General Assembly

Intro to Python

GENERAL ASSEMBLY

Our Goals for Today

- Use Jupyter Notebook to create and run Python programs.
- Understand fundamental Python syntax.
- Distinguish between data types in Python.
- Organize data using dictionaries and lists.
- Begin building Python skills through code challenges.

Intro to Python

Python Syntax

Guided Walk-Through:

1.1 Our First Jupyter Cell

10 Minutes

Throughout our Python journey, we'll be using an interactive Python environment called **Jupyter Notebook** to accompany our lessons with coding exercises.

Let's open the notebook associated with this lesson and execute the first cell to learn more about Python's founding principles, known as the Zen of Python.

Discussion:
Why Python?

There are plenty of great programming languages out there.

Why are we learning Python?

 © 2020 General Assembly

Why Python?

- Python has grown as a high-level, general-purpose programming language with a **huge open-source community** supporting it.
- It’s the fastest-growing programming language on the market, especially within the data analysis community.
- Python's primary advantages:
  - **Clean syntax**.
  - **A wealth of specialized, pre-built libraries**, such as Pandas, for data analysis, and Django, for web development.

Introduction to Python

Variables & Data Types

Variables

Variables are names that have been assigned to specific values or data.

Python allows us to easily define and redefine variables using a simple **assignment operator** (equals sign).

```
best_programming_language = "python"
```

 © 2020 General Assembly

Restrictions on Variables

- Variable names **cannot be just a number** (i.e., 2, 0.01, 10000).
- Variables **cannot be assigned the same name as a default or imported function** (i.e., “type,” “print,” “for”).
- Variable names **cannot contain spaces.**

Best Practices for Variables

- Variable names in Python should be **lowercase**.
- A variable's name should be **indicative of the concept it represents in your program**. Coming up with sensible variable names can take some time and thought, but this will save you a lot of confusion later on.
- If you have to include multiple words in your variable name, use an **underscore** to separate them. This is known as **snake case**.

Primitive Data Types in Python

	Explanation	Examples
String	A collection of characters representing a text-based value, such as a message.	`my_planet = "earth"` `secret_password = "password"`
Integer	Any whole number without decimal points.	`heist_members = 11` `bakers_dozen = 13`
Float	Numbers including points after the decimal, or "floating point" numbers.	`gigawatts = 1.21` `low_low_price = 49.99`
Boolean	The concept of true or false values.	`python_is_readable = True` `programming_is_simple = False`

 © 2020 General Assembly

Knowledge Check

Trivia!

Match the values on left with their correct data type on the right.

1. "Hello world"
2. 6
3. 7.2
4. True

A. Float
B. Boolean
C. String
D. Integer

Solo Exercise:

2. Introduce Yourself

10 Minutes

In Cell 2, we've already set up a message printed to the console. However, if we were to execute it right now, you'd see an error. To get this message to work, we have to define the following variables with string type values:

- `greeting`
- `name`
- `mood`

Don't worry about fully understanding what we've done to have the message show up — we'll be covering that throughout this lesson!

Introduction to Python

Manipulating Variables

Operators

We've already seen the **assignment operator** in action, but there are other operators that can modify variable values:

Symbol	Name	Explanation
+	Addition	Adds numbers or strings.
-	Subtraction	Subtracts numbers.
*	Multiplication	Multiplies numbers or strings.
/	Division	Divides numbers.
%	Modulus	Produces the remainder from division.
**	Exponent	Raises the first number to the second number's power.

 © 2020 General Assembly

Concatenating Strings

You may have noticed that some of the operators, especially the addition operator, work on strings as well as numbers.

Adding two or more strings together is called **concatenation**.

```
beverage_type = "sparkling water"

flavor = "grapefruit"

favorite_drink = flavor + beverage_type
```

 © 2020 General Assembly

Discussion:

A Problem With Concatenation

```
beverage_type = "sparkling water"

flavor = "grapefruit"
```

How could we concatenate these two variables into a single variable named “favorite_drink”?

There’s a good chance that our first solution will end up being "grapefruitsparkling water.” How can we fix that awkward combination of words?

 © 2020 General Assembly

Concatenation vs. String Interpolation

You can imagine that concatenating multiple strings might result in some complex, hard-to-read statements. However, Python provides a way to inject directly, or **interpolate**, a variable directly into a string using **f-strings**.

```
greeting = "Hello there"

person = "Professor Park"

message = f"{greeting}, {person}."
```

The message variable equates to “Hello there, Professor Park.”

