Lists (cont'd)

Accessing All Values in a List:

There are a few ways to access all values in a list, such as a for loop! i.e.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for num in numbers:
    print(num * num)

# output should be 1 \n 4 \n 9 \n 16 \n 25 \n 36 \n 49 \n 64 \n 81 \n 100
```

We can also use a while loop, however it can be a little bit more complicated since it requires a little bit more syntax. i.e.

```
numbers = [1, 2, 3, 4]
i = 0

while i < len(numbers):
    print(numbers[i])
    i += 1

# output should be 1 \n 2 \n 3 \n 4
```

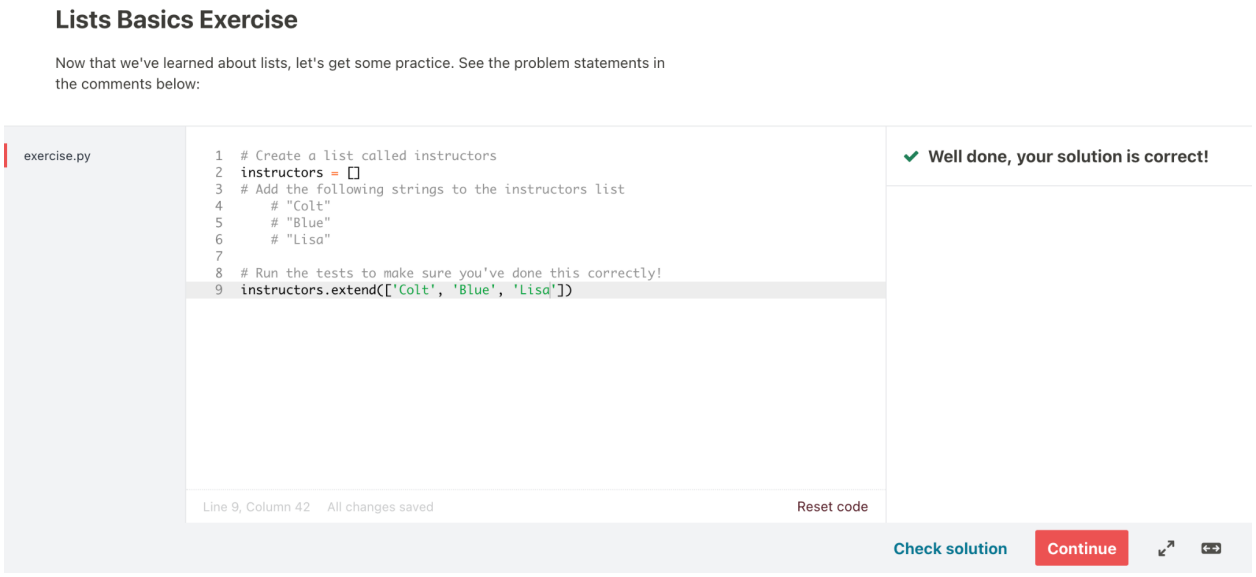
List Methods:

Working with lists is very common—there are quite a few things we can do!

- Append — it will add an item to the end of the list

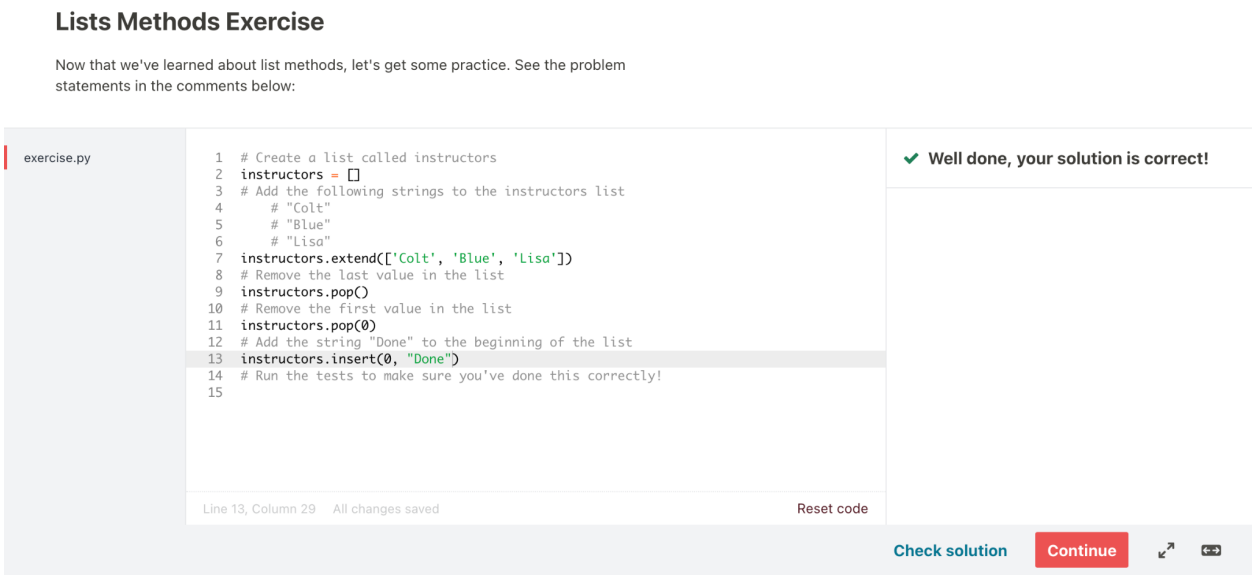
```
i.e. data = [1, 2, 3]
data.append('purple')
print(data)     #outputs [1, 2, 3, 'purple']
```

- Extend — add to the list of all values passed to extend
  i.e. correct_list = [1, 2, 3, 4]
  correct_list.extend([5, 6, 7, 8])
  print(correct_list) #output [1, 2, 3, 4, 5, 6, 7, 8]
- Insert — inserts an item at a given position
  i.e. first_list = [1, 2, 3, 4]
  first_list.insert(2, 'Hi!')
  print(first_list) # outputs [1, 2, 'Hi!', 3, 4]



- Clear — will remove all of the items from the list at once.
  i.e. first_list = [1, 2, 3, 4]
  first_list.clear()
  print(first_list) #output []
- Pop — remove the item at the given position in the list, and return it.
  - If no index is specified, it will remove and return the last item in the list
