Dictionaries (cont'd)

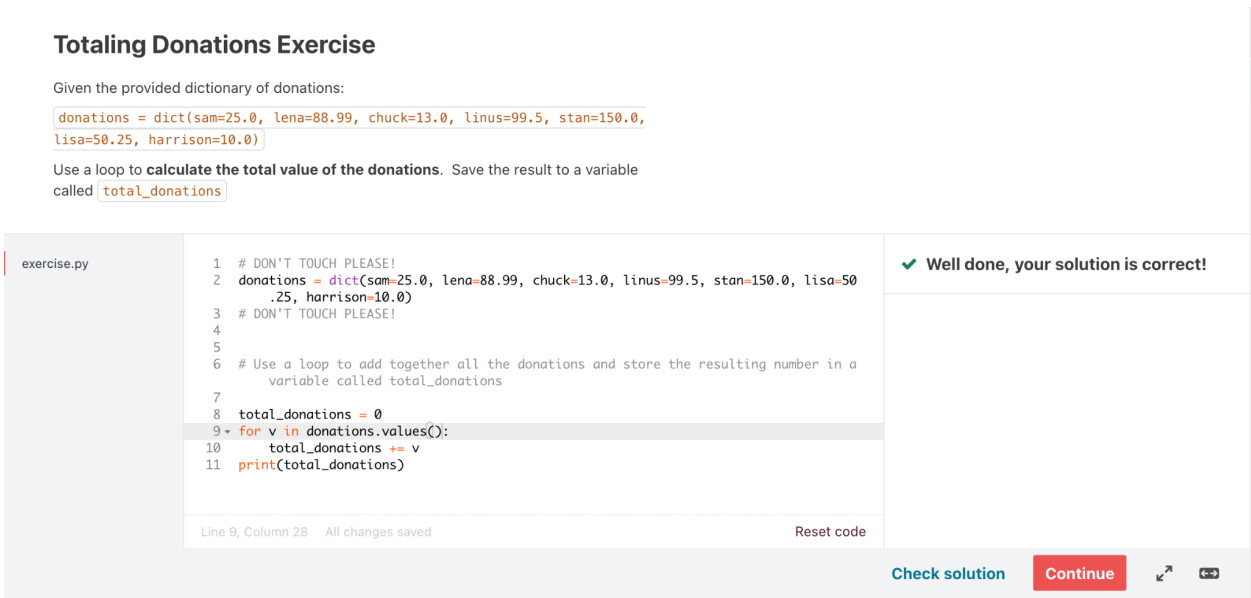
<u>Accessing all values in a dictionary:</u>
We can easily access all the available keys or values by using the **keys()** or **values()** method. For example,

```
information = {
        'name': 'tina',
        'owns_cat': True,
        'age': 25,
        'fav_number': 13,
        'home_state': 'New York'
}
```

information**.keys()** # outputs ['name', 'owns_cat', 'age', 'fav_number', 'home_state']
information**.values()** # outputs ['tina', True, 25, 13, 'New York']

Another way to access them is by using the **items()** method:

information**.items()** # output [('name', 'tina'), ('owns_cat', True), ('age', 25), ('fav_number', 13), ('home_state', 'New York')]



We can also check to see whether the key value pair is in the dictionary by using the keyword **in**.

<u>Dictionary methods:</u>
Working with dictionaries is pretty common. There are a few things we can do:

- Clear — clears all the keys and values from the dictionary
- Copy — copies the dictionary

- Fromkeys — creates key value pairs from comma separated values. It is a way to create default values.

  i.e.

  new_user = {}.fromkeys(['name', 'score', 'email', 'profile bio'], 'unknown')
- Get — retrieves a key in an object and return None instead of a KeyError if the key does not exist

Dictionary Access

For this exercise, I've defined some code for you already.

