CREATING DICTIONARIES

Introduction to Python Dictionaries

A *dictionary* is an unordered set of `key: value` pairs.

It provides us with a way to map pieces of data to each other so that we can quickly find values that are associated with one another.

Suppose we want to store the prices of various items sold at a cafe:

- Avocado Toast is 6 dollars
- Carrot Juice is 5 dollars
- Blueberry Muffin is 2 dollars

In Python, we can create a dictionary called `menu` to store this data:

```
menu = {"avocado toast": 6, "carrot juice": 5, "blueberry muffin": 2}
```

Notice that:

1. A dictionary begins and ends with curly braces `{` and `}`.
2. Each item consists of a key (`"avocado toast"`) and a value (`6`).
3. Each `key: value` pair is separated by a comma.

It's considered good practice to insert a space () after each comma, but our code will still run without the space.

Instructions

1.

Suppose we have a dictionary of temperature sensors in the house and what temperatures they read. We've just added a sensor to the `"pantry"`, and it reads 22 degrees.

Add this pair to the dictionary on line 1.

Hint

It should look something like:

```
sensors = {"living room": 21, "kitchen": 23, "bedroom": 20, "pantry":
22}
```

2.

Remove the `#` in front of the definition of the dictionary `num_cameras`, which represents the number of cameras in each area around the house.

If you run this code, you'll get an error:

```
SyntaxError: invalid syntax
```

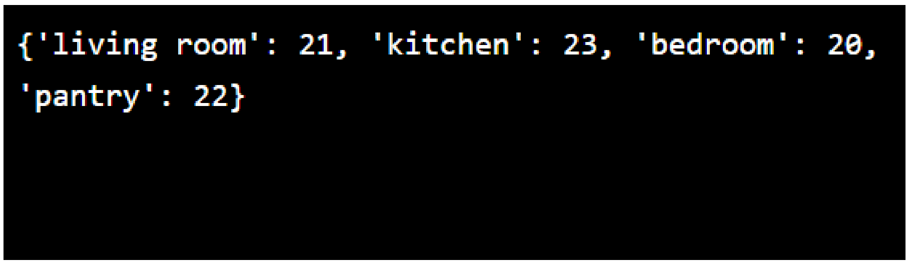
Try to find and fix the syntax error to make this code run.

Hint

Add commas (`,`) to `num_cameras` so that it runs without errors.

script.py

```
sensors =  {"living room": 21, "kitchen": 23, "bedroom": 20, "pantry": 22}
num_cameras = {"backyard": 6,  "garage": 2, "driveway": 1}

print(sensors)
```



```
{'living room': 21, 'kitchen': 23, 'bedroom': 20,
'pantry': 22}
```

Make a Dictionary

In the previous exercise, we saw a dictionary that maps strings to numbers (i.e., `"avocado toast": 6`). However, the keys can be numbers as well.

For example, if we were mapping restaurant bill subtotals to the bill total after tip, a dictionary could look like:

```
subtotal_to_total = {20: 24, 10: 12, 5: 6, 15: 18}
```

Values can be of any type. We can use a string, a number, a list, or even another dictionary as the value associated with a key!

For example:

```
students_in_classes = {"software design": ["Aaron", "Delila",
"Samson"], "cartography": ["Christopher", "Juan", "Marco"],
"philosophy": ["Frederica", "Manuel"]}
```

The list `["Aaron", "Delila", "Samson"]`, which is the value for the key `"software design"`, represents the students in that class.

We can also mix and match key and value types. For example:

```
person = {"name": "Shuri", "age": 18, "family": ["T'Chaka",
"Ramonda"]}
```

Instructions

1.

