Lab 2

Instructor: Dr. Diaa Salama Abdel Moneim  
Eng. Noha El Masry  
Eng. Nour Elhuda Ashraf

**Icon

Description automatically generatedLogo

Description automatically generated**

# Python types and operations(continued):

Primitive (Built-in) data types

* **Lists**

The Python **list** object is the most general sequence provided by the language. Lists are **positionally ordered collections of arbitrarily typed objects** (can collect homogenous or heterogenous data), and they have **no fixed size**.

They are also **mutable** (unlike strings), lists can be modified in place by assignment to offsets as well as a variety of list method calls. Lists allow data duplication (unlike sets).

Because they are sequences, lists support all the sequence operations we discussed for strings; the only difference is that **the results are usually lists instead of strings**. Python does not have arrays; the corresponding data structure is list.  
  
Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++, and Java. Python code is typically one-third to one-fifth the size of equivalent C++ or Java code.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Changing the contents of a list is only done by assigning an item or a set of items with specific values, while concatenation and multiplication don’t change the original list.

Text, letter

Description automatically generated

Text, letter

Description automatically generatedAlthough lists have **no fixed size**, Python still doesn’t allow us to reference items that are not present. **Indexing off the end** of a list is always a mistake, but so is **assigning off the end**.

Changing the contents of a list is only done by assigning an item or a set of items with specific values, while concatenation and multiplication don’t change the original list.

Text, letter

Description automatically generated

Although lists have **no fixed size**, Python still doesn’t allow us to reference items that are not present. **Indexing off the end** of a list is always a mistake, but so is **assigning off the end**.

Text, letter

Description automatically generated

One nice feature of Python’s core data types is that they support **arbitrary nesting**, we can nest them in any combination, and as deeply as we like.

Text, letter

Description automatically generated

Python provides powerful methods for lists, and because lists are mutable, most list methods also change the list object in place, instead of creating a new one.

|  |  |  |
| --- | --- | --- |
| **Function** | **Input** | **Returned Value** |
| append() | object | list w/ appended obj at last |
| insert() | index, object | list w/ object at index |
| extend() | list | concatenated list |
| remove() | object value | list w/o object |
| pop() | object index | list w/o object |
| index() | element value | index of the element |
| del [n]/list\_name | index or list name | list w/o nth element/deleted list |
| clear() | - | remove all elements |
| len() | - | # of elements in the list |
| sort() | - | ascendingly sorted list |
| sort(reverse = True) | - | descending sorted list |
| sort(key = function) | customized function | customized sorted list |
| reverse() | - | reverse list order |
| copy() | - | copy a list to another one |
| + | more than one list | concatenated list |

The Map Function built-in function applies a function to items in a sequence and collects all the results in a new list.

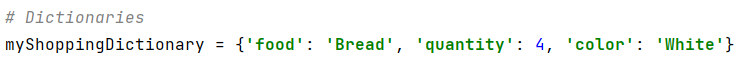


* **Dictionaries**

Python dictionaries are something completely different, they are not sequences at all, but are instead known as **mappings**.

Mappings are also collections of other objects, but they **store objects by key instead of by relative position**. In fact, mappings don’t maintain any reliable left-to-right order; they simply map keys to associated values.

**Dictionaries, the only mapping type in Python’s core objects set**, are also **mutable**: like lists, they may be changed in place and can grow and shrink on demand.



We can index this dictionary by **key** to fetch and **change the keys’ associated values.**

Graphical user interface, text, application

Description automatically generated

It is perhaps more common to see dictionaries built up in different ways **(it’s rare to know all your program’s data before your program runs)**. For example, start with an empty dictionary and fill it out one key at a time.

Text

Description automatically generatedUnlike **out-of-bounds assignments in lists, which are forbidden**, **assignments to new dictionary keys create those keys**.

We can nest dictionaries into each other or any other types of objects, we can access the components of this structure much as any list-based matrix, but this time most indexes are dictionary keys, not list offset

Graphical user interface, text, application

Description automatically generated

As mappings, **dictionaries support accessing items by key only**, we can **assign to a new key to expand a dictionary**, **fetching a nonexistent key is still a mistake**.

Text, letter

Description automatically generated

Text

Description automatically generated

In Python 3.7 and later, dictionaries are changeable. The matter is different for Python 3.6 or earlier

|  |  |  |
| --- | --- | --- |
| **Function** | **Input** | **Output** |
| clear() | - | Removes all the elements from the dictionary |
| copy() | - | a copied dictionary |
| fromkeys() | keys + values | a constructed dictionary |
| get() | key | the value of the specified key |
| items() | - | a list of all pairs |
| keys() | - | a list of all keys |
| values() | - | a list of all values |
| pop() | key + value (optional) | a dictionary w/o the item |
| popitem() | - | a dictionary w/o the last inserted item |
| setdefault() | key + value (optional) | the value of the specified key. If the key does not exist, insert the key with the specified value |
| update() | key + value | a dictionary with inserted item |

* **Files**

File objects are Python code’s main interface to external files on your computer. They can be used to read and write text memos, audio clips, Excel documents, saved email messages, and whatever else you happen to have stored on your machine.

|  |  |  |
| --- | --- | --- |
| **Function** | **Input** | **Description** |
| open() | file pathname/ file name + file operation |  |
| read(n) | file + number of character to be read | reads n characters from a file |
| readline() | - | read a single line |
| write() | text to be written/appended | write text in file |
| close() | - | close the opened file |

File Operations:

1. "r" - Read - Default value. Opens a file for reading, error if the file does not exist.
2. "a" - Append - Opens a file for appending, creates the file if it does not exist.
3. "w" - Write - Opens a file for writing, creates the file if it does not exist.
4. "x" - Create - Creates the specified file, returns an error if the file exists.
5. "t" - Text - Default value. Text mode.
6. "b" - Binary - Binary mode (e.g. images).

* **Time and Date**

Graphical user interface, text, application

Description automatically generatedBefore using Time/Date objects you need to import datetime package.

Text

Description automatically generated with medium confidence

You can reformat dates using strftime() function

# Functions and Generators:

In simple terms, a function is a device that groups a set of statements so they can be run more than once in a program (a packaged procedure invoked by name).

Functions also can compute a result value and let us specify parameters that serve as function inputs and may differ each time the code is run.

Functions are also the most basic program structure Python provides for maximizing code reuse, and lead us to the larger notions of program design.

As we’ll see, functions let us split complex systems into manageable parts. By implementing each part as a function, we make it both reusable and easier to code.

* **Functions**

To define a function in Python, we use **def** keyword. Function prototype in Python differs in missing the type of the returned object.

def function\_name(parameter1, parameter2):

#function Body

#return statements

Text

Description automatically generated In Python function can return multiple objects.



Exercises:

For all following exercise, write the main function to test your implementation.

1. Write a Python function to check whether a number is perfect or not.

Input/Output Scenario:

Please enter a number: 28

28 is a perfect number

Please enter a number: 9

9 is not a perfect number

1. Write a Python function that takes a list and returns a new list with unique elements of the first list.

Input/Output Scenario:

Please enter a sample list: [1,2,3,3,3,4,4,5]

The unique list: [1, 2, 3, 4, 5]

1. Write a Python program to find the key of the maximum value and minimum value in a dictionary.
2. Write a Python program to sort the dictionary once by key.