- The `food` variable will store a randomly chosen food string like "gummy bear" or "morning bun". Some of these items are in the `bakery_stock` dictionary, and some are not.
- Your task is to simply print out a string depending on if `food` is a value in `bakery_stock` .If `food` is contained in `bakery_stock` print out a string that states how many items are left: `"3 left"` if food is "*toffee cookie*" or `"1 left"` if food is "*morning bun*", etc.
- If food is not contained in `bakery_stock` (like "gummy bear"), print out "We don't make that"

exercise.py

```
    "almond croissant" : 12,
    "toffee cookie": 3,
    "morning bun": 1,
    "chocolate chunk cookie": 9,
    "tea cake": 25
}

# YOUR CODE GOES BELOW:

if food in bakery_stock:
    print("{} left".format(bakery_stock[food]))
else:
    print("We don't make that")
```

Line 19, Column 32 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

Fromkeys Exercise

Imagine we're creating a video game and want to model the initial starting-state of our game. I've provided you with a list of strings called game_properties.

- Use `dict.fromkeys` to generate a new dictionary using the provided `game_properties` list. Initialize all values to `0` .
- Save the result in a variable called `initial_game_state`

HINT: the end result should look like this:

{'current_score': 0, 'high_score': 0, 'number_of_lives': 0, 'items_in_inventory': 0, 'power_ups': 0, 'ammo': 0, 'enemies_on_screen': 0, 'enemy_kills': 0, 'enemy_kill_streaks': 0, 'minutes_played': 0, 'notifications': 0, 'achievements': 0}

exercise.py

```
#DO NOT TOUCH game_properties!
game_properties = ["current_score", "high_score", "number_of_lives",
    "items_in_inventory", "power_ups", "ammo", "enemies_on_screen", "enemy_kills",
    "enemy_kill_streaks", "minutes_played", "notifications", "achievements"]

# Use the game_properties list and dict.fromkeys() to generate a dictionary with all
    values set to 0. Save the result to a variable called initial_game_state
initial_game_state = dict.fromkeys(game_properties, 0)
```

Line 5, Column 55 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

- Pop — takes a single argument and removes that key-value pair from the dictionary. Returns the value corresponding to the key that was removed.

```
i.e.  information = {
        'name': 'tina',
        'owns_cat': True,
        'age': 25,
        'fav_number': 13,
        'home_state': 'New York'
}

information.pop('age')        # returns 25
information.pop()             # returns TypeError: expected at least 1
                              argument
information.pop('fav_color') # returns KeyError
```

- PopItem — removes a random key in a dictionary. If we were to pass a key within the arguments of popitem(), it will return TypeError: popitem() takes no arguments.
- Update — update keys and values in a dictionary with another set of key value pairs. If key already exists in dictionary, update() will overwrite the value within the second dictionary

```
i.e.
information = {
        'name': 'tina',
        'owns_cat': True,
        'age': 25,
        'fav_number': 13,
        'home_state': 'New York'
}

person = {'city': 'Antigua'}  # creating second dictionary

person.update(information)  # takes key value pairs from information and
                              copies them into person dictionary
                              information dictionary remains unchanged
```

Dictionary Methods Exercise

I've provided you with a start dictionary called `inventory.`

`inventory = {'croissant': 19, 'bagel': 4, 'muffin': 8, 'cake': 1} #DON'T CHANGE THIS LINE!`

Follow the instructions found in the comments:

1. Make a copy of `inventory` and save it to a variable called `stock_list` using a dictionary method we've covered
2. Add the value 18 to `stock_list` under the key *"cookie"*
3. Remove 'cake' from `stock_list` using a dictionary method we've learned

exercise.py

```
inventory = {'croissant': 19, 'bagel': 4, 'muffin': 8, 'cake': 1} #DON'T CHANGE THIS LINE!

# Make a copy of inventory and save it to a variable called stock_list USE A DICTIONARY METHOD
stock_list = inventory.copy()

# add the value 18 to stock_list under the key "cookie"
stock_list['cookie'] = 18

# remove 'cake' from stock_list USE A DICTIONARY METHOD
stock_list.pop('cake')
```

