RECORDING IN-PROGRESS

Day 7: Python Collections

Tuesday, 7.20.21

RECORDING IN-PROGRESS

Agenda

- Daily check-in
  - Announcements
  - Review solution to Lab 6
  - Talk through feedback from yesterday
- Recap for the functions we worked on yesterday
- Look at a little C++ vs. Python code
- New topic: Python collections
  - Lists
  - Sets
  - Tuples
  - Dictionaries

Daily check-in

Check in

- Questions?
- Announcements
  - Juhan’s slides from last week are in Canvas under Files / Faculty Lectures / Day 5
  - Our tech trek today (MIT Lincoln Lab, or “MITLL”) will be from 12:30 - 1:30pm EST, in Zoom
  - Thank you everyone for a great Google visit yesterday!
- Lab 6
  - Brooks will go over the solution + extra credit, please ask him questions

RECAP

Feedback from yesterday’s survey

Great job everyone! It seems like for the most part, everyone feels comfortable with many of the concepts we’ve been working on.

- **Most common helpful part of the program**: Labs + recaps each day
- **Most common difficult part of the program**: Labs (esp. 2048)

Remember: everyone has to start *somewhere* when learning a new skill.

Please continue to ask questions. We only have a few more days left, so if there’s anything you want to ask me about Python - please do! At the end of this week (Friday), I will spend more time on general advice for CS / learning to code.

Go over solutions from yesterday’s live coding

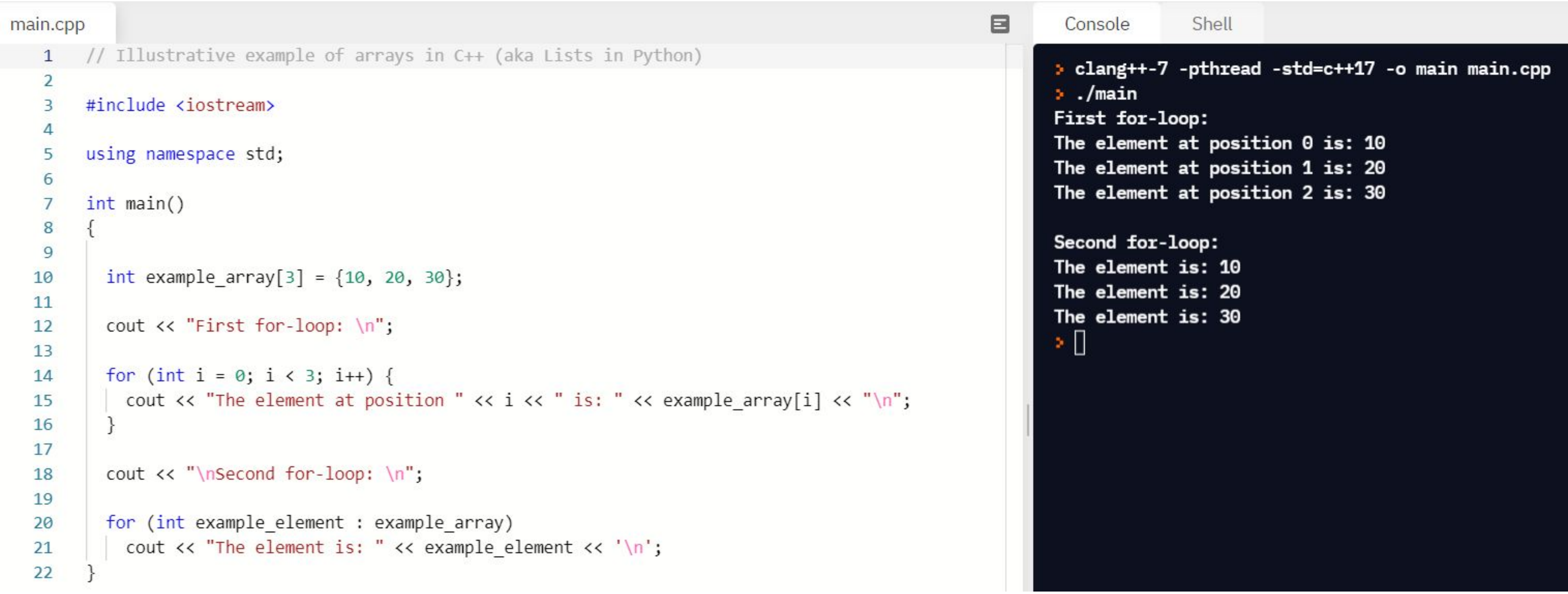
Context for the speed our C101 program has been going in: https://www.cs.tufts.edu/comp/11-2018s/calendar.shtml (*the problems we worked on in class yesterday were given to Tufts college students *3 weeks in* to their CS course! And they had a whole week to complete it*)