  i.e. first_list = [1, 2, 3, 4]
  first_list.pop() #4
  first_list.pop(1) #2
- Remove — remove the first item from the list whose value is *x*.
  - Throws ValueError if the item is not found
  i.e. first_list = [1, 2, 3, 4, 4, 4]
  first_list.remove(4)
  print(first_list) # output [1, 2, 3, 4, 4] (only removes first 4)
  first_list.remove(2)
  print(first_list) # output [1, 3, 4, 4]
- Index — helps the user find the index of an item within the list
  i.e. num = [5, 6, 7, 8, 9, 10]
  num.index(6) # 1
  num.index(9) # 4

- Count — returns the number of times *x* appears in the list
  i.e. num = [1, 2, 3, 4, 3, 2, 1, 4, 10, 2]
  num.count(2) # 3
  num.count(21) # 0
  num.count(3) # 2
- Reverse — reverse the elements of the list (in-place)
  i.e. num = [1, 2, 3, 4]
  num.reverse()
  print(num) # [4, 3, 2, 1]
- Sort — sort the items of the list (in-place)
  i.e. num = [6, 4, 1, 2, 5]
  num.sort()
  print(num) # [1, 2, 4, 5, 6]
- Join — joins an entire list of strings into a sentence
  - Technically a **String method** that takes an iterable argument
  - **Concatenates** (combines) a copy of th base string **between** each item of the iterable
  - Returns a new string
  - Can be used to make sentences out of a list of words by joining on a space
    i.e. words = [‘Coding’, ‘Is’, ‘Fun!’]
    ‘ ‘.join(words) # ‘Coding is Fun!’



Slicing:

Make new lists using slices of the old list!

some_list[start:end:step]

First parameter for slice: start

i.e. what index to start slicing from

