Intro to Analytics With Python

Table of Contents

Literacy

By the end of these classes, you won't be a pro Python dev.

You don't have to be;
The only *true* skill is knowing how to keep learning.

If you have the words to talk about your tools and an idea of how they work, becoming more skilled with them will only get easier.

Today's Goal

Our goal today is to get comfortable reading and writing basic Python. **Learning new things can be uncomfy and nerve-wracking, but we will take it slow and I promise it will be worth it.** Please ask millions of questions and stop me whenever you want.

DISCLAIMER: We might not make it to Pandas today, but worry not! We'll cover it plenty next week, and you'll be super equipped to learn more on your own going forward.

Abstraction

noun
/abˈstrakSH(ə)n/
the quality of dealing with ideas rather than things or events

Abstractions make it possible for us to invent, design, and imagine.

Spoken & written languages allow us to create completely arbitrary abstractions.

Have you ever read a fantasy novel?

Fairies ✧ and dragons □ are not real, but **words,** packaged into various sized bundles of ideas like **sentences**, **paragraphs,** and **storylines** allow us to easily experience the vast, imagined worlds that authors create. Abstractionception.

💰 Money is an abstraction for the value of labor and materials.
💬 The user-interface of a messaging app is an abstraction for a conversation.
😊 Emojis are abstractions for the non-verbal parts of conversation, like feelings.

Literally all programming languages are just abstractions that we can use to express ideas about data and processes performed on it.

When coding, focus on what you are trying to **express** rather than what you're trying to "make Python do."

Abstractions In Python

[interactive demo time]

See `cheatsheet.py`

We are gonna copy code out of that file and into a terminal or Jupyter notebook. (Your choice. You can also choose to just watch)

Brief summaries of the info in that file are in the next slides, but refer to that for more detail. Links are included in the slides to the Official Python Docs.

You are required **by law** to fall deeply in love with the Official Python Docs.

Variables are containers for nearly anything you can express in code. They are the names we use to refer to the data we work with in programs.

```
text = "this is some text"
number = 124
flag = True
millions_of_rows = db.execute(
    "SELECT * FROM huge_table"
    )
```

Data structures are awesome. Think about them all the time.

```
webster = {
    "petrichor": "the smell of freshly fallen rain",
    "verdant": "having green coloration",
    "sufjan": "a sad alien who loves god.",
}

webster["petrichor"]
```

Dicts let you store collections of stuff and access them with arbitrary keys. Lists do what you think they do.

Logical Operators (or Comparators) are simple tools for comparing values. They always return true or false.

```
5 == 5 # returns True
5 != 5 # returns False
4 > 3  # returns True
1 < 0  # returns False
2 >= 1 # returns True
1 <= 1 # returns True
```

Logical Operators (continued)

The operators from the last slide can be chained together

```
answers = ["yes", "maybe"]
answer[0] != "no" and answer[0] != ""
answer[1] == "yes" or answer[1] == "maybe"
```

The first expression returns True.
The second returns True as well.

Logical Operators (last bit I promise)
Control the execution of a program with if/else and logic

```
best_rapper = "Kendrick"
now_playing = ""

if now_playing == best_rapper:
    return "Nice. You have good taste."
else:
    return "You should probably be listening to Kendrick."
```

Functions are the key to code that makes sense and is useful. They allow us to package up up arbitrary, reusable logic and processes. Good functions are "pure" meaning the return something and that the return value will always be the same given the same inputs

```
def add_one(x):
    return x + 1

add_one(5)

def build_rapper(name, age, rap_name):
    print("You've made a rapper named " + rap_name + ".")
    return {"name": name, "age": age, "rap_name": rap_name}
```

Functions are the key to code that makes sense and is useful. They allow us to package up up arbitrary, reusable logic and processes. Good functions are "pure" meaning the return something and that the return value will always be the same given the same inputs

```
def add_one(x):
    return x + 1

add_one(5)

def build_rapper(name, age, rap_name):
    print("You've made a rapper named " + rap_name + ".")
    return {"name": name, "age": age, "rap_name": rap_name}
```

Classes are packages of functionality like modules. Both use dots to access functionality inside them. You know a class when you see it, because it's got a *TitleCasedName*. Using the type() function will let you see the name and find out if something is a class (or any other type.)

```
my_dog = Dog()

my_dog.bark()
my_dog.set_bark("YEEHAW")
my_dog.bark() # YEEHAW

another_dog = Dog(name="Woofer", bark_sound="yip yip yip")
another_dog.name # Woofer
another_dog.bark() # yip yip yip
```