

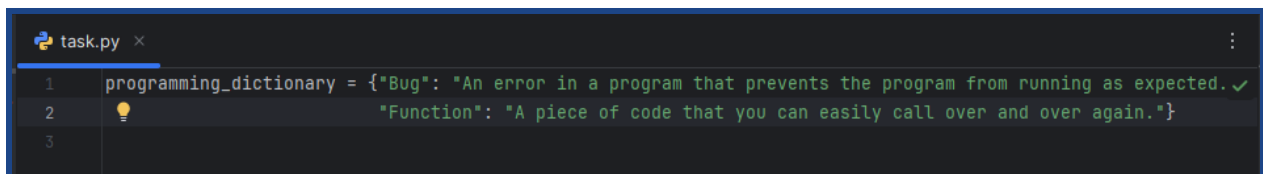
# DAY 9: Dictionaries, Nesting and the Secret Auction

24/01/2025: <https://appbrewery.github.io/python-day9-demo/>

## The Python Dictionary: Deep Dive

Dictionary on python is very similar to a dictionary in real life, lets see this on code:

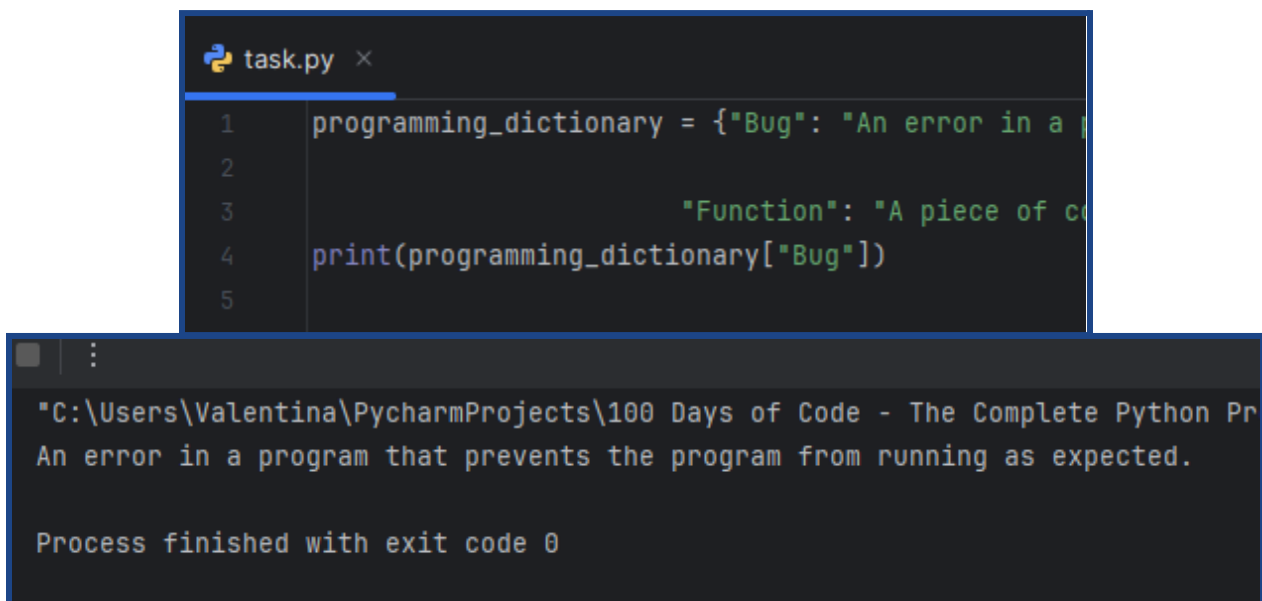
1. They open curly braces at the top and then every subsequent entry is indented by one indent.
2. And then at the very end of that entry there's a comma.
3. And then we hit enter so that the next item goes on to the next line.
4. And finally, the last curly brace should go at the very beginning in line with the start of the dictionary.



```
task.py x
1 programming_dictionary = {"Bug": "An error in a program that prevents the program from running as expected. ✓
2     "Function": "A piece of code that you can easily call over and over again."}
3
```

5. Another nice thing is that you can separate your items with a comma and if you want to add more items you just click enter and add more.

Now, the next thing I want to do is what if I wanted to retrieve an item from the dictionary? For lists we know that we will use index, for dictionaries is kinda similar:



```
task.py x
1 programming_dictionary = {"Bug": "An error in a
2
3     "Function": "A piece of code
4 print(programming_dictionary["Bug"])
5
```

```
"C:\Users\Valentina\PycharmProjects\100 Days of Code - The Complete Python Pr
An error in a program that prevents the program from running as expected.

Process finished with exit code 0
```

And the output will be the definition of the key that we called.

## Common errors:

- Bad spelling of the key
- Putting a key that doesn't exist
- Another pitfall is not using the correct data, So for example, if we defined this dictionary without putting a string around each of these keys, then it's going to error out and it won't even let us run.
- Write the key exactly like it is written between the strings.