 © 2020 General Assembly

Updating Variables

Variables wouldn't be very useful if their values couldn't vary or change. You can use the assignment operator to re-assign values to an existing variable.

```
favorite_language = "SQL"

favorite_language = "python"
```

You can even use the previous value when re-assigning!

```
my_current_age = my_current_age + 1

# Happy Birthday!
```

 © 2020 General Assembly

A Quick Comment on Comments

You've also seen that our Jupyter Notebook contains some lines of text that start with a hashtag (#). These are known as comments.

Comments are used to provide explanations and guidance throughout a program; Python will not try to execute these as code.

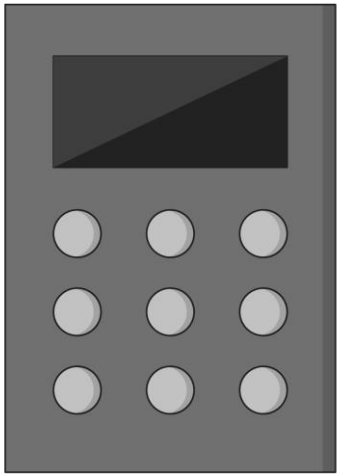
```
# This is a comment and will not cause errors!
```

Solo Exercise:

3. Operators & Updating Variables

10 Minutes

Let's practice with a birthday calculator trick in the Jupyter Notebook by following the instructions in the comments.



 © 2020 General Assembly

All the Values That Are Fit to print()

We've seen a few examples using the print() function so far.

We'll learn more about functions later on, but print() is an important first function to be aware of. It allows us to output messages to the console, which can be extremely valuable for debugging a program.

If something isn't working, we can always use some print() statements to investigate whether or not our variables contain the values we expect.

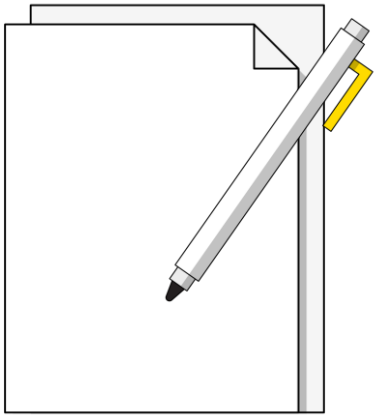
 © 2020 General Assembly

Group Exercise:

4. Python Mad Libs

15 Minutes

In this exercise, we'll practice combining strings together and including variables in string messages.



Introduction to Python

Changing Data Types

You're Just Not My type()

One significant cause of errors in Python occurs when you're trying to perform an operation on variables of different types:

```
2 + "apple"

# This will give you an unsupported operand error!
```

If you're unsure what data type a variable contains, you can use the type() function to investigate, especially in combination with print().

```
print( type(mystery_variable) )
```

 © 2020 General Assembly

...But I can change!

We can overcome incompatible data types by **type casting**, or changing the data type of a variable:

```
str(2) + "apple"

# This turns 2 into a string
```

This technique has its limits, however. If we tried converting “apple” to an integer, we’d get another error, as that just doesn’t make sense.

 © 2020 General Assembly

Discussion:

Compatible Types

Of the four data types we've learned so far, which do you think can be converted into each other and which cannot?

- Strings
- Integers
- Floats
- Booleans

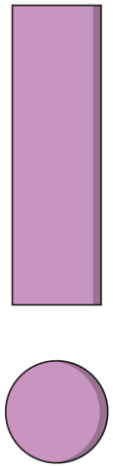
 © 2020 General Assembly

Solo Exercise:

5. Code Fast & Break Things

15 Minutes

Dealing with errors is part of everyday life in Python. Let's explore some common mistakes in Section 5 of the Jupyter Notebook.



 © 2020 General Assembly

Group Exercise:

6. How many ways to print with variables?

15 Minutes

Researching how to do something new is an irreplaceable skill in programming. Even if you don't totally understand what's going on, you still need to be able to find code snippets in documentation and use them to solve problems.

Section 6 of the Jupyter Notebook challenges us to find four distinct methods of achieving the same objective — save us, Stack Overflow!

