Python for Data Science
Day 1

By Craig Sakuma

Introductions

Craig Sakuma

- Founder, QuantSprout
- Data Science Instructor, General Assembly
- MBA from Wharton
- B.Eng from Northwestern University

Fun Fact

Developed a novelty BBQ product that was featured in USA Today

Class Introductions

- Name
- What’s your job?
- How do you plan to apply skills from the bootcamp?
- Fun Fact

Course Outline

Day 1: Python Fundamentals

- Introduction to Python
- If Statements
- Lists, tuples, and dictionaries
- For Loops
- Importing Packages
- Intro to Pandas Package

Day 2: Data Manipulation and Machine Learning

- Importing and Exporting Data
- Exploring and Summarizing Data
- Overview of Machine Learning
- Random Forest Algorithm

Objectives for Class

- Get strong foundation of Python and Data Science
- Immediately use skills at work
- Remove barriers/frustration
- Develop skills to be self-sufficient after class
  - Learn and explore
  - Troubleshoot problems

HAVE FUN!

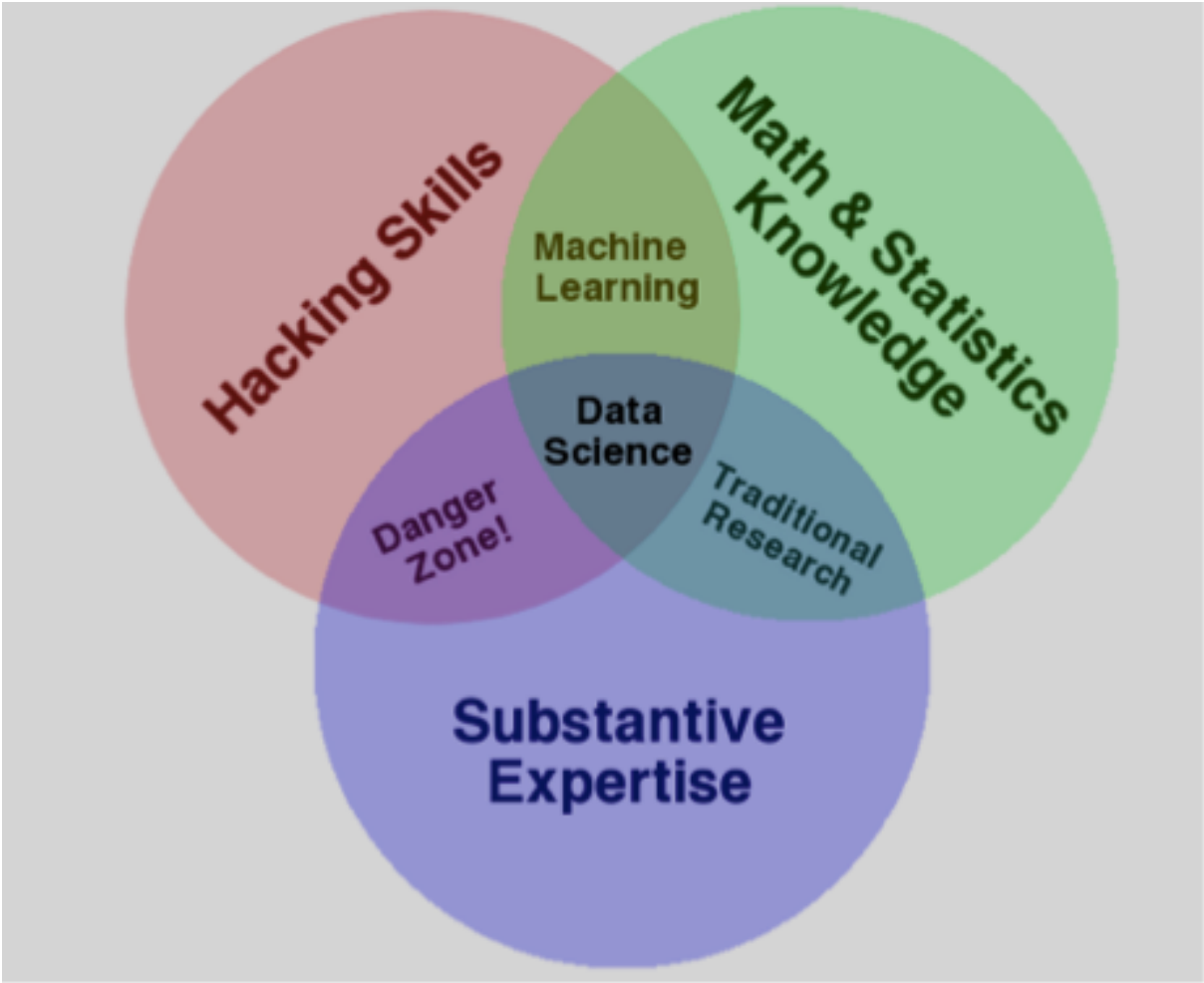
Course Structure

- Lectures on topics
  - Interaction is good
  - Feel free to ask questions
  - If there's not enough time to cover questions, we'll put it in a parking lot for after class
- Hands on exercises
  - Pair programming
  - Mix up partners

Schedule - Day 1

Time	Topic
10:00 – 10:30	Overview of Data Science
10:30 – 12:00	Introduction to Python
12:00 – 1:00	Lunch
1:00 – 2:00	If Statements
2:00 - 3:00	Lists, Tuples and Dictionaries
3:00 – 3:15	Break
3:15 – 4:15	For Loops
4:15 – 5:00	Importing Packages

What is Data Science?



