**Dictionaries**

Dictionary is a data type provided by python, which is another collection of objects.

A dictionary is similar to a list, in many ways:

* It’s mutable (his elements can be changed)
* It’s dynamic (it can grow as needed)

Dictionaries are an unordered collection of items that store values on a **key-value pair** basis.

With a dictionary, you create a structured piece of data and retrieve values using a key.

You can think of them as a spreadsheet table: you have columns with names (keys) and cells with data (values).

**Difference between dictionaries and lists**

How elements are accessed:  
In **lists**, elements are **accessed by their position** in the list (via indexing)

**Example:**

my\_list = ['Rick', 'Sanchez']

print("My last name is:", my\_list[1])

To access the word "Sanchez", we need to put his index in the square brackets [].

In dictionaries, it’s a bit different; elements have no index; they are accessed via keys. A dictionary is an associative array, it consists of a collection of key-value pairs. Each key=value pair maps the key to its associated value (for example, the key first\_name is associated to Rick, and the key last\_name is associated with Sanchez).

**1. Define a dictionary**

While lists are defined by square brackets [], dictionaries are defined by enclosing a list of key-value pairs in curly braces {}.  
A colon : is used to separate each key from its associated value:

my\_dict = {

<KEY1>: <VALUE1>,

<KEY2>: <VALUE2>,

<KEY3>: <VALUE3>

}

rick\_dict = {

'first\_name':'Rick',

'last\_name':'Sanchez'

}

Keys need to be declared as strings with quotation marks. Values can be of any type.

my\_dog = {

'name': 'Rufus',

'age': 4,

'good\_dog': True

}

**2. Accessing data**

Accessing values in a dictionary is just a matter of referencing the key of that value between brackets. This is similar to how we access values from a list, but instead of using the index of the value, we use the name of its **key**.

my\_dog = {

'name': 'Rufus',

'age': 4

}

print(my\_dog['name'])

# 'Rufus'

**Example:**

rick\_dict = {

'first\_name':'Rick',

'last\_name':'Sanchez'

}

print("The last name of rick is:", rick\_dict['last\_name'])

If you refer to a key that is not in the dictionary, you will get an error:

print("The last name of rick is:", rick\_dict['age'])

Traceback (most recent call last):

File "<pyshell#19>", line 1, in <module>

rick\_dict['age']

KeyError: 'age'

You can use [tuple unpacking](https://www.pythontutorial.net/python-basics/python-unpacking-tuple/#:~:text=Unpacking%20tuples%20means%20assigning%20individual,assign%20it%20to%20a%20variable.) to iterate through the keys and values of the dictionary you are working with.

To achieve this, you just need to unpack the elements of every item into two different variables representing the key and the value:

a\_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}

print(a\_dict.items())

# output :

dict\_items([('color', 'blue'), ('fruit', 'apple'), ('pet', 'dog')])

# The items() method returns a view object that displays

# a list of dictionary's (key, value) tuple pairs.

for key, value in a\_dict.items():

print(key, '->', value)

# output

color -> blue

fruit -> apple

pet -> dog

**3. Data types**

You can store any type of data inside a dictionary. That could be strings and integers but also other dictionaries or lists.

my\_dog = {

'name': 'Rufus',

'age': 4,

'best\_friend': {

'name': 'Felix',

'age': 4.5

},

'favorite\_foods': ['steaks', 'sausages', 'shawarma']

}

**Dictionaries and lists**

Dictionaries can be especially useful when stored in a list to represent a collection of multiple items representing the same thing, such as a list of products.

shirts = [

{

'name': 'Awesome T-shirt 3000',

'size': 'S',

'price': 20

},

{

'name': 'Awesome T-shirt 3000',

'size': 'M',

'price': 25

},

{

'name': 'Awesome T-shirt 3000',

'size': 'L',

'price': 30

},

]

Print(shirts[1][‘price’]

# output 25

**Exercise**

Access the value of the key history

sample\_dict = {

"class":{

"student":{

"name":"Mike",

"marks":{

"physics":70,

"history":80

}

}

}

}

print(sample\_dict["class"]["student"]["marks"]["history"])

**4. Modify an entry in a dictionary**

Modifying an element in a dictionary works the same as a list, first, you need to select it (with his key) and then assign it to a new value with the equal sign =.

rick\_dict['last\_name'] = 'SANCHEZ'

**5. Adding an entry to an existing dictionary**

To add an entry to an existing dictionary, just assign a new key to its value, like you would have modified it if it was an existing key.

rick\_dict['hair\_color'] = 'white'

**6. Delete an entry in a dictionary**

To delete an entry, use the del statement, specifying the key to delete:

del rick\_dict['hair\_color']

**Keys restrictions**

Almost any type of value can be used as a dictionary key in Python. But there are a couple of restrictions.

* First, keys are unique; they can appear in a dictionary only once; if you assign a value to an existing dictionary key, it does not add the key a second time but replaces the current value.
* Secondly, the key must be an immutable type; it can be an integer, a float, a string, a boolean, and even a tuple because all those types are immutables. List can’t be a dictionary key.

By contrast, there are no restrictions on dictionary values.

**The in keyword**

In a list, the in keyword returns True or False according to whether the specified operand occurs in the list.

A dictionary returns True if the operand occurs **as a key** in the dictionary.