Create a dictionary called `translations` that maps the following words in English to their definitions in Sindarin (the language of the elves):

English	Sindarin
mountain	orod
bread	bass
friend	mellon
horse	roch

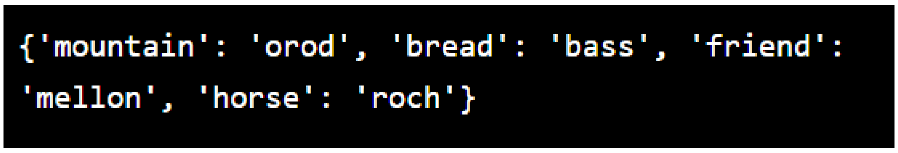
Hint

It should look something like:

```
translations = {"mountain": "orod", "bread": "bass", ...}
```

script.py

```
translations = {"mountain":"orod", "bread":"bass", "friend":"mellon", "horse":"ro
ch"}

print(translations)
```



```
{'mountain': 'orod', 'bread': 'bass', 'friend':
'mellon', 'horse': 'roch'}
```

Invalid Keys

We can have a list or a dictionary as a *value* of an item in a dictionary, but we cannot use these data types as keys of the dictionary. If we try to, we will get a `TypeError`.

For example:

```
powers = {[1, 2, 4, 8, 16]: 2, [1, 3, 9, 27, 81]: 3}
```

This code will yield:

```
TypeError: unhashable type: 'list'
```

The word “unhashable” in this context means that this ‘list’ is an object that can be changed.

Dictionaries in Python rely on each key having a *hash value*, a specific identifier for the key. If the key can change, that hash value would not be reliable. So the keys must alwany be unchangeable, hashable data types, like numbers or strings.

Instructions

1.
Run the code inside **script.py**. You should get an error:

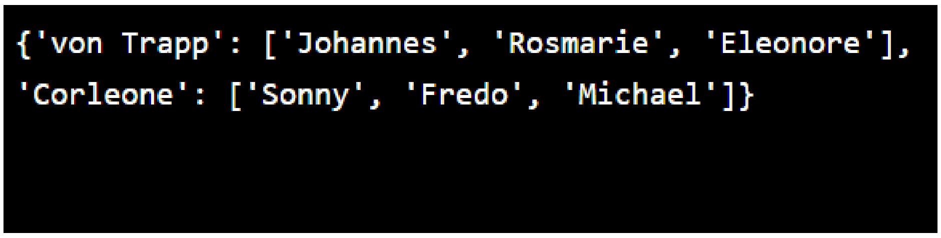
```
TypeError: unhashable type
```
Make the code run without errors by flipping the items in the dictionary so that the strings are the keys and the lists are the values

Hint
It should look something like:

```
children = {"von Trapp": ["Johannes", "Rosmarie", "Eleonore"], ...}
```

script.py

```
children = {"von Trapp": ["Johannes", "Rosmarie", "Eleonore"], "Corleone": ["Sonny", "Fredo", "Michael"]}


print(children)
```



```
{'von Trapp': ['Johannes', 'Rosmarie', 'Eleonore'],
'Corleone': ['Sonny', 'Fredo', 'Michael']}
```

Empty Dictionary
A dictionary doesn’t have to contain anything. Sometimes we need to create an empty dictionary when we plan to fill it later based on some other input.

We can create an empty dictionary like this:

```
empty_dict = {}
```
We will explore ways to fill a dictionary in the next exercise.

Instructions

1.
Create an empty dictionary called `my_empty_dictionary`.

Hint
It should look something like:

```
name = {}
```

script.py

```
my_empty_dictionary = {}
print(my_empty_dictionary)
```

```
{}
```

Add A Key

To add a single `key: value` pair to a dictionary, we can use the syntax:

```
dictionary[key] = value
```

For example, if we had our `menu` dictionary from the first exercise:

```
menu = {"oatmeal": 3, "avocado toast": 6, "carrot juice": 5,
"blueberry muffin": 2}
```

And we wanted to add a new item, `"cheesecake"` for `8` dollars, we could use:

```
menu["cheesecake"] = 8
```

Now, `menu` looks like:

```
{"oatmeal": 3, "avocado toast": 6, "carrot juice": 5, "blueberry
muffin": 2, "cheesecake": 8}
```

Instructions

1.
Create an empty dictionary called `animals_in_zoo`.

2.
Walking around the zoo, you see 8 zebras. Add `"zebras"` to `animals_in_zoo` as a key with a value of `8`.

3.
The primate house was bananas! Add `"monkeys"` to `animals_in_zoo` as a key with a value of `12`.

4.

