

CODE CAMP

Day 004: Dictionaries

Dashboard

CPU Usage: 0% used – 0.00s of 100s. Resets in 23 hours, 15 minutes [More](#)

File storage: 0% full – 124.0 MB of 512.0 MB quota

Recent Consoles

+ 5 -

 [Bash console 15154103](#)

 [Bash console 15153805](#)

New console:

You can have up to 2 consoles. To get more, [upgrade your account!](#)

Recent Files

You have no recently edited files

[+ Open another file](#)



.001 Log In to Python Anywhere

Visit [PythonAnywhere.com](https://pythonanywhere.com) and log in to your account. You can click on one of the bash consoles you logged in to previously. They are listed under “Recent Consoles.”

If your command line is cluttered from your previous work you can type the **clear** command to clear the console visual history.



.001 Log In to Python Anywhere

If you've done everything correctly, your screen should appear similar to the image above. At your command prompt, type the command **nano** to open your text editor.

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
print(grades)
```

.002 Creating a Dictionary

A dictionary is a data structure that is optimized for storing key-value pair data. Think of key-value pairs as two pieces of linked data where one describes the other. For example, players and positions on a baseball team or students and test grades could be stored as key-value pairs.

Where you have a number of key-value pairs to store a dictionary is an appropriate structure.

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
print(grades)
```

.002 Creating a Dictionary

You can use the **print** command to output the dictionary structure all at once. However, ordinarily, you'd deal with one key-value pair at a time. The output from **print** would not make a lot of sense to end users.



```
19:22 ~ $ nano dict.py
19:24 ~ $ python dict.py
{'Neil': 100, 'Jenny': 61, 'Murray': 71, 'Mark': 84, 'Claire': 82, 'Donna': 90, 'Connor': 88}
19:24 ~ $
```

.002 Creating a Dictionary

You will notice that when you print out the dictionary it is surrounded by curly brackets and each key-value pair is printed separately.

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
  
#print(grades)  
print(grades["Mark"])  
print(grades["Neil"])  
print(grades["Jenny"])
```

.003 Accessing Dictionary Members

Instead of using numerical indexes, like lists and tuples, the **key** values for each pair are used to access the data. Because our keys are strings in this instance, double quotes are used around each string key.

Enter the code on the left and see if you get the results you expect when executing the script. Be careful of the syntax in and around the dictionary structure.

Note: The hash sign (#) is used to create a remark in the code. The Python processor ignores any line beginning with the symbol when running code.



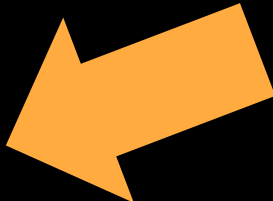
```
19:37 ~ $ python dict.py
84
100
61
19:37 ~ $
```

.003 Accessing Dictionary Members

Run the script you just created in your command line. You will note that the members of the dictionary are access by their key:

"Mark"	:	84
"Neil"	:	100
"Jenny"	:	61


```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
  
#print(grades)  
print(grades["Mark"])  
print(grades["Neil"])  
print(grades["Jenny"])  
print(grades["Bryan"])
```




.003 Accessing Dictionary Members

Note that attempting to access a non-existent key will cause an error.

```
19:39 ~ $ python dict.py  
84  
100  
61  
Traceback (most recent call last):  
  File "dict.py", line 16, in <module>  
    print(grades["Bryan"])  
KeyError: 'Bryan'  
19:39 ~ $ nano dict.py  
19:40 ~ $
```

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
grades["Mark"] = 91  
  
print(grades["Mark"])  
print(grades["Neil"])  
print(grades["Jenny"])
```



.003 Accessing Dictionary Members

When this code is executed, the value for the key **Mark** is replaced with the value 91.

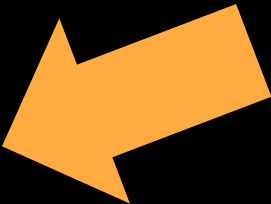
This is reflected in the output when the program is executed.



Bash console 15203305

```
19:43 ~ $ nano dict.py  
19:43 ~ $ python dict.py  
91  
100  
61  
19:43 ~ $
```

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
grades.update({"Sammie" : 94})  
grades.update({"Petula" : 61})  
  
print(grades["Mark"])  
print(grades["Petula"])
```



.004 Manipulating Dictionaries

There are a number of commands that can be used to manipulate the values in dictionaries.

To add a member of the dictionary you need to call the **update** command and state the key-value pair as a tuple, which means surrounding it in curly brackets.

Remember that keys can only be used once in a single dictionary. Key in the example on the left and ensure it works correctly for you.



Bash console 15203305

```
20:02 ~ $ python dict.py  
84  
61  
20:02 ~ $
```

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
grades.update({"Sammie" : 94})  
grades.update({"Petula" : 61})  
  
print(grades["Mark"])  
print(grades["Petula"])
```



Bash console 15203305

```
20:02 ~ $ python dict.py  
84  
61  
20:02 ~ $
```

.004 Manipulating Dictionaries

You can delete individual members from a dictionary, delete all entries from a dictionary and delete the dictionary itself.

del grades["Mark"]	Deletes Mark
grades.clear()	Deletes all
del grades	Deletes Dictionary

Using the dictionary you've already keyed in, try using these three delete commands and make sure they work properly.

