

Going Beyond an Initial Python Script

How to Structure Python Code and Common Operations

Nick DeRobertis¹

¹University of Florida
Department of Finance, Insurance, and Real Estate

September 20, 2020

Table of Contents

- 1 Introduction
- 2 Conditionals
- 3 More with Lists
- 4 Functions
- 5 More about Data Types
- 6 Classes and Dataclasses
- 7 Error Handling

An Organized Structure of an Advanced Python Model

- Now we are going to build our first complex Python model
- We will also learn a bit more Python before we can get there
- Just as we did in Excel, we need to add structure to make the model navigatable
- Logic should be organized in functions and be documented



The Structure of a Complex Model



Table of Contents

- 1 Introduction
- 2 Conditionals**
- 3 More with Lists
- 4 Functions
- 5 More about Data Types
- 6 Classes and Dataclasses
- 7 Error Handling

Python Conditionals - If Statement

If Statements in Python

```
>>> if 5 == 6:
>>>     print('not true')
>>> else:
>>>     print('else clause')
>>>
>>> this = 'woo'
>>> that = 'woo'
>>>
>>> if this == that:
>>>     print('yes, print me')
>>> if this == 5:
>>>     print('should not print')

else clause
yes, print me
```

Explaining the If-Else Statements

- Use two equals signs to compare things (single to assign things)
- Else is equivalent to value if false behavior in Excel
- We can do a lot more than just set a single value, anything can be done in an if or else statement
- `elif` is a shorthand for else if, e.g. not the last condition, but this condition

Conditionals Example

Trying out Conditionals

- On [the course site](#), there is a Jupyter notebook called Python Basics containing all of the examples for today's lecture
- Now I will go through the example material under "Conditionals"

Python Conditionals Lab

Python Basics - Conditionals

- 1 The Jupyter notebook called Python Basics Lab contains all of the labs for today's lecture
- 2 Please complete the exercises under "Conditionals"

Table of Contents

- 1 Introduction
- 2 Conditionals
- 3 More with Lists**
- 4 Functions
- 5 More about Data Types
- 6 Classes and Dataclasses
- 7 Error Handling

Python Patterns - Building a List

List Building

```
>>> inputs = [1, 2, 3]
>>> outputs = []
>>> for inp in inputs:
>>>     outputs.append(
>>>         inp + 10
>>>     )
>>> outputs.insert(0, 'a')
>>> print(outputs)
['a', 11, 12, 13]
```

- Use `.append` to add an item to the end of a list
- Use `.insert` to add an item at a certain position

List Indexing and Slicing

- Index is base zero (0 means first item, 1 means second item)

```
>>> my_list = ['a', 'b', 'c', 'd']
>>> my_list[0]    # first item
'a'
>>> my_list[1]    # second item
'b'
>>> my_list[-1]   # last item
'd'
>>> my_list[: -1] # up until last item
['a', 'b', 'c']
>>> my_list[1:]   # after the first item
['b', 'c', 'd']
>>> my_list[1:3]  # from the second to the third item
['b', 'c']
```

Lists Example

Doing More with Lists

- We will keep working off of Python Basics.ipynb
- Now I will go through the example material under "Working more with Lists"

Python Lists Lab

Python Basics - Lists

- 1 Keep working off of Python Basics Lab.ipynb
- 2 Please complete the exercises under "Working with Lists"

Table of Contents

- 1 Introduction
- 2 Conditionals
- 3 More with Lists
- 4 Functions**
- 5 More about Data Types
- 6 Classes and Dataclasses
- 7 Error Handling

Functions - Grouping Reusable Logic

- In Python, we can group logic into functions
- Functions have a name, inputs, and outputs
- Functions are objects like everything else in Python

- ```
def my_func(a, b, c=10):
 return a + b + c
```

```
>>> my_func(5, 6)
21
```



# Functions Example

## Structuring Code using Functions

- We will keep working off of Python Basics.ipynb
- Now I will go through the example material under "Functions"

# Python Functions Lab

## Python Basics - Functions

- 1 Keep working off of Python Basics Lab.ipynb
- 2 Please complete the exercises under "Functions"

# Table of Contents

- 1 Introduction
- 2 Conditionals
- 3 More with Lists
- 4 Functions
- 5 More about Data Types**
- 6 Classes and Dataclasses
- 7 Error Handling