Data Munging

FOR BIG-DATA SCIENTISTS, ‘JANITOR WORK’ IS KEY HURDLE TO INSIGHTS

From NYTimes on August 18, 2014:

“Data wrangling is a huge — and surprisingly so — part of the job,” said Monica Rogati, vice president for data science at Jawbone, whose sensor-filled wristband and software track activity, sleep and food consumption, and suggest dietary and health tips based on the numbers. “It’s something that is not appreciated by data civilians. At times, it feels like everything we do.”

Data Science is OSEMN (Awesome)

Obtain Data

Scrub Data

Explore

Model Algorithms

i**N**terpret Results

80%

20%

Majority of time is spent data munging

Why Python?

- Readability
- Dynamic typing
- Supports multiple programming paradigms
  - Object oriented
  - Functional
  - Procedural

Libraries of Tools for Data Analysis

What is Anaconda?

- Distribution of Python and commonly used libraries of tools
- Easier than individually installing many libraries
- Ensures the versions of each library are compatible with each other

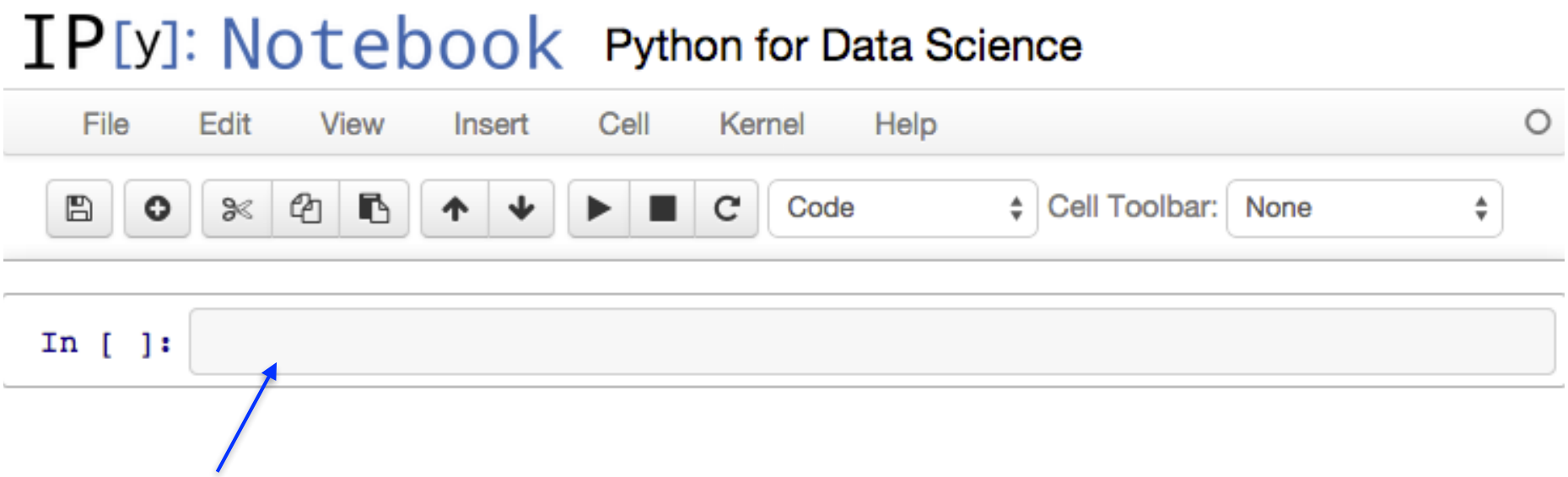
How to Interact with Python

There are several ways to interact with python

- Python Command Line
- Operating System Command Line
- iPython
- iPython Notebook