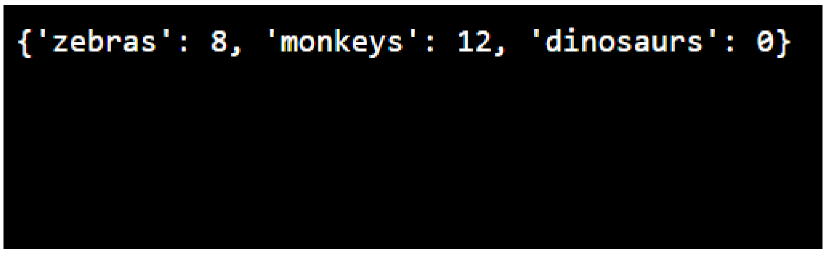
As you leave the zoo, you are saddened that you did not see any dinosaurs. Add `"dinosaurs"` to `animals_in_zoo` as a key with a value of `0`.

5.

Print `animals_in_zoo`.

script.py

```
animals_in_zoo = {}
animals_in_zoo["zebras"] = 8
animals_in_zoo["monkeys"] = 12
animals_in_zoo["dinosaurs"] = 0
print(animals_in_zoo)
```



```
{'zebras': 8, 'monkeys': 12, 'dinosaurs': 0}
```

Add Multiple Keys

If we wanted to add multiple key : value pairs to a dictionary at once, we can use the `.update()` method.

Looking at our `sensors` object from a previous exercise:

```
sensors = {"living room": 21, "kitchen": 23, "bedroom": 20}
```

If we wanted to add 3 new rooms, we could use:

```
sensors.update({"pantry": 22, "guest room": 25, "patio": 34})
```

This would add all three items to the `sensors` dictionary.

Now, `sensors` looks like:

```
{"living room": 21, "kitchen": 23, "bedroom": 20, "pantry": 22,
"guest room": 25, "patio": 34}
```

Instructions

1.

In one line of code, add two new users to the `user_ids` dictionary:

- `theLooper`, with an id of 138475
- `stringQueen`, with an id of 85739

Hint
It should look something like:

```
user_ids.update({"username1": 100, "username2": 101})
```

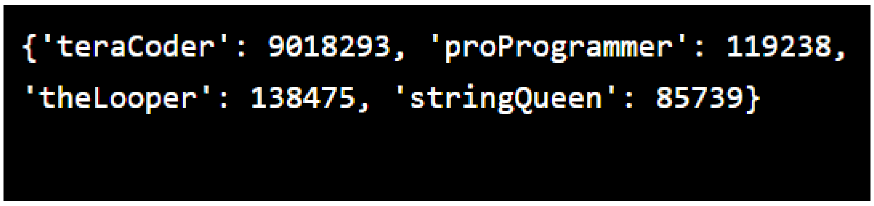
2.
Print `user_ids`.

Hint
It should look something like:

```
print(name)
```

script.py

```
user_ids = {"teraCoder": 9018293, "proProgrammer": 119238}

user_ids.update({"theLooper": 138475, "stringQueen": 85739})

print(user_ids)
```



```
{'teraCoder': 9018293, 'proProgrammer': 119238,
'theLooper': 138475, 'stringQueen': 85739}
```

Overwrite Values

We know that we can add a key by using the following syntax:

```
menu["banana"] = 3
```

This will create a key `"banana"` and set its value to `3`. But what if we used a key that already has an entry in the `menu` dictionary?

In that case, our value assignment would overwrite the existing value attached to that key. We can overwrite the value of `"oatmeal"` like this:

```
menu = {"oatmeal": 3, "avocado toast": 6, "carrot juice": 5,
"blueberry muffin": 2}
menu["oatmeal"] = 5
print(menu)
```

This would yield:

```
{"oatmeal": 5, "avocado toast": 6, "carrot juice": 5, "blueberry
muffin": 2}
```
Notice the value of `"oatmeal"` has now changed to `5`.