# What are Types?

- In Python, everything is an object except for variable names, which are references to objects
- Every object has a type. We have learned about strings, numbers, lists, and booleans (True, False)
- In the next section on classes, we will learn more about the relationship between the type and the object

# Formatting Python Strings

- You may have noticed that we can end up with a lot of decimals in Python output
- Further, you may want to include your results as part of a larger output, such as a sentence.
- For these operations, we have f strings: `f ' '`

## Example

```
>>> my_num = 5 / 6
>>> print(my_num)
0.8333333333333334
>>> print(f'My number is {my_num:.2f}')
'My number is 0.83'
```

# Numeric Types

- So far I have just said that numbers are a type in Python, but this is a simplification
- There are two main types of numbers in python: `float` and `int` corresponding to a floating point number and an integer, respectively
- An `int` is a number without decimals, while a `float` has decimals, regardless of whether they are zero
- For example, 3.5 and 3.0 are floats, while 3 is an `int`, even though `3.0 == 3` is `True`
- Usually, this doesn't matter. But to loop a number of times, you must pass an `int`

# Additional Built-In Types

- A tuple is like a list but you can't change it after it has been created (it is immutable)
- Tuples are in parentheses instead of brackets, e.g. ("a", "b")
- A dict short for dictionary, stores a mapping. Use them if you want to store values associated to other values
- We will come back to dicts later in the course, but I wanted to introduce them now as they are a very fundamental data type

# Data Types Example

## Understanding the Different Data Types

- We will keep working off of Python Basics.ipynb
- Now I will go through the example material under "Exploring Data Types"



# Python Data Types Lab

## Python Basics - Data Types

- 1 Keep working off of Python Basics Lab.ipynb
- 2 Please complete the exercises under "Data Types"

# Table of Contents

- 1 Introduction
- 2 Conditionals
- 3 More with Lists
- 4 Functions
- 5 More about Data Types
- 6 Classes and Dataclasses**
- 7 Error Handling

# Overview of Classes and Objects



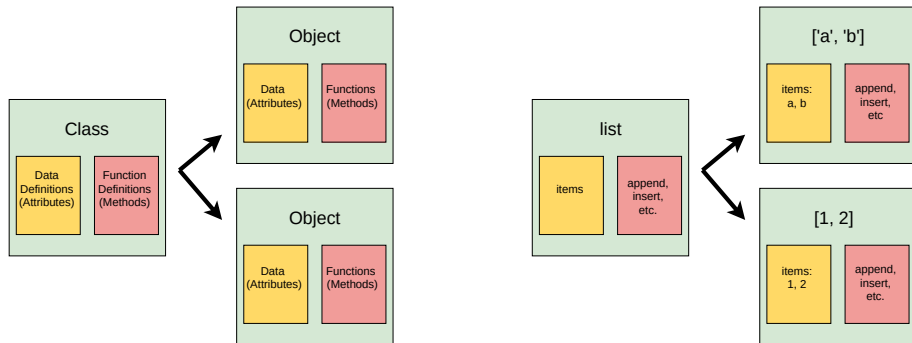
# Everything is an Object. Every Object has a Class

- In Python, everything is an object except for variable names, which are references to objects
- Strings, floats, ints, lists, and tuples are types of objects. There are many more types of objects and users can define their own types of objects
- A class is a definition for a type of object. It defines how it is created, the data stored in it, and the functions attached to it
- We can write our own classes to create new types of objects to work with

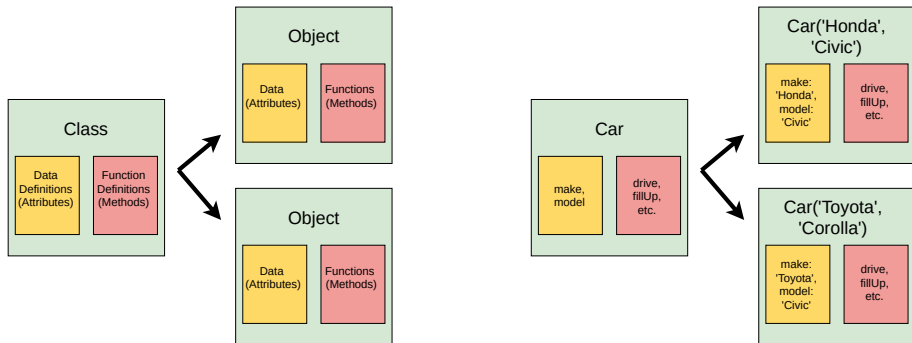
# Many Objects to One Class

- From a single class definition, an unlimited number of objects can be created
- Typically the class definition says it should accept some data to create the object
- Then when you have multiple objects of the same type (created from the same class), they will have the same functions (methods) attached to them, but different data stored within
- For example, we can create two different lists. They will have different contents, but we can do `.append` on either of the lists