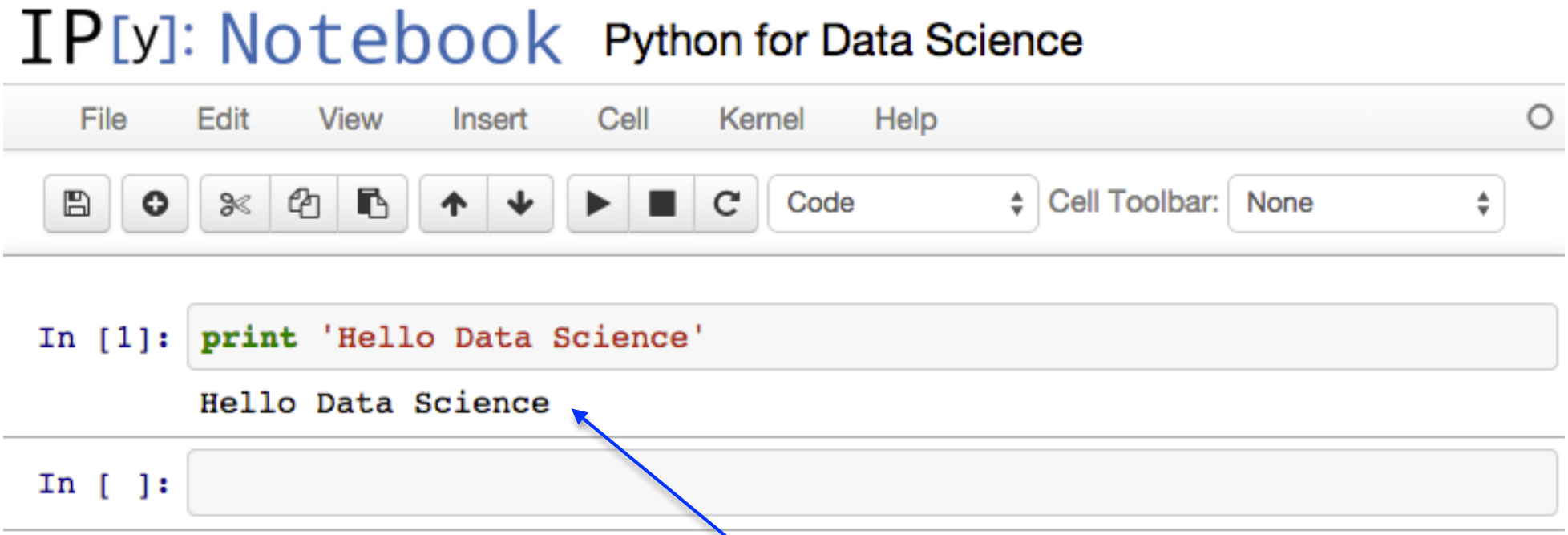
We'll be using iPython Notebooks

iPython Notebook Basics



Enter code here

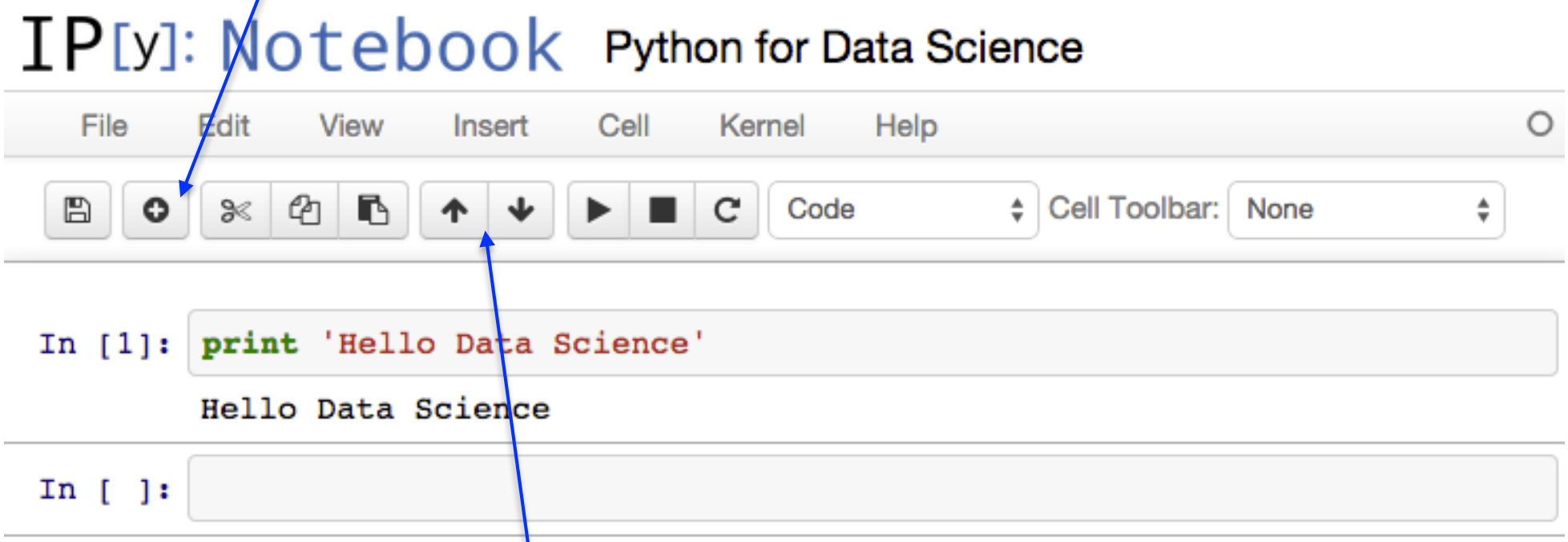
iPython Notebook Basics



Shift + Enter runs code and returns results

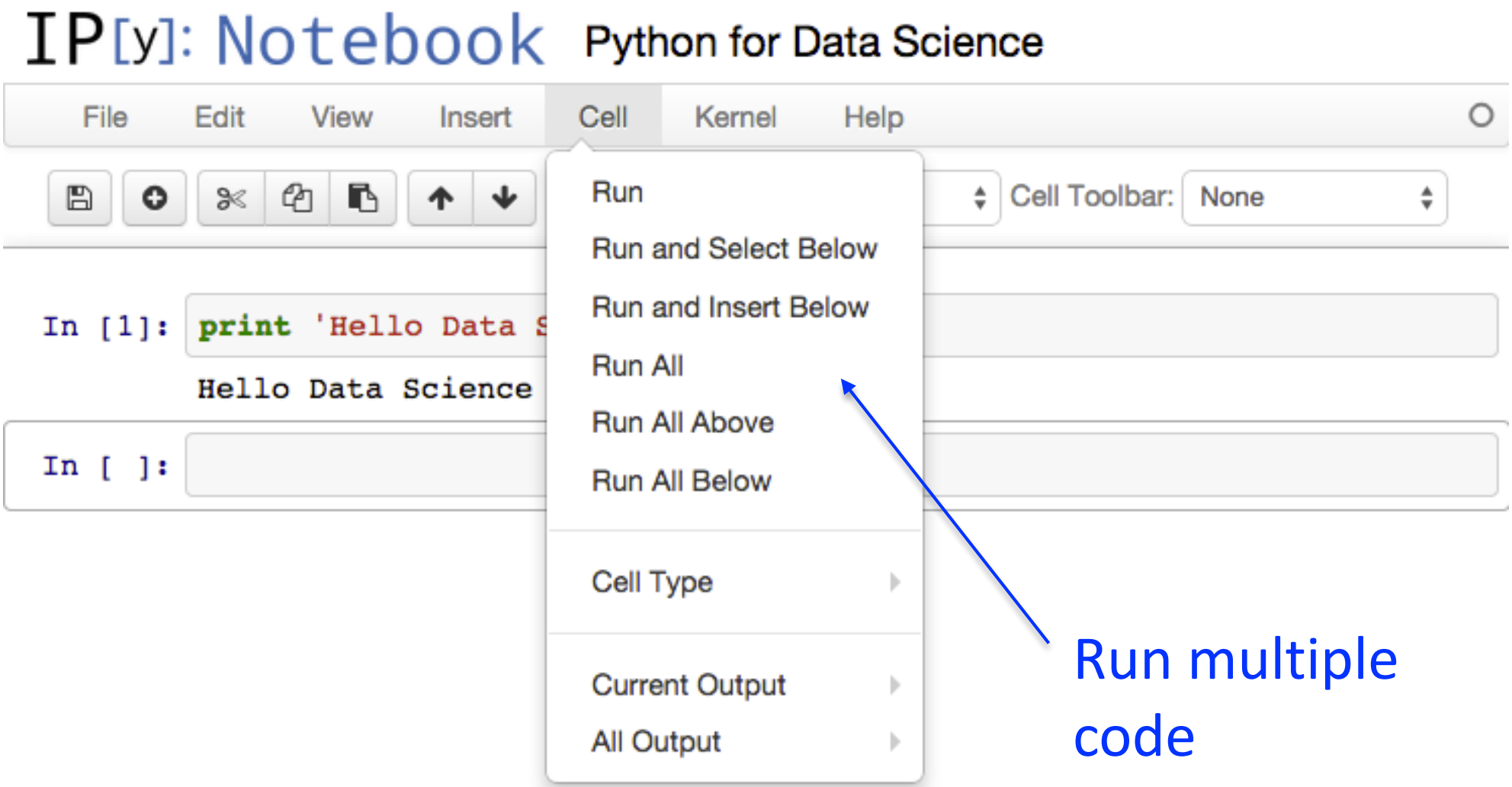
iPython Notebook Basics