```
num = [1, 2, 3, 4]
num[1:]       # [2, 3, 4]
```

- If you enter a negative number, it will start the slice that many back from the end

  i.e. (using the same list num)

```
num[-1:] # [4]
num[-3:] # [2, 3, 4]
```

Second parameter for slice: end

i.e. the index to copy up to (exclusive counting)

```
num = [1, 2, 3, 4]
num[:2]       # [1, 2] (does not include second index)
num[1:3]      # [2, 3]
```

- With negative numbers, how many items to exclude from the end (i.e. indexing by counting backwards)

  i.e. num[1:-1] # [2, 3]

Third parameter for slice: step

- "Step" in Python is basically the number to count at a time
- same as step with range!
- for example, a step of 2 counts every other number (1, 3, 5)

  i.e. num = [1, 2, 3, 4, 5, 6]

```
num[1::2]     # [2, 4, 6]
num[::2]      # [1, 3, 5]
```

- With negative numbers, reverse the order

```
i.e. num[1::-1]       # [2, 1]
num[:1:-1]            # [6, 5, 4, 3]
num[2::-1]            # [3, 2, 1]
```

<u>Tricks with Slices:</u>

- Reversing lists/strings

  i.e. string = "This is fun!"

```
string[::-1]
```

- Modifying portions of lists

  i.e. num = [1, 2, 3, 4, 5]

```
num[1:3] = ['a', 'b', 'c']
print(num)    # [1, 'a', 'b', 'c', 4, 5]
```

<u>Swapping Values in Lists:</u>

```
i.e. names = ['James', 'Michelle']
names[0], names[1] = names[1], names[0]
print(names)          # ['Michelle', 'James']
```

When do you need to swap? Shuffling, sorting, algorithms

List comprehension:

The idea of comprehension applies to many other data structures.

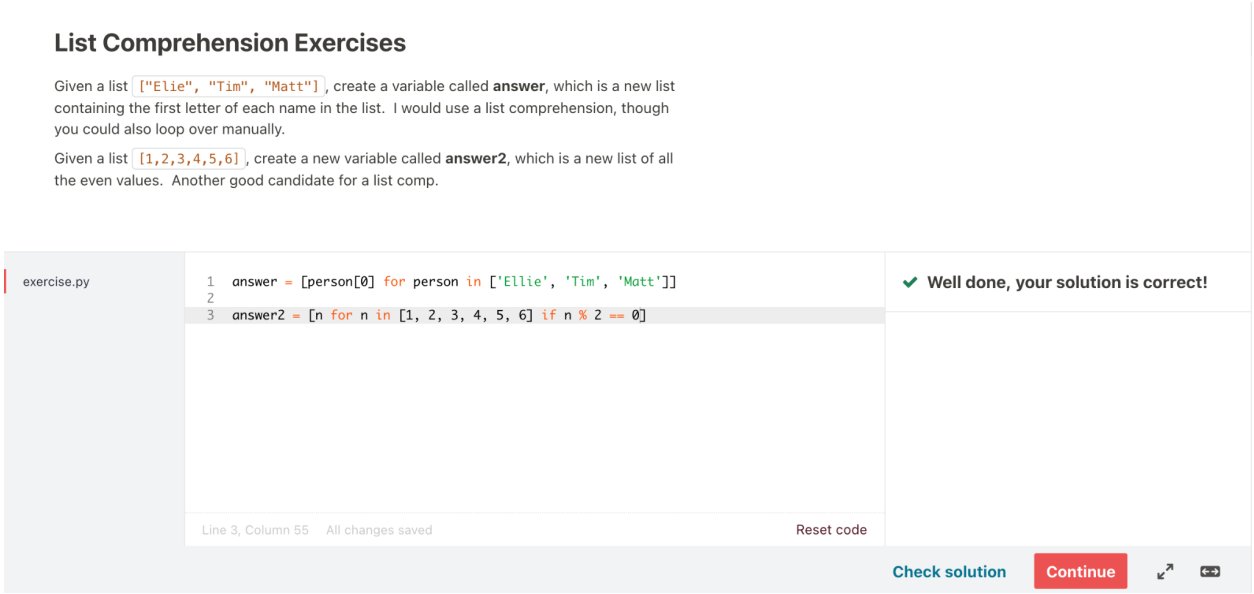
It is a cool shorthanded syntax that allows us to generate new lists, make new lists that are direct copies of other lists, and are, more often than not, tweaked versions where we could take one list and generate a new one that has every number squared or every string reversed.

- The syntax (example) :

```
nums = [1, 2, 3]
x*10 for x in nums    # output [10, 20, 30]
```

- Conditional Logic:

```
numbers = [1, 2, 3, 4, 5, 6]
evens = [num for num in numbers if num % 2 == 0]
odd = [num for num in numbers if num % 2 != 0]
```



More List Comprehension Exercises

1. Given two lists `[1,2,3,4]` and `[3,4,5,6]`, create a variable called **answer,** which is a new list that is the intersection of the two. Your output should be `[3,4]`. Hint: use the `in` operator to test whether an element is in a list. For example: `5 in [1,5,2]` is True. `3 in [1,5,2]` is False.

2. Given a list of words `["Elie", "Tim", "Matt"]` create a variable called **answer2,** which is a new list with each word reversed and in lower case (use a slice to do the reversal!) Your output should be `['eile', 'mit', 'ttam']`

exercise.py