- Finish up the exercises we worked on yesterday in Replit, then come back to slides for new topic of the day

Practicing our coding (C++ vs. Python)

We’ll be learning more about collections in Python, so I thought it might be nice to see an example of a familiar collection data type in C++ (arrays/lists)

Example of lists (aka Arrays) in C++

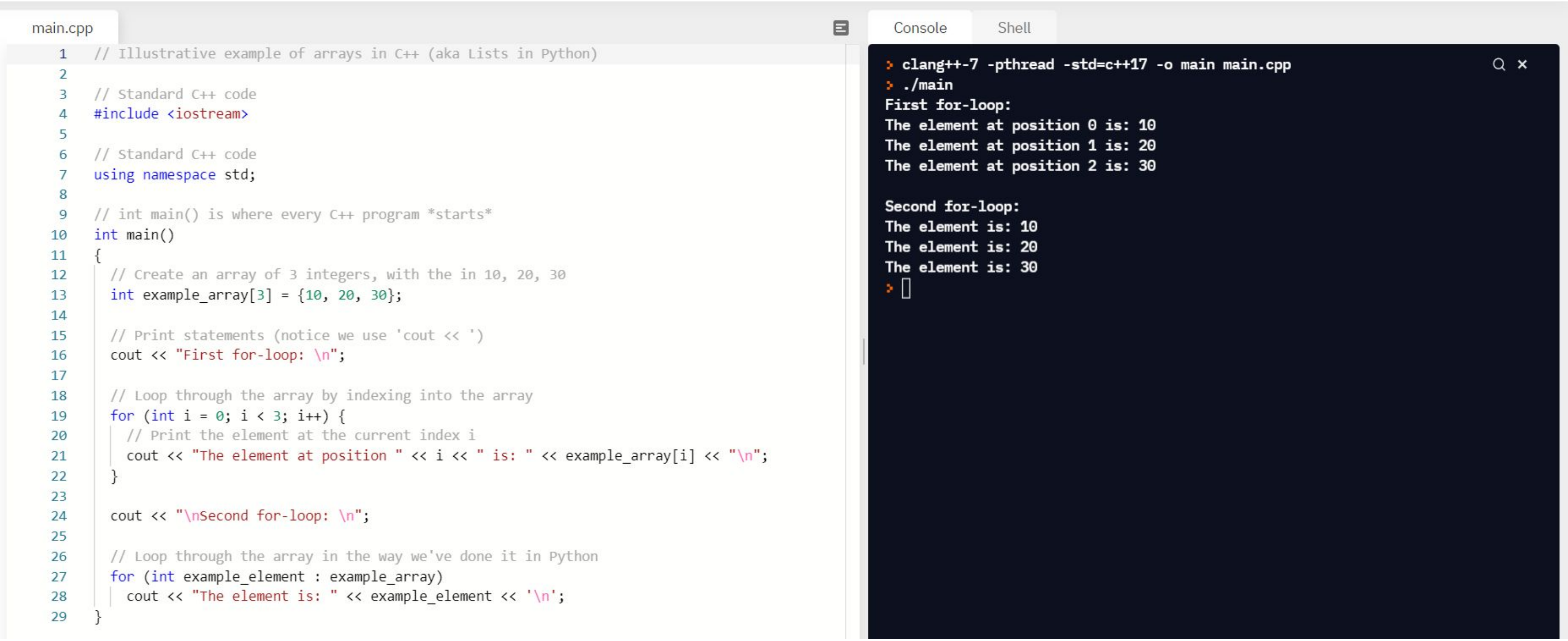


```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

#include <iostream>

using namespace std;

int main()
{

  int example_array[3] = {10, 20, 30};

  cout << "First for-loop: \n";

  for (int i = 0; i < 3; i++) {
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
> 
```

There are no comments in this code, let’s step through it line by line to see if we can figure it out first.

