# Python Programming Language

**Prepared by: Mohamed Ayman**

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

**spring 2020**



sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

# Lecture 10

# Dictionaries

# Course Roadmap


## Part 2: Python Collections and Strings

Lecture 7: Strings

Lecture 8: Lists

Lecture 9: Tuples

**Lecture 10: Dictionaries**

Lecture 11: Sets

Lecture 12: Numbers

# Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to Dictionary

2- Basic Dictionary Operations

3- Dictionary Comprehension

4- Nested Dictionary

5- Built-in Dictionary Functions


Let's
START UP

# Lecture Agenda

Section 1: Introduction to Dictionary

Section 2: Basic Dictionary Operations

Section 3: Dictionary Comprehension

Section 4: Nested Dictionary

Section 5: Built-in Dictionary Functions

# Introduction to Dictionary

- Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair. Dictionaries are optimized to retrieve values when the key is known.
- Creating a dictionary is as simple as placing items inside curly braces { } separated by comma. An item has a key and the corresponding value expressed as a pair, key: value. While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

```
x = {}
x = {1: 'low', 5: 'high'}
x = {'name': 'John', 'grades': [97, 95, 98]}
```

# Python Dictionary

- Dictionary is an unordered collection of key-value pairs, It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. Python dictionaries are kind of hash-table type. They work like associative arrays or hashes found in perl and consist of key-value pairs.
- In Python, a Dictionary can be created by placing sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.
- Dictionary can also be created by the built-in function dict( ). Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this {}

# Python Dictionary


Example:

```
x = {'python':'easy', 'c++':'medium', 'java':'hard'}
print(x)
print(len(x))
print(x['python'])
x['c#'] = 'medium'
x['c++'] = 0
print(x)
print(len(x))
print(x['c#'])
print(x['c++'])
```

Output:

```
{'python':'easy', 'c++':'medium', 'java':'hard'}
3
easy
{'python':'easy', 'c++':0, 'java':'hard', 'c#':'medium'}
4
medium
0
```

# Python Dictionary


Example:

```
x = {'London': 6, 'Paris': 8, 'Barcelone': 11}
print(x)
x['Paris'] = {'Zurich': 2, 'Roma': 5}
x['Barcelone'] = 'windy'
print(x)
print(x['Paris']['Zurich'])
print(x['Barcelone'][:3])
```

Output:

```
{'London': 6, 'Paris': 8, 'Barcelone': 11}
{'London': 6, 'Paris': {'Zurich': 2, 'Roma': 5}, 'Barcelone': 'windy'}
2
win
```

# Lecture Agenda

- ✔ Section 1: Introduction to Dictionary
- **Section 2: Basic Dictionary Operations**
- Section 3: Dictionary Comprehension
- Section 4: Nested Dictionary
- Section 5: Built-in Dictionary Functions

# Python Dictionary

- Dictionary are mutable. We can add new items or change the value of existing items using assignment operator. If the key is already present, value gets updated, else a new key: value pair is added to the dictionary. You can update a dictionary by adding a new entry or a key- value pair, modifying an existing, or deleting an existing entry.
- In Python Dictionary, Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Nested key values can also be added to an existing Dictionary. Note- While adding a value, if the key value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

Example:

```
x = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
print(1 in x)
print(2 not in x)
print(49 not in x)
print(9 in x)
```

Output:

```
True
True
True
True
```

# Updating Dictionary


Example:

```
x = {'name':'Jack', 'age': 26}
print(x)
x['age'] = 27
print(x)
x['address'] = 'Downtown'
print(x)
```

Output:

```
{'name': 'Jack', 'age': 26}
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

# Updating Dictionary

Example:

```
x = {'name':'Jack', 'age': 26}
print(x)
del x['age']
print(x)
del x
print(x)
```

Output:

```
{'name': 'Jack', 'age': 26}

{'name': 'Jack'}

Traceback (most recent call last):
  File "main.py", line 6, in <module>
    print(x)
NameError: name 'x' is not defined
```

# Lecture Agenda

- ✔ Section 1: Introduction to Dictionary
- ✔ Section 2: Basic Dictionary Operations
- **Section 3: Dictionary Comprehension**
- Section 4: Nested Dictionary
- Section 5: Built-in Dictionary Functions

# Dictionary Comprehension

- Dictionary values have no restrictions. They be any arbitrary Python object, either standard objects or user-defined objects. The same not true for the keys. More than one entry per key is not allowed. This means no duplication key is allowed. When duplicate keys are encountered during assignment, the last assignment wins.
- Dictionary comprehension is an elegant and concise way to create new dictionary from an iterable in Python. Dictionary comprehension consists of an expression pair (key: value) followed by for statement inside curly braces { }.