```
1  answer = [n for n in [1, 2, 3, 4] if n in [3, 4, 5, 6]]
2
3  answer2 = [name[::-1].lower() for name in ['Elie', 'Tim', 'Matt']]
```

Line 3, Column 67 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

Another List Comp Exercise

For all the numbers between 1 and 100(including 100), create a variable called **answer**, which contains a list with all the numbers that are divisible by 12. (12, 24, etc)

USE A LIST COMPREHENSION.

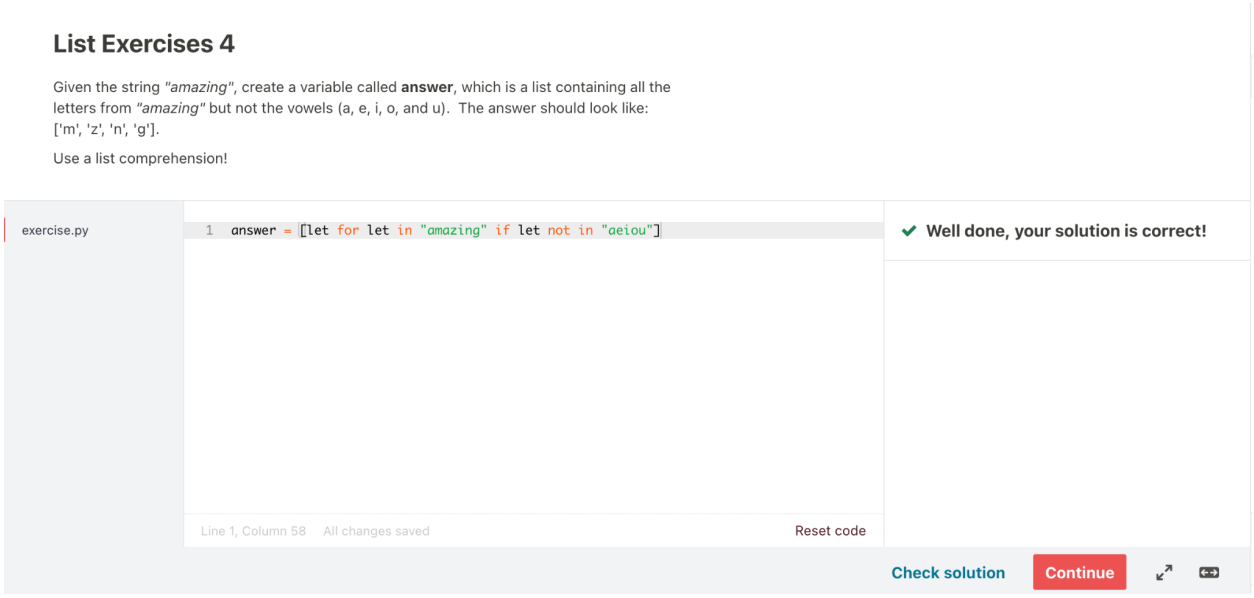
exercise.py