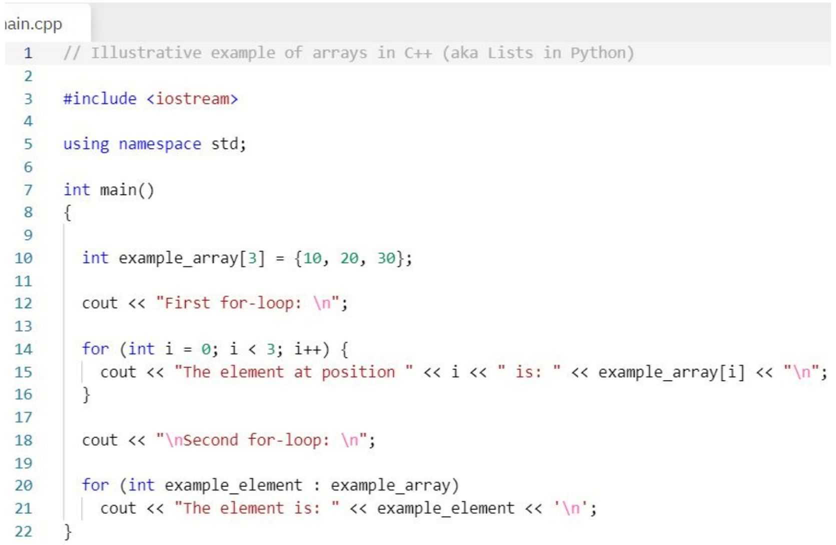
Example of lists (aka Arrays) in C++



```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

// Standard C++ code
#include <iostream>

// Standard C++ code
using namespace std;

// int main() is where every C++ program *starts*
int main()
{
  // Create an array of 3 integers, with the in 10, 20, 30
  int example_array[3] = {10, 20, 30};

  // Print statements (notice we use 'cout << ')
  cout << "First for-loop: \n";

  // Loop through the array by indexing into the array
  for (int i = 0; i < 3; i++) {
    // Print the element at the current index i
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  // Loop through the array in the way we've done it in Python
  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
> 
```