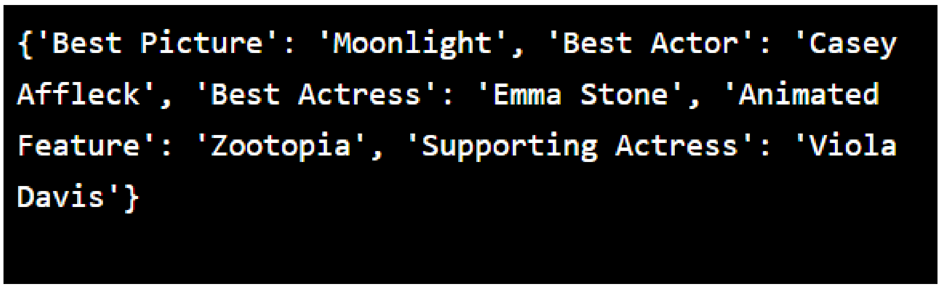
Instructions

1.
Add the key `"Supporting Actress"` and set the value to `"Viola Davis"`.

2.
Without changing the definition of the dictionary `oscar_winners`, change the value associated with the key `"Best Picture"` to `"Moonlight"`.

script.py

```
oscar_winners = {"Best Picture": "La La Land", "Best Actor": "Casey Affleck", "Be
st Actress": "Emma Stone", "Animated Feature": "Zootopia"}

oscar_winners["Supporting Actress"] = "Viola Davis"

oscar_winners["Best Picture"] = "Moonlight"
```



```
{'Best Picture': 'Moonlight', 'Best Actor': 'Casey
Affleck', 'Best Actress': 'Emma Stone', 'Animated
Feature': 'Zootopia', 'Supporting Actress': 'Viola
Davis'}
```

Dict Comprehensions

Let's say we have two lists that we want to combine into a dictionary, like a list of students and a list of their heights, in inches:

```
names = ['Jenny', 'Alexus', 'Sam', 'Grace']
heights = [61, 70, 67, 64]
```
Python allows you to create a dictionary using a dict comprehension, with this syntax:

```
students = {key:value for key, value in zip(names, heights)}
#students is now {'Jenny': 61, 'Alexus': 70, 'Sam': 67, 'Grace': 64}
```
Remember that `zip()` combines two lists into an iterator of tuples with the list elements paired together. This dict comprehension:

1. Takes a pair from the iterator of tuples

2. Names the elements in the pair `key` (the one originally from the `names` list) and `value` (the one originally from the `heights` list)
3. Creates a `key` : `value` item in the `students` dictionary
4. Repeats steps 1-3 for the entire iterator of pairs

Instructions

1.

You have two lists, representing some drinks sold at a coffee shop and the milligrams of caffeine in each. First, create a variable called `zipped_drinks` that is an iterator of pairs between the `drinks` list and the `caffeine` list.

Hint

For example, to create an iterator of pairs between `names` and `heights`:

```
names = ['Jenny', 'Alexus', 'Sam', 'Grace']
heights = [61, 70, 67, 64]

zipped_students = zip(names, heights)
```

2.

Create a dictionary called `drinks_to_caffeine` by using a dict comprehension that goes through the `zipped_drinks` iterator and turns each tuple pair into a key:value item.

