NEW SOFT
Training Center
مركز نيوسوفت للتدريب

Programming

with python™

By
Rami Tailakh
Senior Software Engineer and Data Science Practitioner
MSc in Applied Computing and Information Technology

Day-2 Agenda

- Day-1 Challenge Review
- Advanced Data Types
- Flow Control
- Python Functions

Day-1 Challenge Review-String Methods

Employing ONLY the string methods and operators find a solution of the following challenges:

1. Find if a given word is in lower/upper case

2. Find the length of a string

3. Find all occurrences of a word (from any text) in a given string.

Advanced Data Types-List

- A Python list can be seen as a collection
- It is to hold a sequence of values
- A Python list may hold different types of values
- A Python list is **mutable**
- Examples:

```
['red','green','blue']

[0, 1, 2]

['Sunday','Monday', 1, 2,2.5]
```

Advanced Data Types-List Access & Operations

- **Slicing a List**

```
>>> my_list = [85, 88, 90, 91, 100]
>>> my_list[1:4]
```

Out: **[88, 90, 91]**

```
>>> my_list = [85, 88, 90, 91, 100]
>>> my_list[0]
```

Out: **85**

Advanced Data Types-List Re-assigning


```
>>> my_list = [85, 88, 90, 91, 100]
>>> my_list.append(73)
>>> my_list
Out:        [85, 88, 90, 91, 100, 73]

>>> my_list[5] = 79
>>> my_list
Out:        [85, 88, 90, 91, 100, 79]
```

Advanced Data Types-List Functions


```
>>> my_list = [88, 85, 91, 90, 100]
>>> sum(my_list)
Out:    527
>>> max(my_list)
Out:    100
>>> len(my_list)
Out:    5
>>> sorted(my_list)
Out:    [85, 88, 90, 91, 100]
```

Advanced Data Types-List Methods


```
>>> my_list = [88, 85, 90, 91, 100]
>>> my_list.index(100)
Out:    4
>>> my_list.count(100)
Out:    1
>>> a=[1,3,5,3,4]
>>> a.reverse()
>>> a
Out:    [4, 3, 5, 3, 1]
```

Advanced Data Types-List Operations

- **Concatenation of Python Lists**

```
>>> a, b = [0, 1], [2, 3]
>>> a + b
```

Out: **[0, 1, 2, 3]**

```
>>> a
```

Out: **[0, 1]**

```
>>> b
```

Out: **[2, 3]**

Note that the result is not assigned to a variable, so, no changes made on the variables **a** and **b.**

Advanced Data Types-List Operations

- **Multiplication**

  >>> a = [0, 1]

  >>> a * 2

  Out: **[0, 1, 0, 1]**

- **Membership**

  >>> 1 in a

  Out: **True**

  >>> 2 in a

  Out: **False**

Advanced Data Types-Tuple

- Python Tuples are like a list
- A Python tuple may hold different types of values
- A Python list is **immutable**
- Examples:

```
(1,'Adam','adam@google.com', 2301.25)
```

```
(0, 1, 2, 3)
```

- Packing; tuples can also be created without parentheses

```
>>> a = 1,'Adam','adam@google.com', 2301.25
```

Advanced Data Types-Tuple Unpacking


```
>>> employee = (1,'Adam',2301.25)
>>> id, name, salary = employee
>>> name
```

Out: **'Adam'**

Advanced Data Types-Tuple Re-assigning


```
>>> employee = (1,'Adam',2301.25)
```

```
>>> employee[2] = 2501.50
```

Out: TypeError: 'tuple' object does not support item assignment

But, what if a tuple has an item as a lits.

```
>>> t = (0, 1, 2, [3, 5])
```

```
>>> t[3][1] = 4
```

```
>>> t
```

Out: **(0, 1, 2, [3, 4])**

Advanced Data Types-Tuple Operations

- **Concatenation**

```
>>> employee = (1,'Adam',2301.25)
>>> a, b = (1,2,3),(4,5,6)
>>> a + b
```

Out: **(1, 2, 3, 4, 5, 6)**

```
>>> a = a + (4,5,6)
>>> a
```

Out: **(1, 2, 3, 4, 5, 6)**

Advanced Data Types-Tuple Operations

- **Membership**

```
>>> employee = (1,'Adam',2301.25)

>>> 'Adam' in employee
```