# Lists are Objects



# We can Make Custom Objects Too



# Creating and Using Objects

- Constructing an object from a class looks like calling a function:

## Using Custom Classes in Python

```
from car_example import Car

>>> my_car = Car('Honda', 'Civic')
>>> print(my_car)
Car(make='Honda', model='Civic')
>>> type(my_car)
car_example.Car
>>> my_car.make
'Honda'
>>> my_car.drive()
'The Honda Civic is driving away!'
```



# Where we Will Focus in This Course

- I will not be teaching you about creating general classes in this course. It is very useful but is generally more advanced. I encourage you to learn them outside the course.
- We covered this material for two reasons:
  - To give a better understanding of how Python works in general, and why sometimes we call functions as `something.my_func()` rather than `my_func()`
  - We are going to use `dataclasses` to store our model data. They are a simplified version of classes used mainly for storing data.

# Dataclass Intro

- An organized way to store our model input data:

## Using Dataclasses in Python

```
from dataclasses import dataclass

@dataclass
class ModelInputs:
 interest_rates: tuple = (0.05, 0.06, 0.07)
 pmt: float = 1000

>>> inputs = ModelInputs(pmt=2000)
>>> print(inputs)
ModelInputs(interest_rates=(0.05, 0.06, 0.07), pmt=2000)
>>> type(inputs)
__main__.ModelInputs
>>> inputs.interest_rates
(0.05, 0.06, 0.07)
>>> inputs.pmt
2000
```

# What, When and Why Dataclasses?

- A dataclass is just a class which is more convenient to create, and is typically used to group data together
- If you need to pass around multiple variables together, they make sense. For our models, we will want to pass around all the inputs, so one dataclass for all the inputs to the model makes sense
- This way instead of having to pass around every input individually to every function, just pass all the input data as one argument
- Also enables easy tab-completion. What were the names of my inputs? Just hit tab after data.

# Classes Example

## Working with Classes and Creating Dataclasses

- We will keep working off of Python Basics.ipynb
- For this example, also go and download `car_example.py` and put it in the same folder
- Now I will go through the example material under "Working with Classes"

# Python Classes Lab

## Python Basics - Classes

- 1 Keep working off of Python Basics Lab.ipynb
- 2 Make sure you have car\_example.py in the same folder
- 3 Please complete the exercises under "Working with Classes"

# Table of Contents

- 1 Introduction
- 2 Conditionals
- 3 More with Lists
- 4 Functions
- 5 More about Data Types
- 6 Classes and Dataclasses
- 7 Error Handling**

# Python Error Handling

- You have certainly already seen errors coming from your Python code. When they have come up, the code doesn't run.
- Sometimes you actually expect to get an error, and want to handle it in some way, rather than having your program fail.

## Example

```
>>> my_list = ['a', 'b']
>>> try:
>>> my_value = my_list[10]
>>> except IndexError:
>>> print('caught the error')
caught the error
```

# An Example where Error Handling is Useful

- Let's say you're receiving annuities. There is a single annuity which produces \$100 for 5 years. You receive this annuity in year 0 and in year 3.
- You might define the annuity cash flows as a list of 100, 5 times (`[100] * 5`)
- Then you want to come up with your overall cash flows, going out to 15 years



# Applying Error Handling

## Calculating the Sum of Unaligned Annuity Cash-Flows

```
>>> annuity = [100] * 5
>>> annuities = [
>>> annuity,
>>> [0, 0, 0] + annuity
>>>]
>>> n_years = 10
>>> output = [0] * n_years
>>> for i in range(n_years):
>>> for ann in annuities:
>>> try:
>>> output[i] += ann[i]
>>> except IndexError:
>>> pass
>>> print(output)
[100, 100, 100, 200, 200, 100, 100, 100, 0, 0]
```