Python Fundamentals

An Overview
7.25.19

Link to Jupyter Notebook:

Agenda: Pilot Workshop

Time	Topic
30 min	Set-up, Introductions and Q/A
30 min	Python Review + Exercise 1
30 min	Data Structures + Control Flow
10 min	Exercise 2
60 min	User Defined Functions
20 min	Exercise 3

Kelly Ann Gracia

Data Analyst Consultant

Data Voyage Solutions

Tell me about yourself!

- Name and Role
- Experience with Python and/or other programming languages
- Daily Workflows / Data Analysis Workflow
    - (e.g., size of datasets; frequency of analysis; current data ‘pipelines’)
- Questions/Concerns/Personal Training Goals

Benefits of Python for Data Analysis

- All-in-one shop
  - Replace use of Excel
  - Available Packages
  - Improve documentation and reproducibility of individual work
- Programming Functionality
  - Automate
  - Scale
  - Customize and standardize ‘tools’ used

Expanding python

Are any of the following Packages commonly used?

- Data manipulation: pandas, Numpy, scipy
- Machine Learning: scikit-learn, nltk
- Databases: psycopq2, sqlalchemy, sqllite
- Visualizations: matplotlib, plotly, bokeh, seaborn
- API calls / web scraping: requests, BeautifulSoup, Scrapy
- Web development: Django, Flask, Twisted, Scapy
- Desktop App: pyQt, Tkinter

Taking advantage of programming functionality

```
def download_weather_month(year, month):
    if month == 1:
        year += 1
    url = url_template.format(year=year,
month=month)
    weather_data = pd.read_csv(url, skiprows=15,
index_col='Date/Time', parse_dates=True,
header=True)
    weather_data = weather_data.dropna(axis=1)
    weather_data.columns = [col.replace('\xb0', '')
for col in weather_data.columns]
    weather_data = weather_data.drop(['Year',
'Day', 'Month', 'Time', 'Data Quality'], axis=1)
    return weather_data
```

```
def fix_zip_codes(zips):
    # Truncate everything to length 5
    zips = zips.str.slice(0, 5)

    # Set 00000 zip codes to nan
    zero_zips = zips == '00000'
    zips[zero_zips] = np.nan

    return zips
```

Today’s Workshop Goals

Work through a brief overview of the fundamentals of Python in order to:

- obtain a better understanding of existing tools
- establish baseline
- check out my training style and materials

The Fundamentals

- **Syntax and Operators**
- **Simple Data Types and Variables**
- Data Structures
- Control Flow
- (other) Statements

What are examples of data types in Python?

What are examples of data structures in Python?

What are examples of control-flow structures in Python?

jupyter Python_Fundamentals_Overview

User input

Sometimes you want to create programs that require input from a user. There’s a function for that.

The **input()** function:

- by default, will convert all the information it receives into a string.
- has an optional parameter, commonly known as prompt, which is a string that will be printed on the screen whenever the function is called.
- When called, the program flow stops until the user enters the input via the command line. To actually enter the data, the user needs to press the ENTER key after inputting their string.

jupyter Python_Fundamentals_Overview

Exercise 1

jupyter Python_Fundamentals_Overview

Exercise 1

One solution to Exercise 1

Create a program that converts the number of seconds a user wants to convert to hours, minutes, seconds.

```
str_seconds = input("Please enter the number of seconds you wish to convert")
total_secs = int(str_seconds)

hours = total_secs // 3600
secs_still_remaining = total_secs % 3600
minutes =  secs_still_remaining // 60
secs_finally_remaining = secs_still_remaining  % 60

print("Hrs=", hours, "mins=", minutes, "secs=", secs_finally_remaining)
```

The Fundamentals

- Syntax and Operators
- Simple Data Types and Variables
- **Data Structures**
- **Control Flow**
- (other) Statements