Add more code blocks



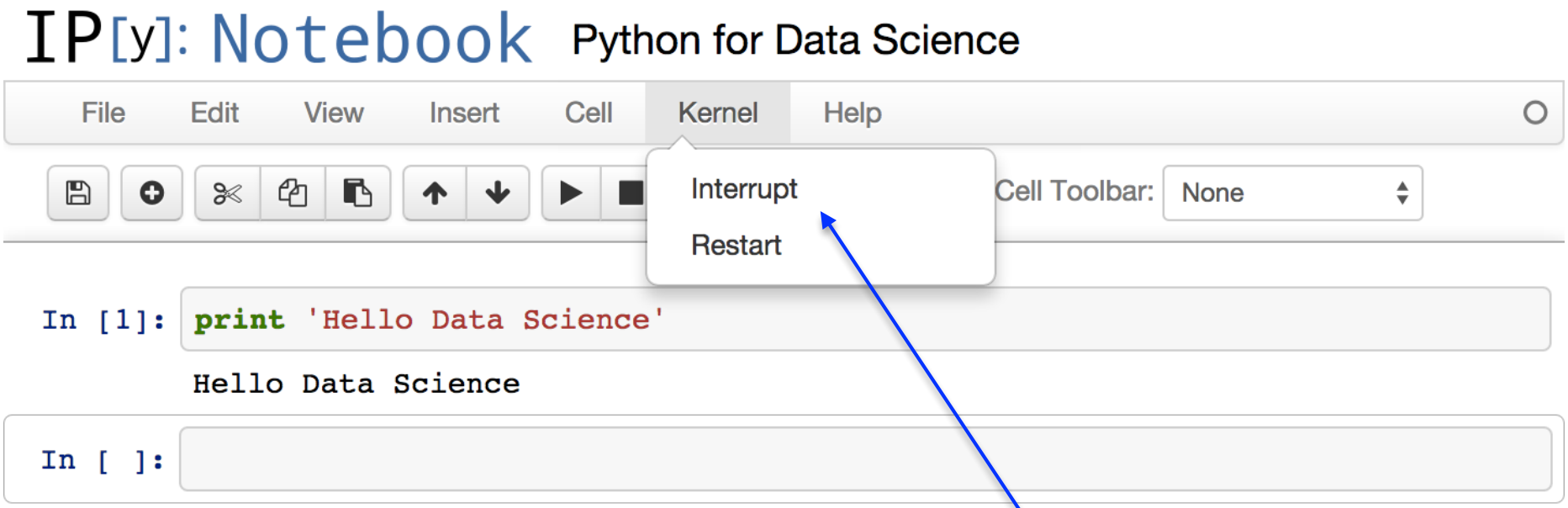
Re-order code blocks

iPython Notebook Basics



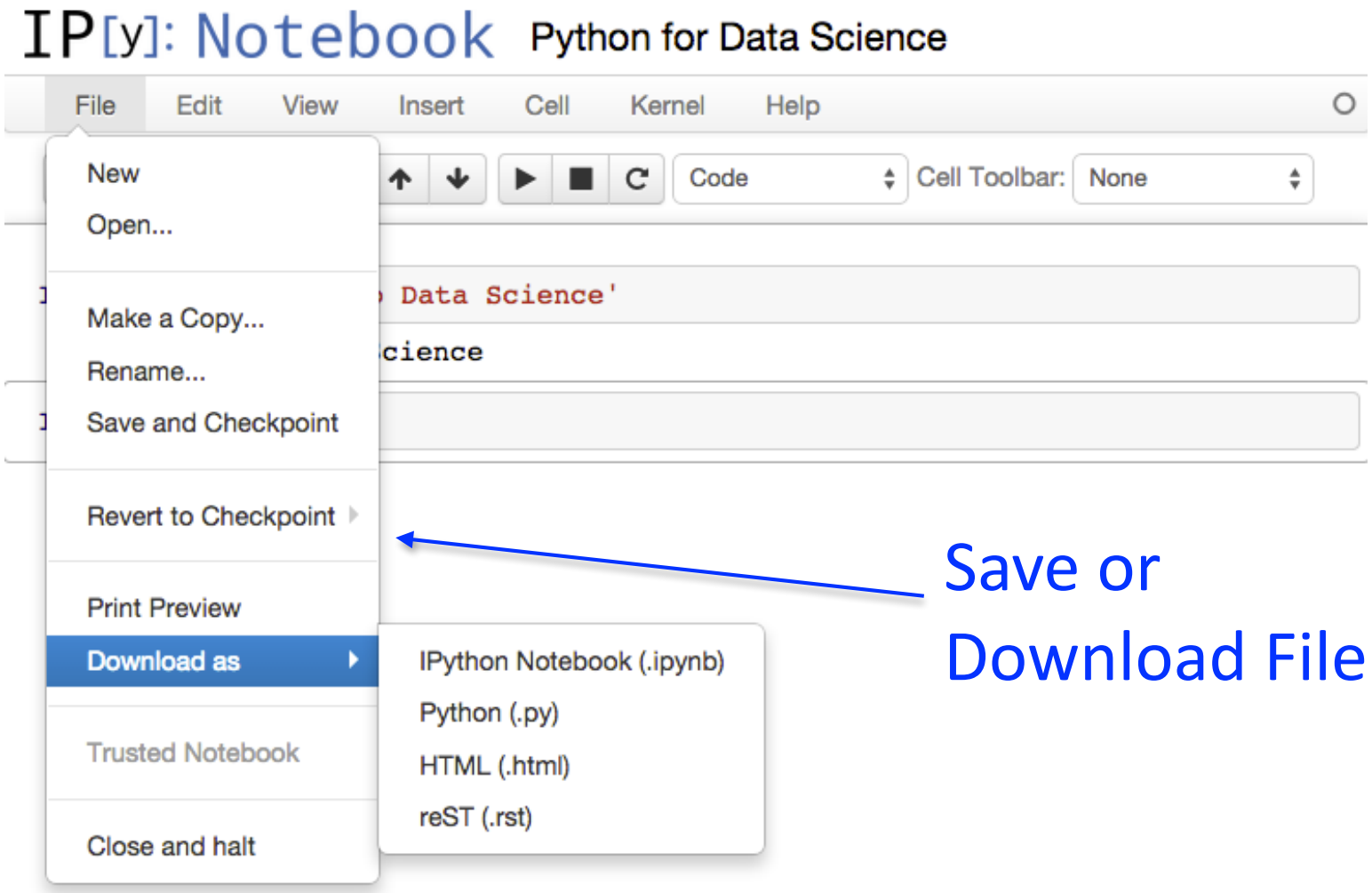
Run multiple code blocks

iPython Notebook Basics



Restart Notebook to Clear Memory

iPython Notebook Basics



Save or Download File

Set Up for Coding

- Create a folder on your Desktop Called “data_science”
- Copy the files from my emails into the folder
- Open your terminal/command prompt and launch ipython notebook
- Navigate to the data_science folder and launch the Python Fundamentals.ipynb file