```
x = {'name':'Jack', 'age': 26}
y = {i:x[i] for i in x}

print(x)
print(y)
```

```
{'name': 'Jack', 'age': 26}
{'name': 'Jack', 'age': 26}
```

# Dictionary Comprehension

- Using generator comprehensions to initialize dictionaries is so useful that Python actually reserves a specialized syntax for it, known as the dictionary comprehension. A dictionary comprehension is a syntax for constructing a dictionary, which exactly mirrors the generator comprehension syntax:

```
{<expression> for <var> in <iterable> {if <condition>}}
```

- For example, if we want to create a dictionary of square-numbers, we can simply write:

```
x = {i: i*i for i in range(6)}
print(x)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

- This produces the exact same result as feeding the dictionary function a generator comprehension. However, using a dictionary comprehension is slightly more efficient than is feeding the dictionary function a generator comprehension.

# Dictionary Comprehension


Example:

```
x = {i: i*i for i in range(6)}
print(x)
```

Output:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

```
x = {}
for i in range(6):
    x[i] = i*i
print(x)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

```
x = {i: i*i for i in range(11) if i%2 == 1}
print(x)
```

```
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

# Lecture Agenda

- ✔ Section 1: Introduction to Dictionary
- ✔ Section 2: Basic Dictionary Operations
- ✔ Section 3: Dictionary Comprehension
- **Section 4: Nested Dictionary**
- Section 5: Built-in Dictionary Functions

# Nested Dictionary

- A Dictionary in Python works similar to the Dictionary in the real world. Keys of a Dictionary must be unique and of immutable data type such as Strings, Integers and tuples, but the key-values can be repeated and be of any type.
- Nesting Dictionary means putting a dictionary inside another dictionary. Nesting is of great use as the kind of information we can model in programs is expanded greatly.
- Addition of elements to a nested Dictionary can be done in multiple ways. One way to add a dictionary in the Nested dictionary is to add values one be one, Nested_dict[dict][key] = 'value'. Another way is to add the whole dictionary in one go, Nested_dict[dict] = { 'key': 'value'}.
- Deletion of dictionaries from a nested dictionary can be done either by using del keyword or by using pop() function.

# Nested Dictionary

Example:

```
x = {
    0: { "flavor": "Vanilla", "price": 0.50, "pints": 20},
    1: { "flavor": "Chocolate", "price": 0.50, "pints": 31},
    2: { "flavor": "Cookies and Cream", "price": 0.75, "pints": 14}
}

print(x)
```

Output:

```
{0: {'flavor': 'Vanilla', 'price': 0.5, 'pints': 20},
 1: {'flavor': 'Chocolate', 'price': 0.5, 'pints': 31},
 2: {'flavor': 'Cookies and Cream', 'price': 0.75, 'pints': 14}}
```

# Lecture Agenda

- ✔ Section 1: Introduction to Dictionary
- ✔ Section 2: Basic Dictionary Operations
- ✔ Section 3: Dictionary Comprehension
- ✔ Section 4: Nested Dictionary

**Section 5: Built-in Dictionary Functions**

# Built-in Dictionary Functions


1- len() Method
2- items() Method
3- keys() Method
4- values() Method
5- update() Method
6- pop() Method
7- get() Method
8- setdefault() Method
9- copy() Method
10- clear() Method

# len() Method


Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
print(len(x))
```

Output: 3

```
x = {'name': 'Jack', 2: 'male', 'grades': [98, 99]}
print(len(x))
```

Output: 3

```
x = {(5, 1): 'Jack', 2: (3.2, 7.1), 'grades': [98, 99]}
print(len(x))
```

Output: 3

# items() Method

Example:
```
x = {'name': 'Jack', 2: 'male', 'grades': [98, 99]}
y = x.items()
print(list(y))
```

Output:
```
[('name', 'Jack'), (2, 'male'), ('grades', [98, 99])]
```

Example:
```
x = {(5, 1): 'Jack', 2: (3.2, 7.1), 'grades': [98, 99]}
y = x.items()
print(list(y))
```

Output:
```
[((5, 1), 'Jack'), (2, (3.2, 7.1)), ('grades', [98, 99])]
```

# keys() Methods


Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
y = list(x.keys())
print(y)
```