What do you think the difference is between deleting all dictionary members with **clear** and deleting the dictionary itself?


.005 Looping Through a Dictionary

You can use a **for** loop through the dictionary object and access all of the key-value pairs. The generic form of the loop is:

for key,value in dictionary.items():

Each time through the loop the key and the value would be assigned. Let's apply this generic form to our current code base.

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
  
for student, grade in grades.items():  
    print(student + " : " + str(grade))
```



.005 Looping Through a Dictionary


We used **student** and **grade** in our loop, however, the names used are arbitrary and should just make sense in the context of your program. Note that when we print out the items accessed, we need to cast the grade from a number to a string so we can use it in the **print** command.

We can also access just the keys or values in the for loop.

 Bash console 15203305

```
20:29 ~ $ python dict.py  
Neil : 100  
Jenny : 61  
Murray : 71  
Mark : 84  
Claire : 82  
Donna : 90  
Connor : 88  
20:29 ~ $
```

```
grades = {  
    "Mark" : 84,  
    "Donna" : 90,  
    "Claire" : 82,  
    "Murray" : 71,  
    "Connor" : 88,  
    "Neil" : 100,  
    "Jenny" : 61  
}  
  
for student, grade in grades.items():  
    print(student + " : " + str(grade))  
  
total = 0  
items = 0  
  
for grade in grades.values():  
    total = total + grade  
    items = items + 1  
  
average = total / items  
print("The average grade is " + str(average))
```



.005 Looping Through a Dictionary

Key in the code in the left and ensure that the class average is correctly calculated.


Note that the **values** command is used to only return the values from the key-value pairs. The total of the grades and the number of items are added up in each loop iteration. The average is then calculated and output.

```
grades = {
    "Mark" : 84,
    "Donna" : 90,
    "Claire" : 82,
    "Murray" : 71,
    "Connor" : 88,
    "Neil" : 100,
    "Jenny" : 61
}

for student, grade in grades.items():
    print(student + " : " + str(grade))

total = 0
items = 0
for grade in grades.values():
    total = total + grade
    items = items + 1
average = total / items
print("The average grade is " + str(average))

print("Class List")
for name in grades:
    print name
```



.005 Looping Through a Dictionary

If we use the **for** loop to loop through the dictionary and don't specify **items** or **values** in the loop, the key will be returned with each iteration. Note the last three lines of code generate a class list out of the keys in the dictionary.



```
20:50 ~ $ python dict.py
Neil : 100
Jenny : 61
Murray : 71
Mark : 84
Claire : 82
Donna : 90
Connor : 88
The average grade is 82
Class List
Neil
Jenny
Murray
Mark
Claire
Donna
Connor
20:50 ~ $ █
```

.005 Looping Through a Dictionary

Here you can see the output of the three **for** loops in the code.



```
21:12 ~ $ python ratings.py
Rate Journey. (1-10)
:10
Rate REO Speedwagon. (1-10)
:7
Rate Styx. (1-10)
:8
Rate Mr. Mister. (1-10)
:9
Rate The Cure. (1-10)
:10
Rate The Doobie Brothers. (1-10)
:7
Rate Neil Diamond. (1-10)
:7
Rate The Beatles. (1-10)
:10
Rate Live. (1-10)
:4
Rate Tina Turner. (1-10)
:10
Rate The Guess Who. (1-10)
:7

Your Ratings:
The Doobie Brothers: 7
Styx: 8
Mr. Mister: 9
Neil Diamond: 7
The Cure: 10
The Beatles: 10
Live: 4
Journey: 10
Tina Turner: 10
REO Speedwagon: 7
The Guess Who: 7

Your average rating is a: 8
21:13 ~ $
```

.006 Activity 001

Carefully examine the output on the left. The ratings were entered in to the program from the command line. The ratings were saved as part of a dictionary and then echoed back to the user with an average calculated.

```
bands = ("Journey", "REO Speedwagon", "Styx",
"Mr. Mister", "The Cure", "The Doobie Brothers",
"Neil Diamond", "The Beatles", "Live", "Tina
Turner", "The Guess Who")
```

```
bandRatings = {}
```

```
for      in      :
    print("Rate " +      + ". (1-10)")
        = input(":")
        bandRatings._____({band : _____})
totalRatings = 0
numRatings = 0
```

```
print("\nYour Ratings:")
for      ,      in bandRatings.      :
    print(band + ": " +      (rating))
    totalRatings =      + rating
    numRatings =      + 1
average = totalRatings/numRatings
```

```
print("\nYour average rating is a: " +
str(_____))
```

.006 Activity 001

Significant parts of the code to make the program demonstrated on the previous page have been deleted. (Please use whatever bands or musical acts you enjoy instead of Mark's!)

Correctly write the missing code so that you have a working program in your console.

Note that `\n` is a character entity the moves to the next line within a string. Also note that the line `bandRatings = {}` creates an empty dictionary.

```
student = ""
gradebook = {}

print("Enter XXX to stop entering students.")

while student != "XXX":
    student = raw input("Name: ")
    if (student != "XXX"):
        grade = int(raw input("Grade: "))
        gradebook.update({student:grade})
    print("\n")
```

.006 Activity 002

Enter the code and complete the program on the left so that after entering each student name and grade, a table of all students and grades is output, along with the average grade earned, the highest grade (and who earned it) and the lowest grade earned (and who earned it).