**Iterating**

Iterating over a list of dictionaries is simple; your iterator contains the dictionary you want to look at.

for shirt in shirts:

print(shirt['size'])

# S

# M

# L

**Built-in methods**

**keys() –**

The my\_dict.keys() method returns a dict\_keys of all the keys in my\_dict

**values() –**

The my\_dict.values() method returns a dict\_values of all the values in my\_dict

**items() –**

The my\_dict.items() method returns a dict\_items of tuples containing the key-value pairs in a dictionary.

rick\_dict = {

'first\_name':'Rick',

'last\_name':'Sanchez'

}

print(rick\_dict.items())

**update(<key>)**

If <obj> and d are dictionaries, d.update(<obj>) merges the entries from <obj> into d.

For each key in <obj>:

* If the key is not present in d, the key-value pair from <obj> is added to d.
* If the key is already present in d, the corresponding value in d for that key is updated to the value from <obj>.

[More dictionary methods](https://docs.python.org/3/library/stdtypes.html#mapping-types-dict)

**Exercise**

Delete set of keys from Python Dictionary

sample\_dict = {

"name": "Kelly",

"age":25,

"salary": 8000,

"city": "New york"

}

keys\_to\_remove = ["name", "salary"]

**Expected output:**

{'city': 'New york', 'age': 25}

**I. For Loops**

**A. For Loops and dictionaries**

We can iterate over both the keys and the values of a dictionary.

my\_books = {

"title": "Harry Potter",

"author": "JK Rowling",

}

for x, y in my\_books.items():

print("the" + x + "is" + y)

>> the title is Harry Potter

the author is JK Rowling

**B. Loops Operator**

**range(start, stop[, step]) : iterator in loops.**

print(list(range(1, 10, 2)))

>> [1, 3, 5, 7, 9]

**enumerate(iterable) : enumerate each item in the *iterable***

for item in enumerate('abcd'):

print(item)

>>

(0, 'a') # Syntax : (index , value)

(1, 'b')

(2, 'c')

(3, 'd')

for (index\_count, letter) in enumerate('abcd'):

print(f'At index {index\_count} the letter is {letter}')

>>

At index 0 the letter is a

At index 1 the letter is b

At index 2 the letter is c

At index 3 the letter is d

**zip(iterable,..) : concat [iterables, …] in a tuple.**

list1 = [1,2,3]

list2 = ['a','b','c']

list3 = [1.1, 2.2, 3.3, 4.4, 5.5]

for item in zip(list1, list2, list3): # only go as far it is possible

print(item)

>>

(1, 'a', 1.1)

(2, 'b', 2.2)

(3, 'c', 3.3)

**C. For Else**

The else part is optional. When included, it is always executed once after the for loop is over unless a break statement is encountered.

for i in range(1, 3):

print(i)

else:

print('The for loop is over')

>>

1

2

The for loop is over

**D. While Else**

while some\_condition:

# do something

else:

# do another thing

x = 0

while x < 2:

print(f'x is {x}')

x += 1

else:

print('x is bigger than 2')

>>

x is 0

x is 1

x is bigger than 2

**E. Break, Continue, Pass**

**break –** break the loop (stop the execution of a looping statement)

if you *break* out of a for or while loop, any corresponding loop else block is **not** executed.

for letter in 'Leonardo':

if letter == 'a':

break

print(letter, end='') # end='' renders each letter next to the other

>> Leon

while True:

s = input('Enter something : ')

if s == 'quit':

break

print('Length of the string is', len(s))

print('Done')

**continue –** return to the top of the loop (*continue* to the next iteration of the loop)

for letter in 'Leonardo':

if letter == 'o':

continue

print(letter, end='') # don’t execute for 'o' letter

>> Lenard

while True:

s = input('Enter something : ')

if s == 'quit':

break

if len(s) < 3:

print('Too small')

continue

print('Input is of sufficient length')

**pass –** do nothing.

for item in [1,2,3]:

pass # to avoid the error

print('Finish the script')

>> Finish the script

**II. List Comprehension: quickly way to creating a list**

Many objects in Python are “iterable”, meaning we can iterate over every element in the object.

List Comprehensions are a unique way of quickly creating a list with Python.

**A. The basic way of appending an element into a list**

my\_number = '1234'

my\_list = []

for num in my\_number:

my\_list.append(num)

print(my\_list)

>> ['1', '2', '3', '4']

**B. The list comprehension way**

my\_number = '1234'

my\_list = [num for num in my\_number]

print(my\_list)

>> ['1', '2', '3', '4']

**Examples with the range method**

my\_list = [x for x in range(0,6)]

print(my\_list)

>> [0, 1, 2, 3, 4, 5]

my\_list = [x\*\*2 for x in range(0,6)] # square

print(my\_list)

>> [0, 1, 4, 9, 16, 25]

my\_list = [x for x in range(0,11) if x%2 == 0] # only even

print(my\_list)

>> [0, 2, 4, 6, 8, 10]

Look up time complexity in python.

**C. The basic way of appending an element into a list with Nested Loop**

my\_list = []

for i in [2, 3, 4]:

for j in [100, 200, 300]:

my\_list.append(i\*j)

print(my\_list)

>> [200, 400, 600, 300, 600, 900, 400, 800, 1200]

**D. The list comprehension way**

my\_list = [(i\*j) for i in [2, 3, 4] for j in [100, 200, 300]]

print(my\_list)

>> [200, 400, 600, 300, 600, 900, 400, 800, 1200]

**E. Dictionary comprehension**

dictionary = {key: value for var in iterable}

family\_age = {'Lea': 12, 'Mark': 25, 'George': 50}

new\_year = 1

new\_family\_age = {name: age+new\_year for (name, age) in family\_age.items()}

print(new\_family\_age)

>> {'Lea': 13, 'Mark': 26, 'George': 51}