, 70, 100, 120]
x
```

```
## [30, 50, 60, 70, 100, 120]
```

```
x.index(70) # returns the index of 70 in the list
```

```
## 3
```

```
x # the list is not affected
```

```
## [30, 50, 60, 70, 100, 120]
```

35. How many times does the element 50 appears in the list (`x = [30, 50, 50, 50, 50, 60, 70, 100, 120]`).

```
x = [30, 50, 50, 50, 50, 60, 70, 100, 120]
x
```

```
## [30, 50, 50, 50, 50, 60, 70, 100, 120]
```

```
x.count(50) # there are 4 occurances of 50 in the list
```

```
## 4
```

36. Sort the elements of the list (`x = [60, 20, 90, 35, 15]`).

```
x = [60, 20, 90, 35, 15]
x.sort() # sorts in ascending order by default
x
```

```
## [15, 20, 35, 60, 90]
```

37. Sort the elements of the list (`x = [60, 20, 90, 35, 15]`) in reverse order.

```
x = [60, 20, 90, 35, 15]
x.sort(reverse= True) # sorts in descending order
x
```

```
## [90, 60, 35, 20, 15]
```

38. Print the elements of the list (`x = [60, 20, 90, 35, 15]`) in reverse order.

```
x = [60, 20, 90, 35, 15]
x
```

```
## [60, 20, 90, 35, 15]
```

```
x.reverse() #reverses the order of elements in x
x
```

```
## [15, 35, 90, 20, 60]
```

8. Python "Dictionaries" Data Structure

39. Define a dictionary data structure as follows: the keys for the dictionary represents countires (Peru, Armenia, Jordan, and Norway), and the values represent their capitals (Lima, Yervan, Aman, Oslo).

```
capitals = {

'Peru': 'Lima',
'Armenia': 'Yerevan',
'Jordan': 'Amman',
'Norway': 'Oslo'

}

print(capitals)
```

```
## {'Peru': 'Lima', 'Armenia': 'Yerevan', 'Jordan': 'Amman', 'Norway': 'Oslo'}
```

```
type(capitals)
```

```
## <class 'dict'>
```

```
capitals = dict([
             ('Peru','Lima'),
             ('Armenia','Yerevan'),
             ('Jordan', 'Amman'),
             ('Norway', 'Oslo')
             ])

print(capitals)
```

```
## {'Peru': 'Lima', 'Armenia': 'Yerevan', 'Jordan': 'Amman', 'Norway': 'Oslo'}
```

```
type(capitals)
```

```
## <class 'dict'>
```

40. Access and print the capitals of Peru and Jordan.

```
capitals['Peru']
```

```
## 'Lima'
```

```
capitals['Jordan']
```

```
## 'Amman'
```

Note that, you can also build the dictionary incrementally as follows:

```
person = {} #initialize the dictionary
type(person)
```

```
## <class 'dict'>
```

```
person['fname'] = 'Joe'
person['lname'] = 'Fonebone'
person['age'] = 51
person['spouse'] = 'Edna'
person['children'] = ['Ralph', 'Betty', 'Joey']
person['pets'] = {'dog': 'Fido', 'cat': 'Sox'}

print(person)
```

```
## {'fname': 'Joe', 'lname': 'Fonebone', 'age': 51, 'spouse': 'Edna', 'children': ['Ralph', 'Betty', 'J
```

We can access elements as follows:

```
person['fname']
```

```
## 'Joe'
```

```
person['children']
```

```
## ['Ralph', 'Betty', 'Joey']
```

What if we want to acces the third child in the list of children?

```
kids = person['children']
kids[2]

## OR
```

```
## 'Joey'
```

```
person['children'][2]
```

```
## 'Joey'
```

41. Create two lists seperately, one for the countries ('Peru','Armenia','Jordan','Norway'), and the other for the cities (Lima', 'Yerevan', 'Amman', 'Oslo'). Create a dictionary called `capitals` where the countries are the keys and cities are the values of the dictionary.

```
countries = ['Peru','Armenia','Jordan','Norway']
cities = ['Lima', 'Yerevan', 'Amman', 'Oslo']

capitals = dict(zip(countries, cities))
```