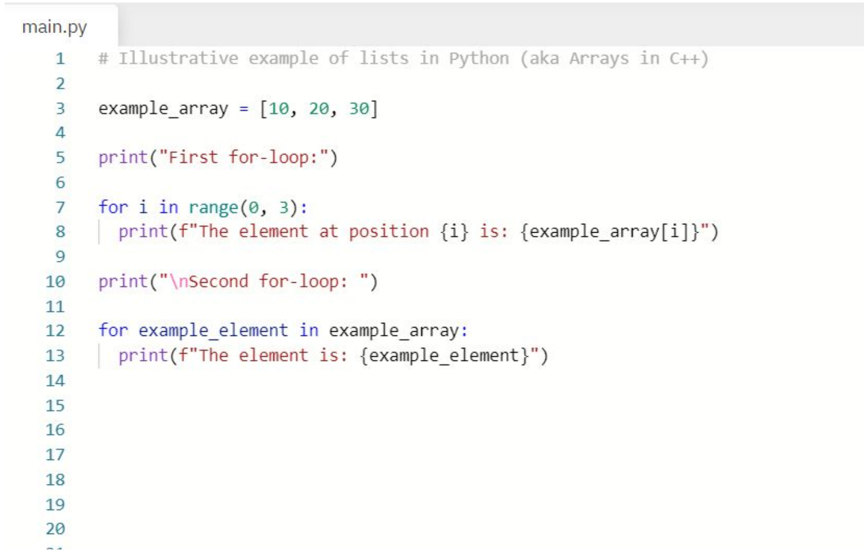
C++ vs. Python



ıain.cpp

```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

#include <iostream>

using namespace std;

int main()
{

  int example_array[3] = {10, 20, 30};

  cout << "First for-loop: \n";

  for (int i = 0; i < 3; i++) {
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

C++ version

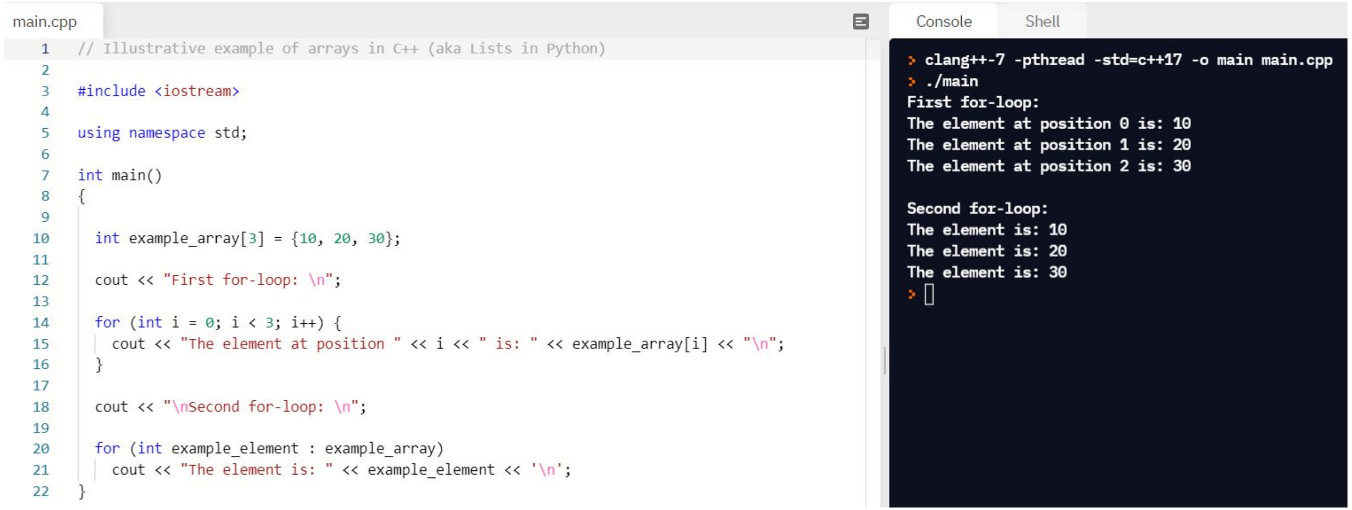


main.py

```python
# Illustrative example of lists in Python (aka Arrays in C++)

example_array = [10, 20, 30]

print("First for-loop:")

for i in range(0, 3):
  print(f"The element at position {i} is: {example_array[i]}")

print("\nSecond for-loop: ")

for example_element in example_array:
  print(f"The element is: {example_element}")
```

Python version

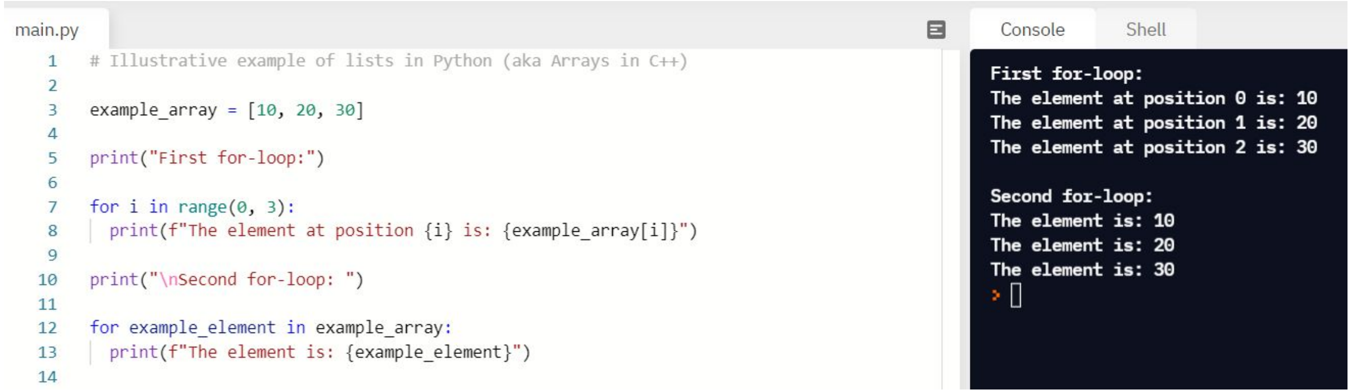
C++ version:



```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

#include <iostream>

using namespace std;

int main()
{

  int example_array[3] = {10, 20, 30};

  cout << "First for-loop: \n";

  for (int i = 0; i < 3; i++) {
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
> 
```

Python version:



```python
# Illustrative example of lists in Python (aka Arrays in C++)

example_array = [10, 20, 30]

print("First for-loop:")

for i in range(0, 3):
  print(f"The element at position {i} is: {example_array[i]}")

print("\nSecond for-loop: ")

for example_element in example_array:
  print(f"The element is: {example_element}")

```

```
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
> 
```

New topic: Python collections

I have to go over this topic somewhat quickly, I apologize.
I want to get to the live coding at the end of class.
Take short notes, and focus on absorbing the concepts by listening.

Python collections

Python collections (aka **containers** aka **collection data types**) give us a way to store values, or data, in specific ways. Depending on the collection type we use, the data will be stored, accessed, and updated differently.

Examples of Python collections:

- Lists
- Sets
- Tuples
- Dictionaries

Lists

Python collections shouldn’t scare you: we’ve already looked at Lists!

Given the list:

```
    players = [“Mario”, “Luigi”, “Yoshi”, [“Daisy”, “Peach”]]
```

What are some things we know about this list?

- **Is the list ordered?**
- **Can we access the values in the list?**
- **Can we update the values in the list?**
- **Can we have duplicate values in the list?**

Lists

Given the list:

```
    players = [“Mario”, “Luigi”, “Yoshi”, [“Daisy”, “Peach”]]
```

What are some things we know about this list?

- **Is the list ordered?**
  - Yes: our list always starts at the index number 0, and each element is ordered by its position in the list, e.g., 0, 1, 2, 3, ..., n-1
- **Can we access the values in the list?**
  - Yes: we can use the list’s index to access elements directly, e.g., players[0]
- **Can we update the values in the list?**
  - Yes: we can update any value in our list by directly updating the element, e.g., players[0] = “Wario”
- **Can we have duplicate values in the list?**
  - Yes: players = [“Mario”, “Mario”, ...]

Lists

These same questions we just answered about lists:

- **Is the list ordered?**
- **Can we access the values in the list?**
- **Can we update the values in the list?**
- **Can we have duplicate values in the list?**

Is how we differentiate between *all* of the data collections in Python.

We can answer those same questions not just for lists, but also for **sets**, **tuples**, and **dictionaries**.

Sets

Sets are generally the same across many modern programming languages.

The purpose of a set is to store a collection of values, where no value is repeated twice. In other words, sets do not allow duplicate values.

Given the set:

```
    example_set = {“Peach”, “Mario”, “Daisy”}
```

If I did `example_set.add(“Peach”)`, what do you think would happen?

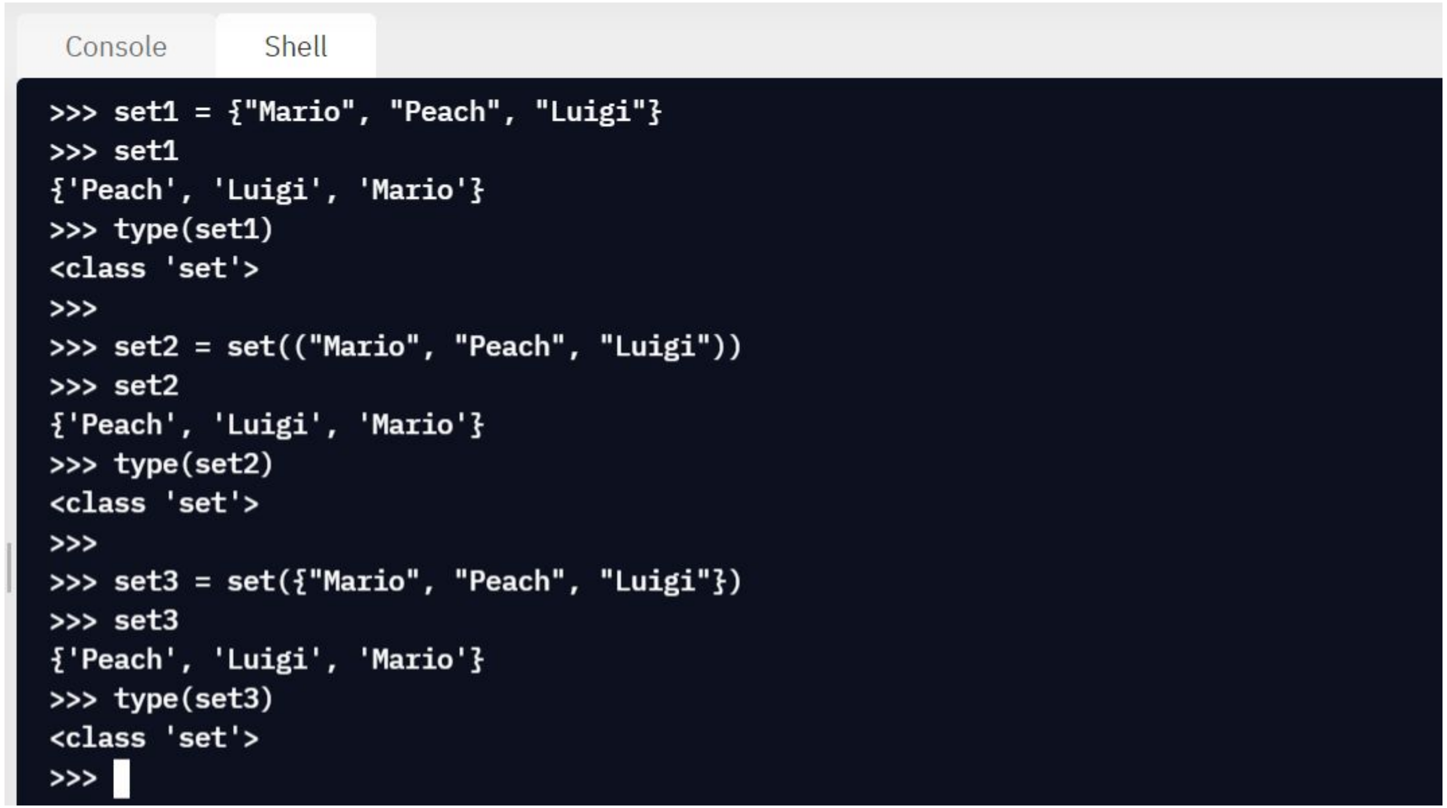
- Good to know: sets use curly brackets!
- Read about them more here: https://www.w3schools.com/python/python_sets.asp

Creating a set

Two ways to create a set in Python:

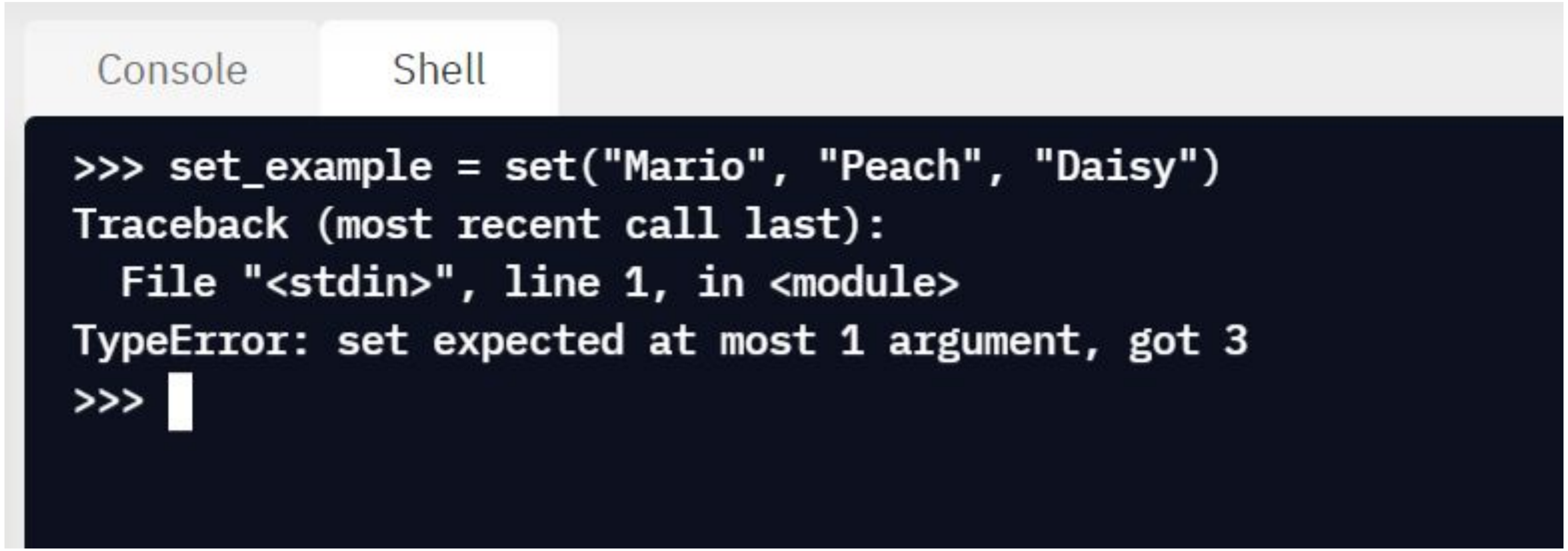
```
# Creating a set of 3 elements