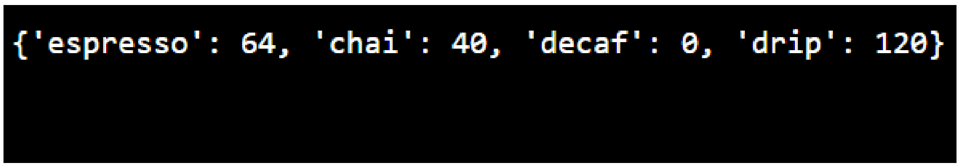
Hint

Use the following syntax:

```
dict_variable = {key: value for key, value in zip_iterator}
```

script.py

```
drinks = ["espresso", "chai", "decaf", "drip"]
caffeine = [64, 40, 0, 120]

zipped_drinks = zip(drinks, caffeine)

drinks_to_caffeine = {key:value for key, value in zipped_drinks}

print(drinks_to_caffeine)
```



```
{'espresso': 64, 'chai': 40, 'decaf': 0, 'drip': 120}
```

Review
So far we have learned:

- How to create a dictionary
- How to add elements to a dictionary
- How to update elements in a dictionary
- How to use a dict comprehension to create a dictionary from two lists

Let's practice these skills!

Instructions

1.
We are building a music streaming service. We have provided two lists, representing songs in a user's library and the amount of times each song has been played.

Using a dict comprehension, create a dictionary called `plays` that goes through `zip(songs, playcounts)` and creates a `song:playcount` pair for each `song` in `songs` and each `playcount` in `playcounts`.

Hint
To create a dictionary from a dict comprehension use this syntax:

```
new_dict = {key:value for key, value in zip(key_list, value_list)}
```

2.
Print `plays`.

3.
After printing `plays`, add a new entry to it. The entry should be for the song `"Purple Haze"` and the playcount is `1`.

4.
This user has caught Aretha Franklin fever and listened to "Respect" 5 more times. Update the value for `"Respect"` to be 94 in the `plays` dictionary.

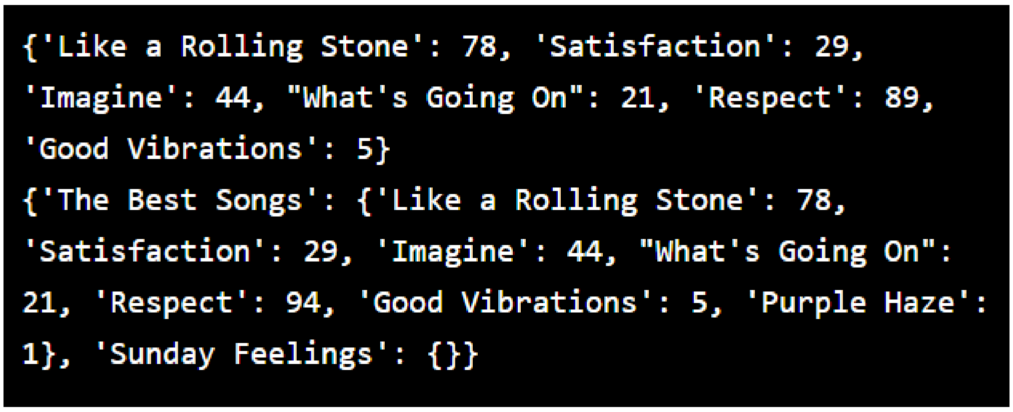
5.
Create a dictionary called `library` that has two key: value pairs:

- key `"The Best Songs"` with a value of `plays`, the dictionary you created
- key `"Sunday Feelings"` with a value of an empty dictionary

6.
Print `library`.

script.py

```
songs = ["Like a Rolling Stone", "Satisfaction", "Imagine", "What's Going On", "R
espect", "Good Vibrations"]
playcounts = [78, 29, 44, 21, 89, 5]

plays = {key:value for key, value in zip(songs, playcounts)}

print(plays)

plays["Purple Haze"] = 1

plays["Respect"] = 94

library = {"The Best Songs": plays, "Sunday Feelings": {}}

print(library)
```



```
{'Like a Rolling Stone': 78, 'Satisfaction': 29,
'Imagine': 44, "What's Going On": 21, 'Respect': 89,
'Good Vibrations': 5}
{'The Best Songs': {'Like a Rolling Stone': 78,
'Satisfaction': 29, 'Imagine': 44, "What's Going On":
21, 'Respect': 94, 'Good Vibrations': 5, 'Purple Haze':
1}, 'Sunday Feelings': {}}
```