Intro to Python

Data Structures

Python Syntax

Data Structures

Control Flow

Functions & Code Challenges

What's Next

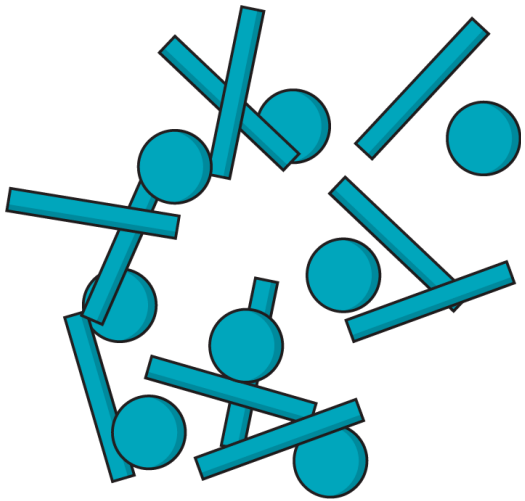
- Organize data using dictionaries and lists.

Our Sad, Unstructured Data

So far, our variables have only stored a single piece of information or data. Imagine trying to represent a data set using this method:

```
customer_one = "Anees Rosario"
customer_two = "Alya Pham"
customer_three = "Marc Wormald"
customer_four = "Ellie-Mai Muir"
```

You won't get very far using a new variable for every new piece of data!



Data Structures to the Rescue

Fortunately, Python provides us with some options for compiling data into a single structure:

- **Lists** are exactly what they sound like: comma-separated lists of values.
- **Tuples** are like lists, but more strict. You can't update the values in a tuple!
- **Dictionaries** allow us to associate multiple properties together, much like a single row of a spreadsheet can contain many columns.

Discussion:

Guess the Data Structure

Before we get into the specifics, let's think about which of our three data structures (lists, tuples, and dictionaries) make the most sense when representing the following information:

- The five continents on Earth.
- An item for sale in our store.
- All transactions conducted in our store.
- A single transaction conducted in our store.
- A weather forecast for today.
- The three possible quality rankings for produce.

 © 2020 General Assembly

Discussion:

Data Structures Revealed

Tuples hold values that will not change, while lists can be updated more easily. Dictionaries represent a single but complex piece of information.

- The five continents on Earth: **Tuple**.
- An item for sale in our store: **Dictionary**.
- All transactions conducted in our store: **List**.
- A single transaction conducted in our store: **Dictionary**.
- A weather forecast for today: **Dictionary**.
- The three possible quality rankings for produce: **Tuple**.

Data Structures

Lists & Tuples

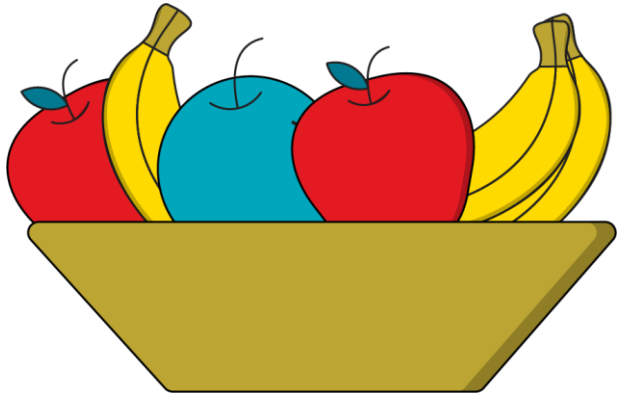
Lists

A list is an ordered collection of data combined into one variable.

Each item in a list is assigned an **index** value based on its position. These index values allow us to access individual elements within the list.

```
["banana", "orange", "apple"]
```

0 **1** **2**



 © 2020 General Assembly

Syntax

```
fruits = ["banana", "orange", "apple"]
```

You create a list using a set of square brackets.

Inside the brackets, each value must be separated by a comma.

Although it most often makes sense for all values to be the same data type, there's no rule against using a list to store data of varying types.

Accessing List Values

```
fruits = ["banana", "orange", "apple"]
# fruits[0] will access "banana"
# fruits[1] will access "orange"
# fruits[2] will access "apple"
```

Access list items by using square brackets around their index values. It's pretty simple — just remember that the first index value is always zero!

 © 2020 General Assembly

Updating List Values

```
fruits = ["banana", "orange", "apple"]
fruits[0] = "kiwi"
fruits[1] = "strawberry"
# The fruits list now looks like ["kiwi", "strawberry", "apple"]
```

We can update the items in a list using the same index syntax. The same assignment operator that we used with individual variables can also be used to set the value of a specific item in a list.

Solo Exercise:

7. Colors of the Rainbow

15 Minutes

Let’s practice creating a list and accessing and updating values using the specific list index syntax.