Write Your First Python Code

Type in the first code block:

```
print “Hello Data Science”
```

Press Shift + Enter

Data Types

- **Numeric Types**
  - Integer (whole numbers)
  - Float (includes decimals)
  - Boolean (True/False)

- **Strings**
  - Text
  - Must be in single or double quotes

Python has function to return data type:

type(<value>)

Try Data Types

```
type(1)
```

```
type(2.5)
```

```
type(True)
```

```
type('string')
```

String Insertion Syntax

Text in quotes

Placeholder for insertion

Command for string insertion

“Bunch of text {}”.format(<value>)

Value in parentheses

Try Basic String Insertion

```
“My name is {}”.format(‘Craig’)
```

```
name = “Waldo”
“Where in the world is {}”.format(name)
```

Multiple Insertions

Multiple insertions require values in the brackets

```
place = “LA”
“{0} is in {1}”.format(name, place)
```

What happens when you change the order of the variables?

Using Names in Insertions

Names can be used in brackets

Easier to read but more verbose

```
“{nm} is in {pl}”.format(nm=name, pl=place)
```

```
“I love {pl}.  I love {pl}!".format(pl=place)
```

Basic Math

Some operators are pretty obvious

```
5 + 5
```

```
3 * 7
```

Basic Math

Some are less intuitive

```
print “Hello ” + “World”
```

```
10 % 4  # modulo
```

```
10 ** 2  # exponent
```

```
1E3 + 1E-3  # exponent base 10
```

Variables

- Variables are objects that hold values
- Name variable using letters, numbers and underscore
- Special characters can’t be used for naming variables (e.g., [,*,@)
- Python commands can’t also be used as variable names
- Assign values to variables using single =
- You can re-assign values to variables

Assign Values to Variables

Create a few variables

```
x = 10
y = 5
z = 4
```

Try math with variables

```
x - y
```

```
x * y
```

Data Types in Math

Try dividing two integers

```
x / z
```

Now try using one float

```
x / 4.0
```

Functions

- Reusable snippets of code
- Define the function once
- Call the function to execute your code as many times as you like
- Can receive inputs and return results

Function Syntax

start with define

name you assign to function

Parentheses

```
def <function> ():
    <code line 1>
    <code line 2>
```

Colon

4 space indent

Code to Execute (can be multiple lines if also indented)

Create a simple function

Write a function

```
def simplestFunction():
    print “I made a function”
```

Call the function

```
simplestFunction()
```

Function with Input

Write a function that requires an input

```
def square(x):
    print x ** 2
```

Call the function

```
square(5)
```

Functions Can Return Values

Write a function that returns a result

```
def cube(x):
    return x ** 3
```

Call the function

```
y = cube(5)
print y
```

Line Continuation

- Sometimes code gets too long to write on one line
- Python automatically recognizes line continuation in specific cases like commas
- Backslashes (\ ) can be used to continue line of code

Line Continuation

Line continuation with commas

```
numbers = [1, 2, 3,
           4, 5, 6,
           7, 8, 9]
```

Backslashes can also be used for continuation

```
long_string = 'This is a really, really, really ' \
              + 'long sentence'
```

Instructions for Exercises

- Pair programming
  - Using only one computer
  - Take turns typing
  - Collaborate on solutions
- Save Examples for Future Reference
  - Add notes using # Comments
- Error Tracking
  - Create a text file to keep notes on your errors
- Trouble-shooting References
  - Online documentation
  - Stackoverflow / Google

Exercise

1. Create a function that converts Celsius to Fahrenheit. Results should be accurate to at least one decimal point.
2. Update your function to return a sentence (string type) with the Celsius and Fahrenheit values inserted into the string.

If Statements

- Used to execute commands when defined conditions are met
- Contains a conditional statement that has a True/False value
- If statement is True then a series of commands will be executed
- If the statement is False then commands are skipped

If Statements Syntax

Must be lower case

Conditional statement

```
if <condition>:
    <code 1>
    <code 2>
```

Colon

4 space indent

Code to Execute (can be multiple lines if also indented)

Conditional Statements

a == b	***Equal***
a != b	***Not Equal***
a > b	***Greater Than***
a >= b	***Greater Than or Equal***
a <= b	***Less Than or Equal***