Review

Data Structures (Containers)

lists	dictionaries
[ ]	{ k : v }

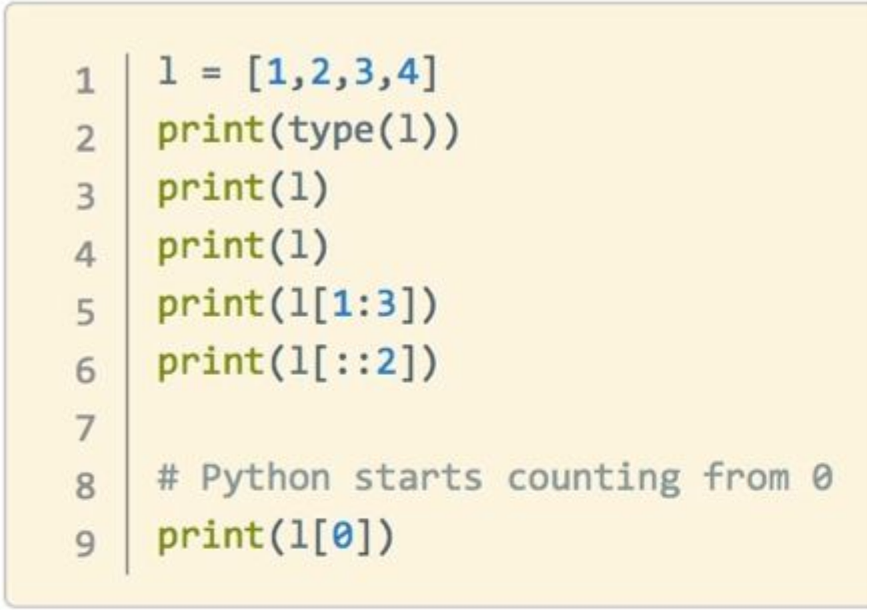
Control Flow

if statement	for loop	functions

Data Structures

Lists

- A **collection** of objects
  - Mixed types are okay
- Defined with **square brackets** [ ]
- They can be modified
  - my_list**.append()**
  - my_list**.remove()**
- **Slicing**
  - Access elements
  - my_list**[** start **:** end **:** step **]**



```python
l = [1,2,3,4]
print(type(l))
print(l)
print(l)
print(l[1:3])
print(l[::2])

# Python starts counting from 0
print(l[0])
```

jupyter Python_Fundamentals_Overview

Data Structures: Lists

Recap: Syntax, Functions, and Methods for Lists

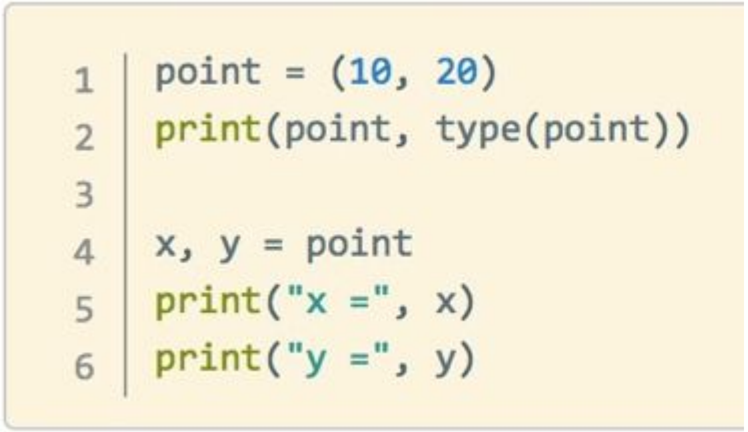
Links to documentation (or relevant forums):

- List syntax
  - Mutability in Python
- list()
- Indexing lists
- Nested list indexing
- Replacing item in list
- .append()
- .extend()
  - Relevant forum
- .insert()
- del
  - Docs
- .pop()
- .remove()
- .sort()
- sorted()
- .reverse()
- len()
- .count(X)

Data Structures

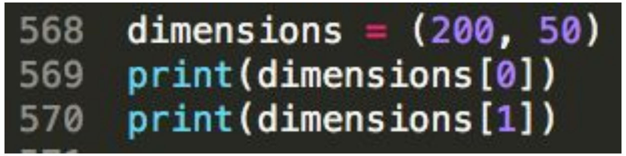
Tuples

- very similar to lists, but:
  - They are defined with parentheses ( ) instead of square brackets
  - They cannot be changed
    - No append() method
    - No remove() method
- Slicing works the same way



```python
point = (10, 20)
print(point, type(point))

x, y = point
print("x =", x)
print("y =", y)
```