Line 7, Column 26 All changes saved Reset code

✔ **Well done, your solution is correct!**

Check solution Continue

more examples

```
{num: num**2 for num in [1,2,3,4,5]}
```

```
str1 = "ABC"
str2 = "123"
combo = {str1[i]: str2[i] for i in range(0,len(str1))}
print(combo) # # {'A': '1', 'B': '2', 'C': '3'}
```

State Abbreviations Exercise

Given two lists `["CA", "NJ", "RI"]` and `["California", "New Jersey", "Rhode Island"]` create a dictionary that looks like this `{'CA': 'California', 'NJ': 'New Jersey', 'RI': 'Rhode Island'}`. Save it to a variable called **answer.**

I expect you to do this with a dictionary comprehension, but you can also use a built-in function called `zip`. We cover it later in the course.

exercise.py

```
list1 = ["CA", "NJ", "RI"]
list2 = ["California", "New Jersey", "Rhode Island"]

# make sure your solution is assigned to the answer variable so the tests can work!
answer = {}
answer = dict(zip(list1, list2))
```

Line 6, Column 31 All changes saved Reset code

✔ **Well done, your solution is correct!**

Check solution Continue

List to Dictionary Exercise

Given a person variable:

`person = [["name", "Jared"], ["job", "Musician"], ["city", "Bern"]]`

Create a dictionary called `answer`, that makes each first item in each list a key and the second item a corresponding value. That's a terrible explanation. I think it'll be easier if you just look at the end goal:

`{'name': 'Jared', 'job': 'Musician', 'city': 'Bern'}`

There are many potential solutions for this.

exercise.py

```
person = [["name", "Jared"], ["job", "Musician"], ["city", "Bern"]]

# use the person variable in your answer
answer = {k:v for k,v in person}
```

Line 3, Column 41 All changes saved Reset code

✔ **Well done, your solution is correct!**

Check solution Continue

Vowels Dict Exercise

Create a dictionary with the key as a vowel in the alphabet and the value as 0. Your dictionary should look like this `{'a': 0, 'e': 0, 'i': 0, 'o': 0, 'u': 0}`.

Do this programmatically (using a dict comprehension or dict method) rather than hard coding the answer!

exercise.py

```
# make sure your solution is assigned to the answer variable so the tests can work!
answer = dict.fromkeys('aeiou', 0)
```

Line 2, Column 35 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

ASCII Codes Dictionary

This is a bit different. Every character has an ASCII code (basically, a number that represents it). Python has a function called chr() that will return a string if you provide the corresponding integer ASCII code. For example:

`chr(65)` will return 'A'

`chr(66)` will return 'B'

All the way up to:

`chr(90)` will return 'Z'

Your task is to create dictionary that maps ASCII keys to their corresponding letters. Use a dictionary comprehension and `chr()`. Save the result to the answer variable. You only need

exercise.py

```
# make sure your solution is assigned to the answer variable so the tests can work!
answer = {c: chr(c) for c in range(65,91)}
```

Line 2, Column 26 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

Tuples and Sets

Objectives:

- Describe, create, and access tuples and sets
- Use built-in methods to modify sets and access values in tuples
- Iterate over sets using loops and set comprehensions
- Compare and contrast sets & tuples with lists & dictionaries

What is a tuple? It is an ordered collection or grouping of items. The difference with tuples is that it is immutable—aka it cannot be changed.

Why use a tuple? Tuples are faster than lists—they are lighter weight. It makes your code safer. We can use tuples as valid keys in a dictionary

<u>Sets:</u>

- Sets are like formal mathematical sets
- Sets do not have duplicate values
- Elements in sets aren't ordered
- You cannot access items in a set by index
- Sets can be useful if you need to keep track of a collection of elements, but don't care about ordering, keys or values, and duplicates

<u>Set Methods:</u>

- Add — adds an element to a set. If the element is already in the set, the set doesn't change.
- Remove — removes a value from the set - returns a KeyError if the value is not found
- Clear — removes all the contents of the set