Output:

```
['name', 'age', 'address']
```

```
x = {'name': 'Jack', 2: 'male', 'grades': [98, 99]}
y = list(x.keys())
print(y)
```

```
['name', 2, 'grades']
```

# values() Methods


Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
y = list(x.values())
print(y)
```

Output:

```
['Jack', 26, 'Downtown']
```

```
x = {'name': 'Jack', 2: 'male', 'grades': [98, 99]}
y = list(x.values())
print(y)
```

```
['Jack', 'male', [98, 99]]
```

# update() Method

Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
y = {'name': 'Jack', 'age': 28, 'gender': 'male'}
x.update(y)
print(x)
```

Output:

```
{'name': 'Jack', 'age': 28, 'address': 'Downtown', 'gender': 'male'}
```

Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
y = {'name': 'Jack', 'age': 28, 'gender': 'male'}
y.update(x)
print(y)
```

Output:

```
{'name': 'Jack', 'age': 26, 'gender': 'male', 'address': 'Downtown'}
```

# pop() Method

Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
x.pop('age')
print(x)
```

Output:

```
{'name': 'Jack', 'address': 'Downtown'}
```

Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
x.pop('Downtown')
print(x)
```

Output:

```
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    x.pop('Downtown')
KeyError: 'Downtown'
```

# get() Method


Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
print(x.get('name'))
print(x.get('age'))
print(x.get('salary'))
print(x.get('salary', 0.0))
```

Output:

```
Jack
26
None
0.0
```

# setdefault() Methods


Example:

```
x = {'name':'Jack', 'age': 26}
x.setdefault('gender')
print(x)
x.setdefault('gender', 'male')
print(x)
x.setdefault('age')
print(x)
x.setdefault('age', 22)
print(x)
x.setdefault('address', 'Downtown')
print(x)
```

Output:

```
{'name': 'Jack', 'age': 26, 'gender': None}
{'name': 'Jack', 'age': 26, 'gender': None}
{'name': 'Jack', 'age': 26, 'gender': None}
{'name': 'Jack', 'age': 26, 'gender': None}
{'name': 'Jack', 'age': 26, 'gender': None, 'address': 'Downtown'}
```

# copy() Method


Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
y = x.copy()
print(x)
print(y)
```

Output:

```
{'name': 'Jack', 'age': 26, 'address': 'Downtown'}
{'name': 'Jack', 'age': 26, 'address': 'Downtown'}
```

Example:

```
x['age'] = 22
y['gender'] = 'male'
print(x)
print(y)
```

Output:

```
{'name': 'Jack', 'age': 22, 'address': 'Downtown'}
{'name': 'Jack', 'age': 26, 'address': 'Downtown', 'gender': 'male'}
```

# clear() Method

Example:

```
x = {'name': 'Jack', 'age': 26, 'address': 'Downtown'}
print(x)
```

Output:

```
{'name': 'Jack', 'age': 26, 'address': 'Downtown'}
```

Example:

```
x.clear()
print(x)
```

Output:

```
{}
```

# Practice

# Practice Problems


1- Implement a function which calculates the length of a dictionary
2- Implement a function which converts a dictionary  to list of tuples (key, value)
3- Implement a function which gets the keys of a dictionary
4- Implement a function which gets the values of a dictionary
5- Implement a function which updates a dictionary with a given dictionary
6- Implement a function which deletes an item in the dictionary with it's key
7- Implement a function which sets a default value in a given key if it's not exists
8- Implement a function which counts the frequency of each word in a sting
9- Implement a function which gets the most 3 words repeated in a string
10- Implement a function which prints the dictionary items in non-descending order of keys

# Built-in Dictionary Functions


1- len() Method

2- items() Method

3- keys() Method

4- values() Method

5- update() Method

6- pop() Method

7- get() Method

8- setdefault() Method

9- copy() Method

10- clear() Method

# Lecture Agenda

- ✔ Section 1: Introduction to Dictionary
- ✔ Section 2: Basic Dictionary Operations
- ✔ Section 3: Dictionary Comprehension
- ✔ Section 4: Nested Dictionary
- ✔ Section 5: Built-in Dictionary Functions


DO
MORE.