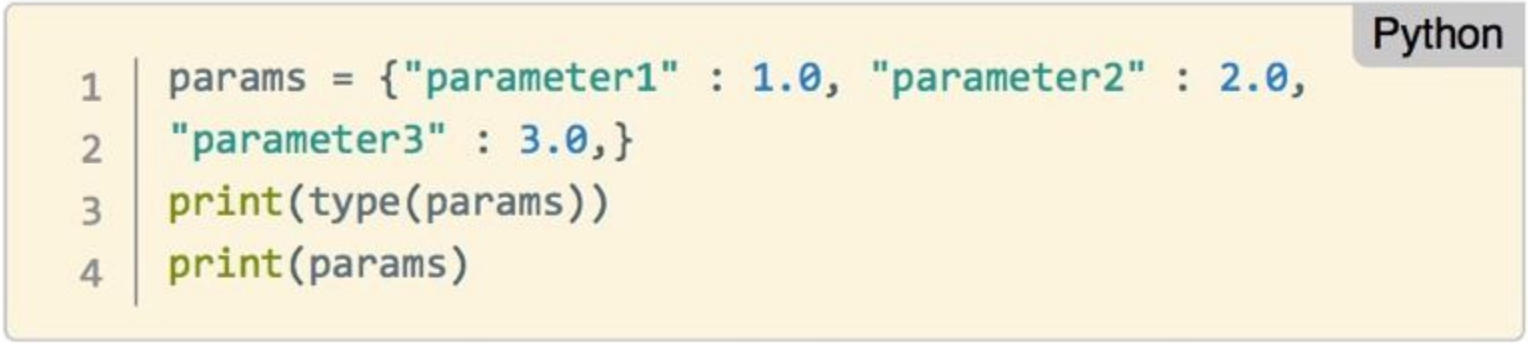


```python
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

Data Structures

Dictionaries

- Collections of **key/value pairs**
- Defined by curly brackets **{ }**
- Slicing uses **keys**
- **Order** is not preserved
    - **(well, for 3.6+ it is:** https://stackoverflow.com/questions/39980323/are-dictionaries-ordered-in-python-3-6**)**



```python
params = {"parameter1" : 1.0, "parameter2" : 2.0,
"parameter3" : 3.0,}
print(type(params))
print(params)
```

Review

Data Structures (Containers)

lists	dictionaries
[ ]	{ k : v }

Control Flow

if statement	for loop	functions

jupyter Python_Fundamentals_Overview

Data Structures: Dictionaries

Control Flow

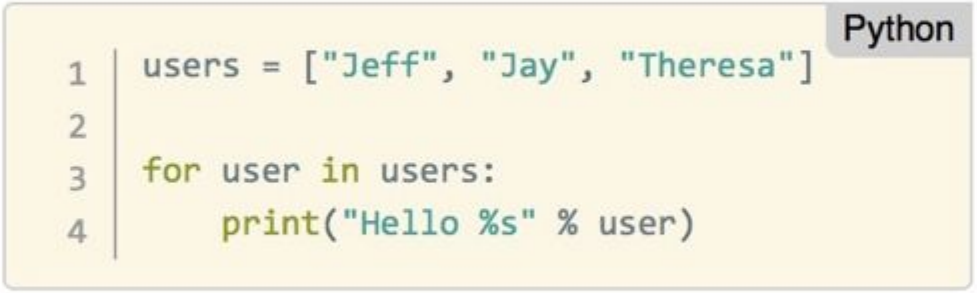
A block of programming that analyses variables and chooses a direction in which to go based on given parameters is a **control structure.**

- The term **control flow** details the direction the program takes (how program logic “flows”). It determines how a computer will respond when given certain conditions and parameters.

Control Structures

for loop

- Repeat operations
- Loop variable takes each value from list in turn
- Indentation controls end of loop
- Watch out for infinite loops!
  - Interrupt or restart kernel when this happens



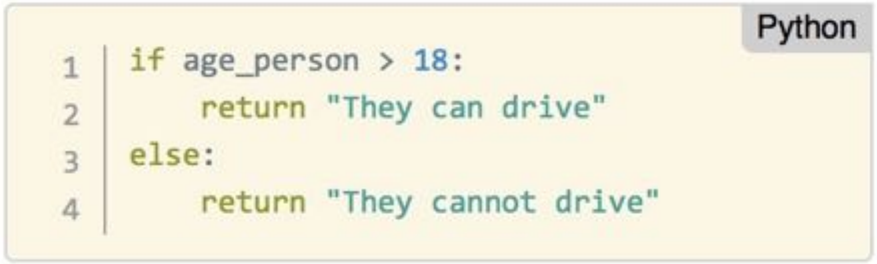
```python
users = ["Jeff", "Jay", "Theresa"]