# First way using curly brackets
players = {“Mario”, “Luigi”, “Yoshi”}

# Second way using set()
players = set({“Mario”, “Luigi”, “Yoshi”})
```

Creating sets (examples)



```
>>> set1 = {"Mario", "Peach", "Luigi"}
>>> set1
{'Peach', 'Luigi', 'Mario'}
>>> type(set1)
<class 'set'>
>>>
>>> set2 = set(("Mario", "Peach", "Luigi"))
>>> set2
{'Peach', 'Luigi', 'Mario'}
>>> type(set2)
<class 'set'>
>>>
>>> set3 = set({"Mario", "Peach", "Luigi"})
>>> set3
{'Peach', 'Luigi', 'Mario'}
>>> type(set3)
<class 'set'>
>>>
```

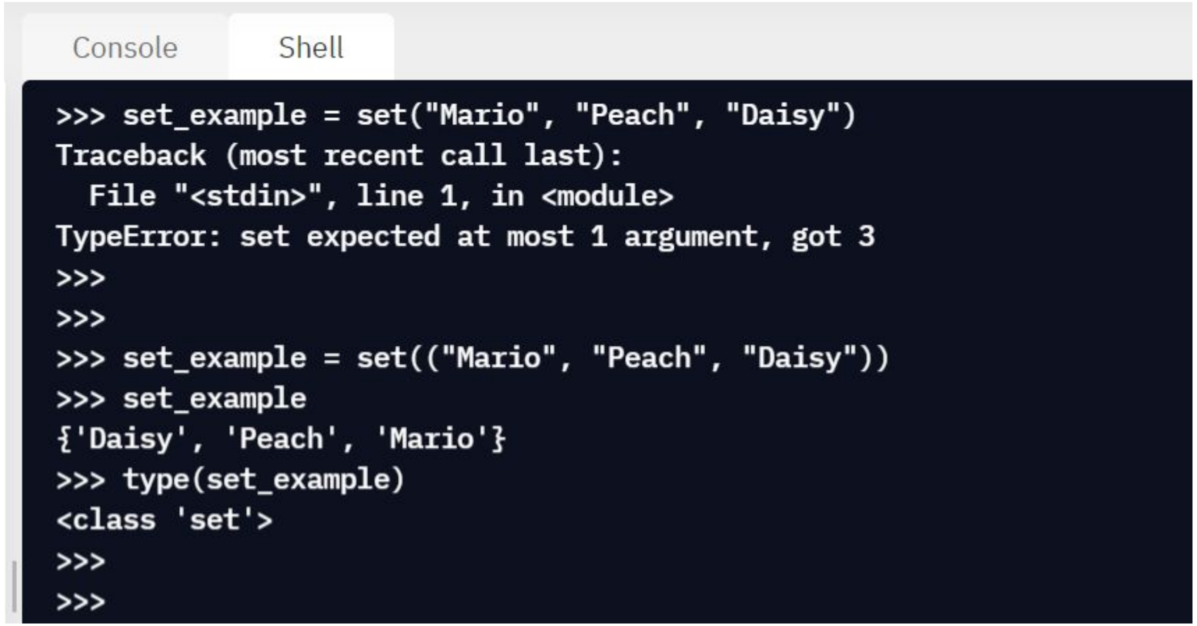
Creating sets (examples)



```
>>> set_example = set("Mario", "Peach", "Daisy")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: set expected at most 1 argument, got 3
>>>
```

Why do you think we get this error?

Creating sets (examples)



```
>>> set_example = set("Mario", "Peach", "Daisy")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: set expected at most 1 argument, got 3
>>>
>>>
>>> set_example = set(("Mario", "Peach", "Daisy"))
>>> set_example
{'Daisy', 'Peach', 'Mario'}
>>> type(set_example)
<class 'set'>
>>>
>>>
```

Fixing our error

Tuples

Tuples are similar to Lists, except we can’t update or change the values in our Tuples -- unlike Lists.

https://www.w3schools.com/python/python_tuples.asp

Creating a tuple

Two ways to make a tuple in Python:

```
# Creating a tuple

# First way using rounded brackets
players = (“Mario”, “Peach”, “Yoshi”)

# Second way using tuple()
players = tuple((“Mario”, “Peach”, “Yoshi”))
```

Tuples

So why would we ever use tuples? Sometimes we *want* the data to be immutable, or unchangeable. This ensures certain safeties and security in our code, which we may care about when writing large or complicated coding projects.

For example, we may store data in a program that we do *not* want to get changed in our code - ever! And if it does somehow get changed, or we try to attempt to change it… we want Python to give us an error. By using tuples, we can ensure that no where in our program will the values in our tuple accidentally get changed.

Dictionaries

Finally, the coolest Python collection - dictionaries (change my mind)

Dictionaries let us store data by assigning a unique “key” its own “value”:

```
dictionaryName[key] = value
```

- In the above code, we are assigning the value ‘value’ to the key ‘key’ in our dictionary, ‘dictionaryName’.

```
dictionaryName = {key1: value1, key2: value2, key3: value3}
```

- In the above code, we are assigning 3 values to 3 different keys in our dictionary, dictionaryName

https://www.w3schools.com/python/python_dictionaries.asp

Dictionaries

Examples:

```
players_ages = dict()          # create an empty dictionary

players_ages[“Mario”] = 45     # key is Mario, value is 45
players_ages[“Peach”] = 40     # key is Peach, value is 40
players_ages[“Yoshi”] = 100    # key is Yoshi, value is 100
```

Dictionaries

We don’t always need to create entries in our dictionary using “dictionaryName[key] = value”. We can also do the last slide like this:

```
# same dictionary as the last slide
players_ages = {“Mario”: 45, “Peach”: 40, “Yoshi”: 100}
```

Dictionaries

An inherent feature of dictionaries is that each dictionary can only have one unique key! That is, we can’t have any duplicate keys in our dictionary, just like sets.

We can have duplicate *values*, but we can only ever have one instance of a key.

```
players_age = dict()

players_age[“Mario”] = 45    # key is Mario, value is 45
players_age[“Peach”] = 45    # key is Peach, value is 45
players_age[“Yoshi”] = 45    # key is Yoshi, value is 45
```

Dictionaries

We can have duplicate *values*, but we can only ever have one instance of a key.

```
players_age = dict()

players_age[“Mario”] = 45
players_age[“Mario”] = 145

print(players_age[“Mario”])  # what will get printed?
```

Dictionaries

We can have duplicate *values*, but we can only ever have one instance of a key.

```
players_age = dict()

players_age[“Mario”] = 45
players_age[“Mario”] = 145


print(players_age[“Mario”])  # what will get printed? 145
```

Note that every (key, value) pair in a dictionary is actually a **tuple**!

Caveats

How will I know when to list a **list** vs a **set** or a **dictionary**??

Usually, these collections serve a very specific purpose. Sets can be really fast at finding whether a certain element exists in it - maybe we care about speed for accessing elements, in which case we might want to use a set. Dictionaries are really good at storing a value for a specific key, perhaps like a password (*value*) assigned to a person’s username (*key*). In “real-world” programming, all of these collections are very common! (However, their use comes with experience)

In today’s lab later, you’ll work on a few popular **search algorithms** to get a better sense of *why* we would use lists vs. sets vs. dictionaries. It should help you wrap your mind around these collection data types!

Live coding

Caveat: Knowing when to use **lists** vs **sets** vs **tuples** vs **dictionaries** is something that will come with more programming experience. You may never work with dictionaries or sets again until your next coding course, and that’s alright! All that I want you to take away from this lecture are the high-level *concepts* of these different collection types, so it’ll make more sense when you *have* to work with them in the future.

Why do we need different types of data collections?

Many real-life problems rely on us being able to contain the data on our computer, or in our code, in a certain way. Sometimes for security purposes, other times for efficiency / optimization purposes.

Things we might care about when choosing a certain data collection:

- Performance (how fast can my computer access the data?)
- Flexibility (how much work do I need to do to update the data?)
- Stability (how can I prevent overwriting important data?)
- Security (how can I make sure data is available to the right people?)