Also we can add new items by adding them in a new string, like this:

```
task.py x
1 programming_dictionary = {"Bug": "An error in a program that prevents the program from running as expected"}
2
3     "Function": "A piece of code that you can easily call over and over again."}
4 print(programming_dictionary["Bug"])
5
6 programming_dictionary["Loop"] = "The action of doing something over and over again"
7 print(programming_dictionary)
8
```

This can be really helpful when you have empty dictionaries and you want to add the keys later on. Just as you saw previously, you can create an empty list by simply having a set of square brackets with nothing inside. You can also create an empty dictionary by simply creating a set of curly braces with nothing inside, and then at a later stage you can add to your dictionary by using this method that you saw here.

We can wipe the dictionary like this:

```
8
9
10 empty_dictionary = {} #We set the condition
11
12
13
14
15 #Wipe an empty dictionary
16 #There's 3 keys on the dictionary
17 programming_dictionary={} #We called the dictionary
18 print(programming_dictionary) #We print it
19
20
```

And the output will just be the curly braces. That can be really useful if you wanted to clear out a user's progress, or for example, if a game restarts, then all the scores and stats will probably have to be wiped empty. We can also use a lot of this method on edit the keys:

```
19
20 #Edit an item in a dictionary
21 programming_dictionary["Bug"] = " A thing on your computer" #Were going to redefine their value
22 print(programming_dictionary)
23
```

The computer will look through the dictionary and re assign the new definition and finally if you printed it your output will look with the new definition.

```
23
24 #Loop through a dictionary
25 for thing in programming_dictionary:
26     print(thing)
27
```

"C:\Users\Valentina\PycharmProjects\100 Days of Code - T  
An error in a program that prevents the program from run  
{'Bug': 'An error in a program that prevents the program  
Bug  
Function  
Loop  
  
Process finished with exit code 0  
|

The code would just give you the keys, and it's not what I expected personally , but if you changed it like this:

```
24 #Loop through a dictionary
25 for key in programming_dictionary:
26     print(key)
27     print(programming_dictionary[key])
28
```

So now when I hit Run, you can see it's giving me first the key from this line and then secondly the value based on this line. I'm using that retrieval code in order to get the value.

## Coding Exercise 9: Grading Program

Write a program that converts their scores to grades.

```
task.py ×
29
30 student_scores = {
31     'Harry': 88,
32     'Ron': 78,
33     'Hermione': 95,
34     'Draco': 75,
35     'Neville': 60
36 }
37 student_grades={}
38
39 for student, score in student_scores.items():
40     if score in range(91, 101):
41         grade="Outstanding"
42     elif score in range(81, 91):
43         grade="Exceeds Expectations"
44     elif score in range(71, 81):
45         grade="Acceptable"
46     else:
47         grade="Fail"
48
49
50     student_grades[student] = grade
51
52 print(student_grades)
53
```

# Nesting Lists and Dictionaries

Nesting is when you put one thing (like a list or dictionary) inside another. Think of it as folders inside folders. This helps you organize complex data in Python.

## A Simple Dictionary

A dictionary has key-value pairs:

```
1 capitals = {  
2     "France" : "Paris",  
3     "Germany" : "Berlin",  
4 }
```

Here, the key "France" has the value "Paris".

## Nesting a List in a Dictionary

If a country has multiple cities, we can use a **list** as the value:

```
#Nested list dictionary  
travel_log = {  
    "France": ["Paris", "Lille", "Dijon"],  
    "Germany": ["Stuttgart", "Berlin"],  
}
```

The key "France" has a list of cities as its value. How can we print Lille?

```
12  
13 print(travel_log["France"][1]) # Output: Lille  
14
```

## Nesting a Dictionary in a Dictionary

We can store more details about each country by nesting another **dictionary**:

```
18  travel_log = {
19      "France": {
20          "num_times_visited": 5,
21          "cities_visited": ["Paris", "Lille", "Dijon"]
22      },
23      "Germany": {
24          "num_times_visited": 3,
25          "cities_visited": ["Berlin", "Hamburg", "Munich"]
26      }
27  }
```

"France" is a key, and its value is another dictionary.

Inside "France", there are two keys:

- "num\_times\_visited": Stores how many times you visited France, make sure you put them between strings.
- "cities\_visited": A list of cities.

## Why Use Nesting?

Nesting helps store and organize **complex data**:

- Example: A travel log can have countries, the cities you visited, and how many times you've been there.
- Without nesting, organizing all this would be much harder!

# The Secret Auction Program

## Instructions and Flow Chart

main.py ×

A newer version of this course is available. We strongly recommend updating to the new version.

```
4
5 # Step 2: Collect bids using a loop
6 while True:
7     # Ask for user input
8
9     user_name = input("What is your name? ")
10    user_bid = int(input("What is your bid? "))
11    # Add to the dictionary
12    bid_dictionary[user_name] = user_bid
13
14    # Ask if there are other bidders
15    other_bid = input("Are there other bidders? Type 'yes' or 'no': ").lower()
16    if other_bid == "no":
17        break
18    print("\n" * 20)
19
20 # Step 3: Determine the highest bid
21 highest_bid = 0
22 winner = ""
23
24 for bidder, bid in bid_dictionary.items():
25     if bid > highest_bid:
26         highest_bid = bid
27         winner = bidder
28
29 # Step 4: Print the winner
30 print(f"The highest bid is ${highest_bid}, and the winner is {winner}.")
31
32
33
34 # TODO-2: Save data into dictionary {name: price}
35
36 # TODO-3: Whether if new bids need to be added
37
38 # TODO-4: Compare bids in dictionary
39
```