List Methods

In addition to using index syntax to refer to specific items, we can also use **methods** of lists to perform specific actions on that list. The syntax looks like:

```
list_name.method_name( any_inputs_here )
```

The parentheses might remind you of the print statements we've used thus far. That's because, just like print(), methods can also accept specific input, or **arguments**, for their operations.

Adding Items With `.append()`

```
fruits = ["orange"]

fruits.append("kiwi")
# fruits is now: ["orange", "kiwi"]
```

The `.append()` method adds the item inside the parentheses to the **end** of the list.

 © 2020 General Assembly

Adding Items With `.prepend()`

```
fruits = ["orange"]

fruits.prepend("lemon")
# fruits is now: ["lemon", "orange"]
```

The `.prepend()` method adds the item inside the parentheses to the **beginning** of the list.

Adding Items With `.insert()`

```
fruits = ["orange", "kiwi"]

fruits.insert(1, "lemon")
# fruits is now: ["orange", "lemon", "kiwi"]
```

The `.insert()` method starts by taking an index number, then adds the second argument in the parentheses to the given index.

Note the original item at that index is simply bumped to the next spot in the list.

Removing Items With `.pop()`

```
fruits = ["orange", "kiwi", "lime"]
fruits.pop()
# fruits is now: ["orange", "kiwi"]
fruits.pop(0)
# fruits is now: ["kiwi"]
```

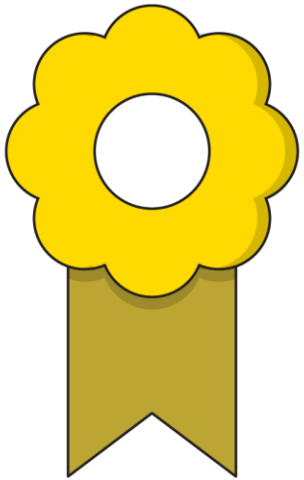
The `.pop()` method can be used to remove the item at a specific index; if you don't provide any index, it will simply remove the last item in the list.

Partner Exercise:

8. The Wait List

20 Minutes

Let's practice these list methods and more by managing the waiting list for the Python Academy, the most exclusive private boarding school in all of East Python-shire.



More List Functions

	Explanation	Example

`len()`	Produces the length of the list.	`fruits = ["apples", "bananas", "oranges"]`
		`len(fruits)`
		`# 3`

`min()` and `max()`	Produces the minimum or maximum.	`scores = [10, 20, 30]`
		`min(scores)`
		`# 10`
		`max(scores)`
		`# 30`

`sum()`	Produces the total sum of all items in the list.	`sales = [25, 15, 10]`
		`sum(sales)`
		`# 50`

Tuples: You Already Know What They Are

The good news about tuples is that they work exactly the same way as lists, but simply use parentheses instead of square brackets.

```
valid_statuses = ("operational", "faulty", "non-operational")
```

The difference is that tuple values are **immutable**, meaning you cannot change values in a tuple once it's been created; thus, tuples are mostly used for reference information that will not change.

Sets

A set is an unordered collection of unique values. You can't use an index to access a specific element in a set. Instead, sets have methods for accessing or manipulating collection members:

```
primary_colors = { "red", "blue", "yellow" }

primary_colors.add("green")

primary_colors.remove("yellow")
```

Data Structures

Dictionaries

Dictionary Definition

A dictionary is a collection of associated **keys** and **values**. Think of a dictionary like a row of data in a spreadsheet — you have column names, which are your keys, and then you have the actual values inside of the columns.

item_name	category	price
tomatoes	food	1.99

```
# The above row would be translated into this dictionary

{ "item_name": "tomatoes", "category": "food", "price": 1.99 }
```

Syntax for Dictionaries (Cont.)

```
item = { "item_name": "tomatoes", "category": "food", "price": 1.99 }
```

Curly braces are used to start and end the dictionary.

 © 2020 General Assembly

Syntax for Dictionaries (Cont.)

```
item = { "item_name": "tomatoes", "category": "food", "price": 1.99 }
```

Key names are provided first and surrounded by quotation marks.

Syntax for Dictionaries (Cont.)

```
item = { "item_name": "tomatoes", "category": "food", "price": 1.99 }
```

Values are provided after a colon and can be of any data type.

Syntax for Dictionaries (Cont.)

```
item = { "item_name": "tomatoes", "category": "food", "price": 1.99 }
```

Finally, note the **commas** separating each key-value pair.