Out: **True**

- **Identity**

```
>>> a =(1,2)


>>> (1,2) is a
```

Out: **False**

Advanced Data Types-Set

- A Python set holds a sequence of values
- A Python set does not support indexing
- A Python set also cannot contain duplicate elements
- A set may contain values of different types
- A Python set is **mutable**
- Examples:

```
{0, 1, 2, 3}

{'Sunday','Monday', 1, 2,2.5}
```

Advanced Data Types-Set Updating


```
>>> n={0, 1, 2, 4}
>>> n[3] = 3
```

Out: TypeError: 'set' object does not support item assignment

Solution:

```
>>> n.remove(4)
>>> n.add(3)
>>> n
```

Out: **{0, 1, 2, 3}**

Advanced Data Types-Dictionary

- A Python dictionary is a collection
- It is unordered, **mutable,** and indexed.
- Python dictionaries have keys and values.
- Values may hold different types of values
- Examples:

```
{"id": 1,
 "name": "Adam",
 "salary": 4564.75}
```

Advanced Data Types-Dictionary Accessing

```
>>> employee = {"id": 1,
                "name": "Adam",
                "salary": 4564.75}

>>> employee["name"]
```

Out: **'Adam'**

Advanced Data Types-Dictionary Re-assigning

- **Updating a value of an existing key**

```
>>> employee = {"id": 1,
                "name": "Adam",
                "salary": 4564.75}
>>> employee["salary"] = 4620.50
>>> employee
```

Out: **{"id": 1, "name": "Adam", "salary": 4620.50}**

Advanced Data Types-Dictionary Re-assigning

- **Adding a new key**

```
>>> employee["email"] = "adam@gmail.com"

>>> employee
```

Out: **{"id": 1, "name": "Adam", "salary": 4620.50, "email": "adam@gmail.com"}**

Advanced Data Types-Dictionary Re-assigning

- **Deleting an existing key-value pair**

```
>>> del employee["email"]

>>> employee

Out:    {"id": 1, "name": "Adam", "salary": 4620.50}
```

Advanced Data Types-Dictionary Functions

```
>>> len(employee)
```

Out: 3

Advanced Data Types-Dictionary Methods


```
>>> employee.keys()
Out:    dict_keys(['id', 'name', 'salary'])
>>> employee.values()
Out:    dict_keys([1, 'Adam', 4620.5])
>>> employee.items()
Out:    dict_items([('id', 1), ('name', 'Adam'),
('salary', 4620.5)])
```

Advanced Data Types-Dictionary Operations

- **Membership**

```
>>> 'name' in employee
```

Out: **True**

```
>>> 'Adam' in employee
```

Out: **False**

Advanced Data Types-Nested Dictionaries


```
>>> employees = {
 "employee-1" : {
          "name": "Adam",
          "job": "manager"
 },
 "employee-2" : {
          "name": "Sami",
          "job": "engineer"
 }
}
```

Conditional Statements-if Statements

```
a = 2

if a > 0:

    print('{} is a positive number'.format(a))
```

Out: **2 is a positive number**

But, what if a = -2?

Conditional Statements-if/else Statements


```
a = -2
if a > 0:
    print('{} is a positive number'.format(a))
else:
    print('{} is a negative number'.format(a))
```

Out: **-2 is a negative number**

Conditional Statements-if/elif/else Statements

```
a = 2

if a > 0:

    print('Positive')

elif a < 0:

    print('Negative')

else:

    print('Neutral')
```

Out: **Neutral**

Loops-While

```
>>> a=3
>>> while(a>0):
        print(a)
        a-=1
```

Out:

```
3
2
1
```

Loops-For

```
>>> a=3
>>> for a in range(3):
        print(a+1)
```

Out:

1

2

3

Note: the **range** function returns a sequence

```
>>> list(range(3))
```

Out: **[0, 1, 2]**

Iteration over a List, Tuple, Dictionary

```
>>> for a in range(3):
1.          print(a+1)
```

Out:

```
1
2
3
```

Challenges

1. Convert a dictionary into a list. Each key is repeated according to its value. E.g.:

   ```
   {'A':1, 'B': 2, 'C': 3}

      =>

   ['A', 'B', 'B', 'C', 'C', 'C']
   ```

2. Find count, sum average Of numbers in a list