for user in users:
    print("Hello %s" % user)
```

Control Structures

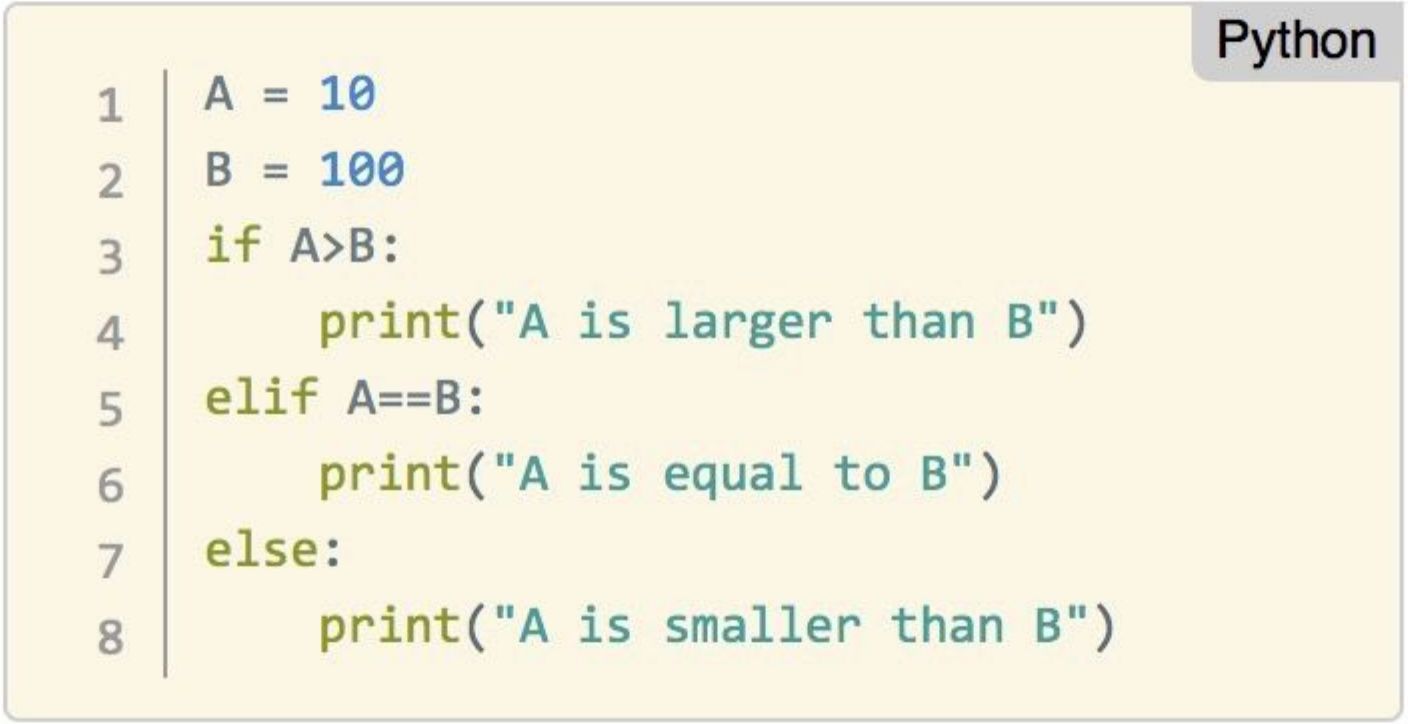
If / elif / else

- Check conditions with boolean operators (i.e. <, >, ==)
- Execute a single code block depending on result
- If True: code runs
- Can be:
  - if alone
  - if/else
  - if/elif(s)/else
- Indentation controls end of **if** block
- Data controls which code runs



```python
if age_person > 18:
    return "They can drive"
else:
    return "They cannot drive"
```

Control Structures - if / elif / else


Python

```python
A = 10
B = 100
if A>B:
    print("A is larger than B")
elif A==B:
    print("A is equal to B")
else:
    print("A is smaller than B")
```

Dictionaries

- A dictionary is a collection of **key-value pairs**.
- A key-value pair is a set of values **associated with each other**.
- A dictionary is **accessed by key** (not position)
- A **key is unique** and *must* be immutable.

Dictionaries

Below is a phone directory, which is a great example of a dictionary.

Why is that?