Multiple Conditions

True and True ***= True***

True & False ***= False***

and, & are interchangeable

True or False ***= True***

False | False ***= False***

or, | are interchangeable

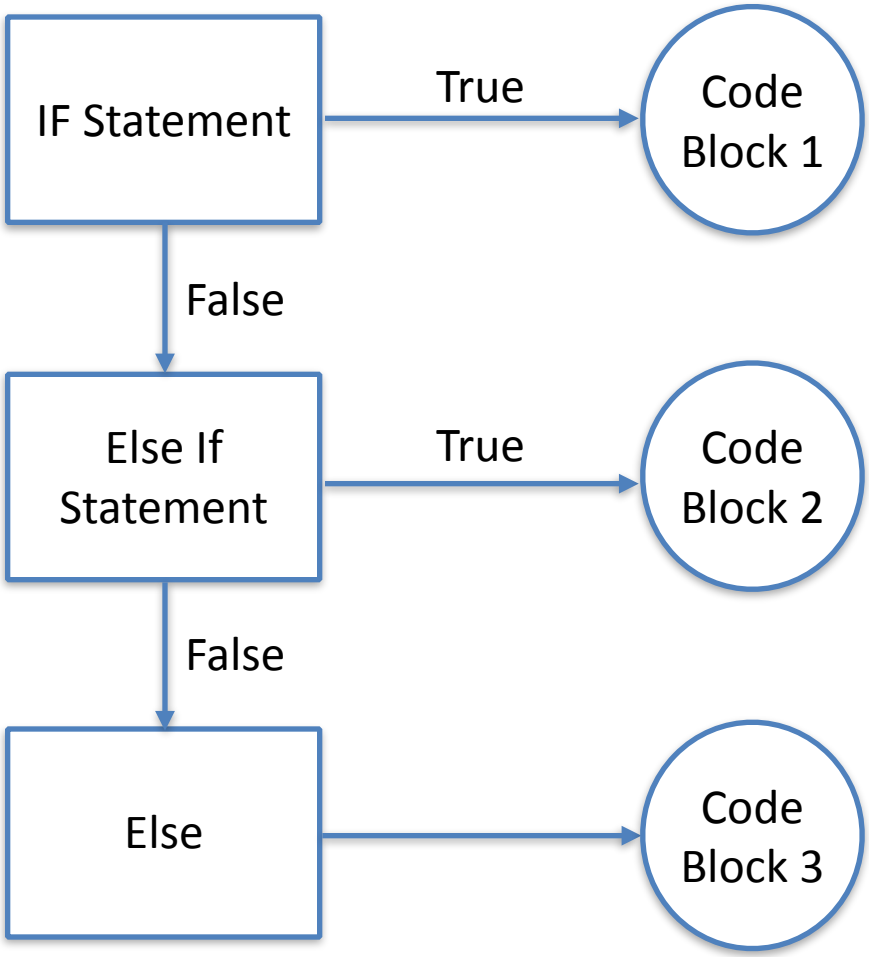
If Statement

Write a simple if statement

```
x = 3
if x > 0:
    print x
```

Else and Else If

Allow additional conditions and actions



Executes First True Statement

Else is catch all and must be at the end

Else If Statement

If statement with Else If

```
x = 3
if x > 0:
    print “{} is a positive number”.format(x)
elif x < 0:
    print “{} is a negative number”.format(x)
else:
    print “x equals 0”
```

Exercise

1. Create a function that checks the type of an input and returns a message stating whether it is numeric or not
2. Update your temperature function from the Python Fundamentals exercise to return an error message if a string is entered instead of a number

Lists, Tuples and Dictionaries

- Python has built-in objects that can hold multiple values
- Can be assigned to variables
- Has built-in methods
- Methods are functions for object

Lists

- Lists are ordered data containers
- Lists are defined with square brackets [ ]
- They can contain any type of objects
  - Mix of data types (e.g., integer, string, float)
  - Lists can even contain other lists
- List are mutable (you can edit them)
- Uses index to reference items in lists
- Lists can be empty

List Basics

Use brackets to define list

```
x = [1, ‘b’, True]
```

Use index position to reference items

```
print x[2]
```

Reassign values in a list

```
x[1] = ‘a’
```

Indexing Lists

Create list of lists

```
a = [[1,2,3], 4, 5]
```

Use multiple indexes for lists within lists

```
print a[0][1]
```

Index from the end of the list

```
print a[-1]
```

Appending and Indexing

Append an item to a list

```
a.append('one more item')
```

Reference multiple items in a list

```
print a[2:4]
```

Open ended indexes go to the ends of lists

```
print a[:3]
```

Tuples

- Tuples are similar to lists
- Tuples are defined using parentheses ( )
- Only difference is that tuples are immutable (you can’t change them)
- Tuples with single value must have a comma (1,)

Tuple Basics

Use parentheses to define tuple

```
y = (1, ‘a’, 2.5)
```

Use index position to reference items

```
print y[0]
```

Try reassigning values in a tuple

```
y[0] = 2
```

Dictionaries

- Dictionaries are collections of key-value pairs
- Dictionaries are indicated by curly braces { }
- Values are looked up by key
- Dictionaries are unordered

Dictionary Basics

Create a dictionary

```
info = {'name': 'Bob', 'age': 54, 'kids': ['Henry', ‘Phil']}
```