 © 2020 General Assembly

Accessing Values in a Dictionary

```
item = { "item_name": "tomatoes", "category": "food", "price": 1.99 }
```

Accessing specific values in a dictionary works a lot like accessing specific items in a list. This time, instead of numbered indices, we use the name of the key.

```
item["category"]

# This accesses the value "food"
```

 © 2020 General Assembly

Adding Values to a Dictionary

One of the most powerful features of a dictionary is the ability to add new keys and values whenever necessary.

```
item["fresh"] = true
```

```
item["discount"] = .15
```

The same syntax can update a given property as well.

```
item["fresh"] = false
```

 © 2020 General Assembly

Solo Exercise:

9. Key Value Properties

15 Minutes

Practice accessing and updating properties in a dictionary by modifying a real estate listing in Section 10 of the Jupyter Notebook.

Discussion:

Representing a Data Set

Based on what we know about data structures, **how would we represent an entire data set (like a .csv file)**, with many rows and columns of information, **in Python?**

item_name	category	price
tomatoes	food	1.99
mango	food	3.00
journal	office	15.00

 © 2020 General Assembly

Discussion:

Representing a Data Set

Answer: A list of dictionaries! Each dictionary is a specific row, and the list is our “table,” collecting all of the rows together.

```
[ {"item_name" : "tomatoes", "category": "food", "price" : 1.99},

{"item_name" : "mango", "category": "food", "price" : 3.00},

{"item_name" : "journal", "category": "office", "price" : 15.00} ]
```

 © 2020 General Assembly

Data Structures

Combining Data Structures

Nested Data Structures

So far, we've seen lists and dictionaries that organize simpler data types, like strings and numbers. However, it's common to see the more complex data structures nested within each other, creating elaborate mazes of data.

When getting data from outside sources, for example, you might be given a dictionary that contains another dictionary that contains a list containing another dictionary that finally contains the information you're looking for!

 © 2020 General Assembly

Discussion:

What is this accessing?

Think about what the following line of code is saying, one layer at a time:

```
authors[0]["books"][1]["title"]
```

Without seeing the exact data structure in question, what is this line attempting to access?

 © 2020 General Assembly

Discussion:

Peeling Back the Layers

```
authors[0]

# There is a list called authors and we want the first thing in that
list.
```

Discussion:

Peeling Back the Layers

```
authors[0]["books"]

# There is a list called authors and we want the first thing in that
list.

# That first thing is a dictionary and we want the "books" property.
```

 © 2020 General Assembly

Discussion:

Peeling Back the Layers

```
authors[0]["books"][1]

# There is a list called authors and we want the first thing in that list.

# That first thing is a dictionary and we want the "books" property.

# Turns out "books" is a list and we want the second thing in it.
```

 © 2020 General Assembly

Discussion:

Peeling Back the Layers

```
authors[0]["books"][1]["title"]

# There is a list called authors and we want the first thing in that
list.

# That first thing is a dictionary and we want the "books" property.

# Turns out "books" is a list and we want the second thing in it.

# That second thing is another dictionary with a "title" property.
```

 © 2020 General Assembly

Keep Calm and print() On

When dealing with a complex, multi-level data structure, you can lean on print statements to help peel back the layers one at a time. The syntax for accessing elements doesn't change; you just simply need more layers:

```
print(authors[0])

print(authors[0]["books"])

print(authors[0]["books"][1])

print(authors[0]["books"][1]["title"])
```

 © 2020 General Assembly

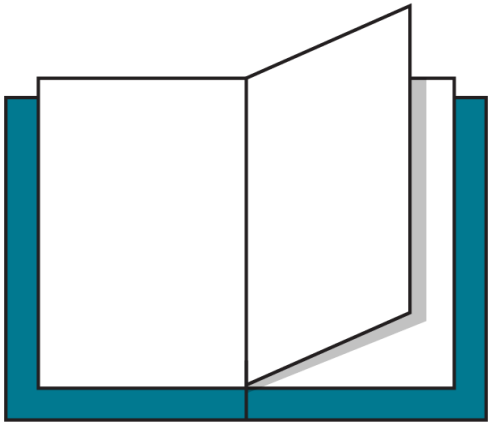
Partner Exercise:

10. Python Librarian

15 Minutes

Managing nested data structures can be difficult.

Work through the challenges of accessing and modifying the “authors” dictionary found in Section 10 of the workbook.



Questions?


GA