```
#------------------------------#
# Luna            | 444 - 4444
# Tee             | 123 - 4567
# Ada Lovelace    | 101 - 0101
#------------------------------#
```

jupyter DeeperDive_Dictionaries_Functions

Nested Dictionaries

Here is an example of a nested dictionary:

```
nested_example = {'info': {42: 1, type(''): 2}, 'spam': [1,2,3,'four']}
```

If we wanted to access the value associated with the key 42, you would use the syntax below:

```
print(nested_example['info'][42]) # fetches 1
```

Nested Dictionaries

- You can nest a dictionary inside another dictionary.
- For example, if you have several users for a website, each with a unique username, you can use the usernames as the keys in a dictionary.
- You can then store information about each user by using a dictionary as the value associated with their username.

Control Structures: Functions

Let's Consider a Repetitive Program...

Consider a program that prints a $5 shipping charge for products on a website:

```python
print("You've purchased a Hanging Planter.")
print("Thank you for your order. There will be a $5.00 shipping charge for this order.")

# 10 minutes later...
print("You've purchased a Shell Mirror.")
print("Thank you for your order. There will be a $5.00 shipping charge for this order.")

# 5 minutes later...
print("You've purchased a Modern Shag Rug.")
print("Thank you for your order. There will be a $5.00 shipping charge for this order.")
```

What if there are 1,000 orders?

Seeing Functions in Action

So we *define* the function, then we can call the function by pairing its name with the parenthesis: `print_order()`.

```
def print_order():
 print("Thank you for your order. There will be a $5.00 shipping charge for this order.")

print("You've purchased a Hanging Planter.")
print_order()

print("You've purchased a Shell Mirror.")
print_order()

print("You've purchased a Modern Shag Rug.")
print_order()
```

Functions

We can write a **function** to print the order.

A function is simple — it's a reusable piece of code. We only define it once. Later, we can use its name as a shortcut to run that whole chunk of code.

- Functions are defined using the def syntax.
  - `def` stands for "define."
- In this case, we're *defining* a function named `function_name()`.

```
def function_name():
    # What you want the function to do.

# Call the function by name to run it:
function_name()
```

Pro tip: Don't forget the `()`, and be sure to indent!

Appendix 1

Python Keywords

Each of the following keywords has a specific meaning, and you'll see an error if you try to use them as a variable name.

`False`	`class`	`finally`	`is`	`return`
`None`	`continue`	`for`	`lambda`	`try`
`True`	`def`	`from`	`nonlocal`	`while`
`and`	`del`	`global`	`not`	`with`
`as`	`elif`	`if`	`or`	`yield`
`assert`	`else`	`import`	`pass`	
`break`	`except`	`in`	`raise`	

Python Built-in Functions

You won't get an error if you use one of the following readily available built-in functions as a variable name, but you'll override the behavior of that function:

`abs()`	`divmod()`	`input()`	`open()`	`staticmethod()`
`all()`	`enumerate()`	`int()`	`ord()`	`str()`
`any()`	`eval()`	`isinstance()`	`pow()`	`sum()`
`basestring()`	`execfile()`	`issubclass()`	`print()`	`super()`
`bin()`	`file()`	`iter()`	`property()`	`tuple()`
`bool()`	`filter()`	`len()`	`range()`	`type()`
`bytearray()`	`float()`	`list()`	`raw_input()`	`unichr()`
`callable()`	`format()`	`locals()`	`reduce()`	`unicode()`
`chr()`	`frozenset()`	`long()`	`reload()`	`vars()`
`classmethod()`	`getattr()`	`map()`	`repr()`	`xrange()`
`cmp()`	`globals()`	`max()`	`reversed()`	`zip()`
`compile()`	`hasattr()`	`memoryview()`	`round()`	`__import__()`
`complex()`	`hash()`	`min()`	`set()`	`apply()`
`delattr()`	`help()`	`next()`	`setattr()`	`buffer()`
`dict()`	`hex()`	`object()`	`slice()`	`coerce()`
`dir()`	`id()`	`oct()`	`sorted()`	`intern()`