Use key to reference a value

```
print info[‘name’]
```

Change the value for a key-pair

```
info[‘age’] = 55
```

Dictionary Methods

View all keys

```
info.keys()
```

View all values

```
info.values()
```

Check if a key exists

```
info.has_key('age')
```

For Loops

- Iterates through multiple values
- Commonly used to process values in a list
- Loop of code is executed for each item

For Loop Syntax

Starts with for

Variable in List

Name of list

```
for <variable> in <list>:
    <code 1>
    <code 2>
```

colon

4 space indent

Block of code executed for each loop

Basic For Loops

Create basic for loop

```
for x in [1,2,3]:
    print x
```

Create for loop with multiple values

```
for x, y in [[1, 4], [2, 5], [3, 6]]:
    print x * y
```

For Loops with Empty List

Capture the all the results of a for loop

```
results = []
for x in [1,2,3]:
    squared = x **2
    results.append(squared)

print results
```

Exercise

1. Create a function that receives a list of numbers as an input, adds 1 to each number and returns the results as a list
2. Update your temperature conversion function from the Python Fundamentals exercise to accept a list of Celsius temperatures and return a list of Fahrenheit temperatures

Bonus:

Add error handling to your temperature conversion function.

Python Packages

- Data analytics packages are what make python so powerful
- Packages are just files of python code
- Importing packages allow you to use the functions from these files
- Most packages have online documentation and code examples

Common Packages for Data Science

Package	Usage
numpy	Scientific computing
pandas	Data slicing and manipulation
datetime	Manage date and time formats
matplotlib	Creating charts and graphs
scikit-learn	Machine learning
statsmodels	Statistics

Importing Packages

- Plain import statement:
  import <package name>

- Use a nickname:
  import <package name> as <nickname>

- Import a subset of the package:
  from <package> import <function>

- Avoid this technique, because it can create name-space conflicts
  from <package> import *

Import Packages

Let's import a package with a nickname

```
import pandas as pd
```

Use ipython magic to see function options. Type pd. and press tab. Highlight DataFrame and press enter. Hit shift-tab

```
pd.DataFrame
```

Intro to Pandas

- Primary objects in Pandas are DataFrames
- DataFrames are like tables
  - Contain rows and columns of data
  - Columns have names
  - Rows have index values

Indexing in Pandas

There are a couple ways to index DataFrames

- Column names and row index
- Relative position (similar to Excel)

Indexing can be a large source of confusion and frustration

However, we'll go over some examples that will help avoid a lot of that pain

Create DataFrame Syntax

Pandas DataFrame function

Data in Curly Brackets

```
pd.DataFrame({<column1>: <list>,
              <column2>: <list>}
```

Column Names

Colon Between Name and Data for each column

List of Values

Create a DataFrame

Build simple DataFrame

```
import pandas as pd
df = pd.DataFrame({'name':['Jen', 'Bob', 'Tim'],
                   'age':[20, 30, 40],
                   'pet':['dog', 'cat', 'bird']})
print df
```

View the column names and index values

```
print df.columns
print df.index
```

Indexing DataFrames

Select a column by name in 2 different ways

```
print df['name']
print df.name
```

Select multiple columns

```
print df[['name','pet']]
```

Select a row by index

```
print df.ix[0]
```

Pandas Documentation

Let's look up the sort function

- Go to pandas.pydata.org
- Select the documentation for the version you have (0.16.2)
- Use search to find 'sort'

Pandas - Sort Function

pandas.DataFrame.sort

`DataFrame.sort`(*columns=None, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last'*) ¶

Sort DataFrame either by labels (along either axis) or by the values in column(s)

Parameters :

columns : *object*

Column name(s) in frame. Accepts a column name or a list for a nested sort. A tuple will be interpreted as the levels of a multi-index.

ascending : *boolean or list, default True*

Sort ascending vs. descending. Specify list for multiple sort orders

axis : *{0, 1}*

Sort index/rows versus columns

inplace : *boolean, default False*

Sort the DataFrame without creating a new instance

kind : *{'quicksort', 'mergesort', 'heapsort'}, optional*

This option is only applied when sorting on a single column or label.

na_position : *{'first', 'last'} (optional, default='last')*

'first' puts NaNs at the beginning 'last' puts NaNs at the end

Returns :

sorted : *DataFrame*

Row Index Stays with Data

Sort the data by name

```
df.sort('name',inplace=True)
```

View the index after the sort

```
print df
```

The index is a fixed reference that is assigned when you create a DataFrame

Indexing by Relative Position

Panda's has another index method - **.iloc**

- Syntax:

  <DataFrame>.iloc[<row>,<column>]

- Row and column are the relative position of the data cells you want
- To select multiple rows or columns, use a colon to separate the start and end values
- Colon with no value returns all rows or columns

Don’t Confuse ix and iloc

Difference between ix and iloc

```
print df.ix[0]
print df.iloc[0]
```

Use iloc to select all rows of a column

```
print df.iloc[:,2]
```

Use iloc to select the last row

```
print df.iloc[-1:,]
```

Exercises

- Create a DataFrame with data provided
- Sort the DataFrame by sales in descending order.
- What is the name of the first column (by relative position)?
- Which customer is in the last row (by relative position)?