```
1  answer =  [n for n in range(1, 101) if n % 12 == 0]
2
3
```

Line 1, Column 51 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue

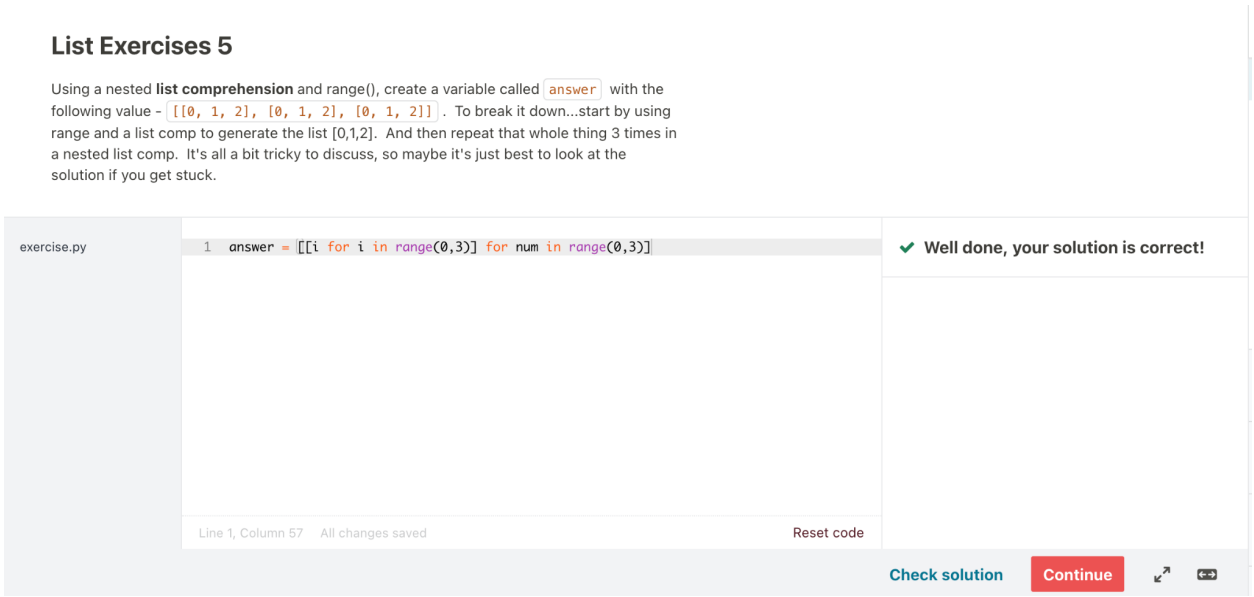


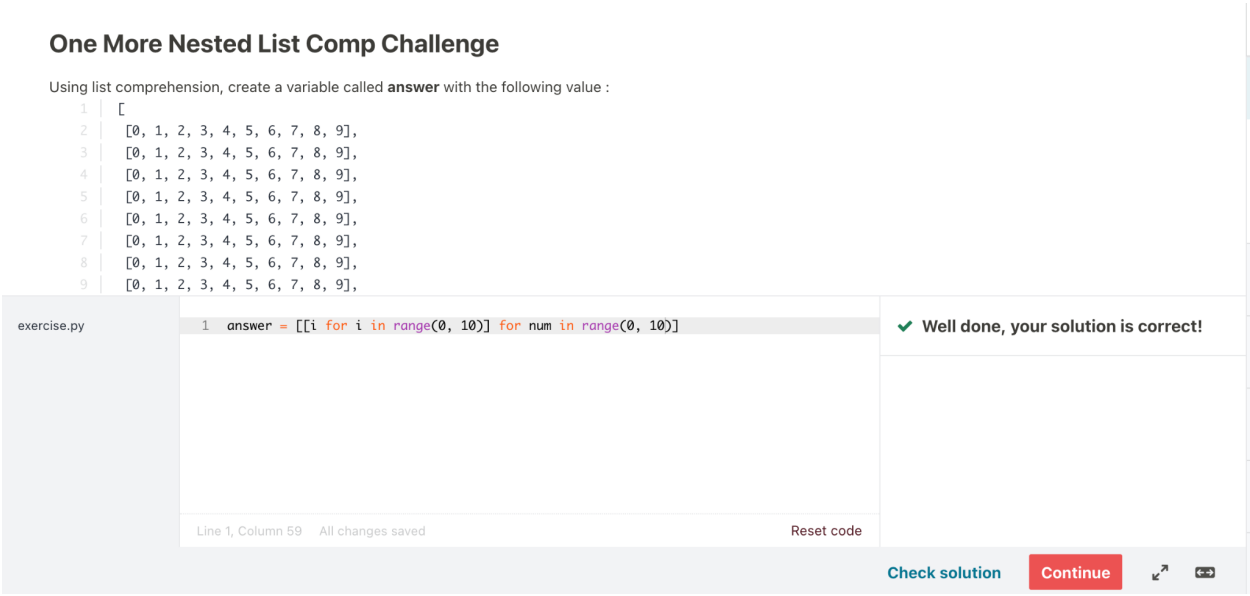
Nested Lists:

Any list can contain any sort of elements, whether it's strings, numbers, booleans, other lists, or dictionaries.

Why? Complex data structures—matrices, game boards/mazes, rows and columns for visualizations, tabulation and grouping data.

i.e. nested = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[print(val) for val in l] for l in nested]





Recap:

- Lists are fundamental data structures for ordered information
- List can include any type, even other lists or dictionaries
- We can modify lists with a variety of methods
- Slices are quite useful when making copies of lists
- List comprehension is used everywhere when iterating over lists, strings, ranges, and even more data types
- Nested lists are essential for building more complex data structures like matrices, game boards, or mazes.
- Swapping is quite useful when shuffling or sorting

Dictionaries

Objectives:

- Describe, create, and access a dictionary data structure
- Use built-in methods to modify and copy dictionaries
- Iterate over dictionaries using loops and dictionary comprehensions
- Compare and contrast dictionaries over lists

Lists are great, however there are some severe limitations. For example:

instructor = [‘Colt’, True, 4, ‘Python’, False]

This data does not contain enough information to fully explain what kind of list this is.

We want to be able to describe this data in more detail.

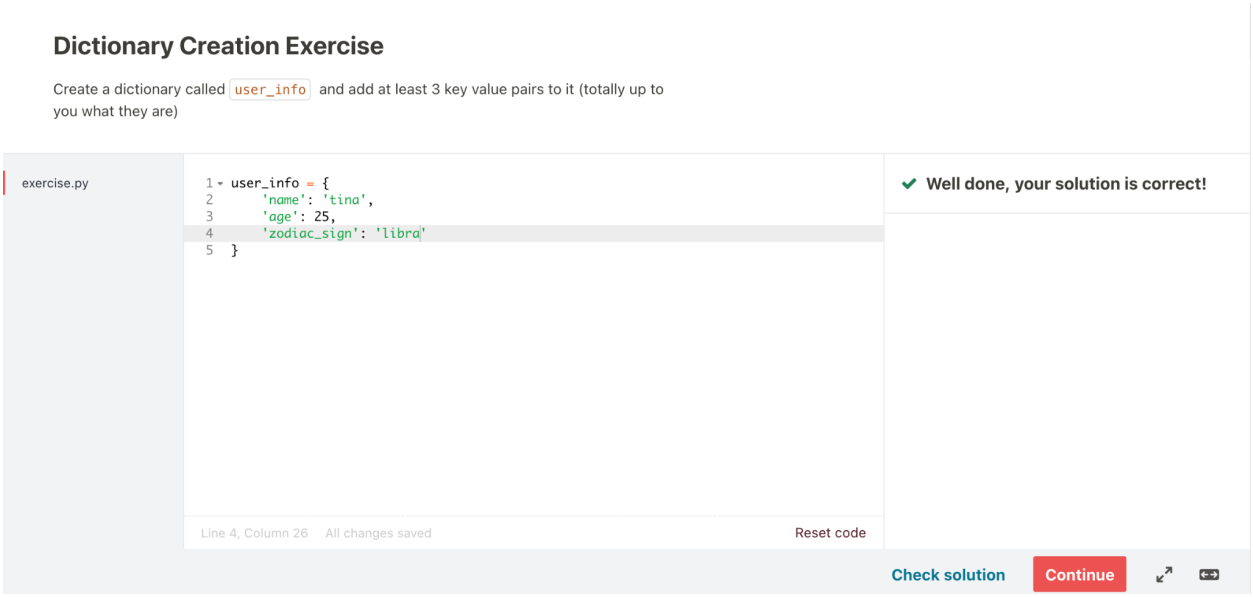
This is why we'd want to use a **dictionary**! It is a data structure that consists of key value pairs.

We can create a dictionary like this:

cat = {"name": "blue", "age": 3, "is_cute": True}

Another approach is to use the **dict** function. You assign values to keys by passing in keys and values separated by an =
For example,

another_dictionary = dict(key = 'value')



<u>Accessing Individual Values:</u>
In order to access individual values, we would do it as shown below:

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

information['name']   # outputs tina
information['thing']   # KeyError
```

Access Data in a Dictionary Exercise

Given the dictionary below:

```
artist = {
    "first": "Neil",
    "last": "Young",
}
```

Declare a variable called `full_name` that is equal to artist's `first` and `last` names with a space between. You must reference the values associated with those keys in the artist dictionary.

```
print(full_name)
# Neil Young
```

exercise.py

```
artist = {
    "first": "Neil",
    "last": "Young",
}

full_name = artist['first'] + ' ' + artist['last']
```

Line 6, Column 49 All changes saved Reset code

✔ Well done, your solution is correct!

Check solution Continue