

Lecture 6: Python Libraries

Chirag Sakhuja

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

LBE: Debug decorator

```
def debug(func):  
    def wrapper(*args, **kwargs):  
        print('Arguments: ', args, kwargs, end=' -> ')  
        return func(*args, *kwargs)  
    return wrapper
```

```
@debug  
def foo(x, y):  
    print(x + y)
```

```
foo(1, 2)      # => Arguments: (1, 2) {} -> 3
```

LBE: Car information

```
class Car:
    total_cars = 0
    def __init__(self, make='Toyota', model='Camry'):
        Car.total_cars += 1
        self.make = make
        self.model = model
car = Car()
dream_car = Car('McLaren', 'P1')
print(car.make)           # => Toyota
print(dream_car.make)     # => McLaren
print(Car.total_cars)     # => 2
```

LBE: Complex numbers

Need to **import** math

```
class Complex:
    def __init__(self, real=0, imag=0):
        self.real = real
        self.imag = imag
    def getRho(self):
        return math.hypot(self.real, self.imag)
    def getTheta(self):
        return math.atan2(self.imag, self.real)
c = Complex(1, 1)
print(c.getRho(), c.getTheta()) # => 1.414... 0.785...
```


Class definitions

- Class object: a definition that encapsulates attributes and functions
- Class instance: a "copy" of a class object that has its own state
 - Created by calling a constructor defined in a class object
- Class attributes: variables encapsulated by a class object that are the same for all class instances and can be directly accessed via the class object (similar to `static` in C++)
- Instance attributes: variables encapsulated by a class object that have independent values for each class instance

Polymorphism

```
class Vehicle:
    def __init__(self):
        print('Constructing Vehicle')
    def func(self):
        print('func from Vehicle')
class Car(Vehicle):
    def __init__(self):
        print('Constructing Car')
    def func(self):
        print('func from Car')
c = Car()      # => Constructing Car
c.func()      # => func from Car
```

Comma-delimited class objects from which to inherit



LBE: Pretty printing

```
class Complex:
    def __init__(self, real=0, imag=0):
        self.real = real
        self.imag = imag
    def __str__(self):
        return '({}, {})'.format(self.real, self.imag)
```

```
c = Complex(1, 2)
print(c) # => (1, 2)
```


Basic error checking

```
while True:
    try:
        float(input('Enter the first operand: '))
        break
    except ValueError:
        print('Try again!')
# => Enter the first operand: <= Hi
# => Try again!
# => Enter the first operand: <= 2
```

Good OOP is a lot of work

- Proper object-oriented programming requires following two principles
 - Cohesion: the focus of an individual component
 - Coupling: the relationship between individual components
- High cohesion means everything related to a component is encapsulated within that component, and everything else is encapsulated by other components
- Loose coupling means that components only depend on other components when necessary
- We want high cohesion and loose coupling so changing a component doesn't affect other components or require rethinking the abstraction

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

import keyword

- Similar to `#include` in C/C++
- Local Python files can be `imported`
 - As long as they have a `.py` extension
- Python also has many built-in libraries, some of which we've seen
- Package managers, such as `pip` and `conda`, allow you to `import` more libraries that don't come installed by default

math library

```
import math
```

```
math.floor(math.log(10))    # => 2  
math.hypot(3, 4)           # => 5.0  
math.gamma(4.0)            # => 6.0
```

Command line arguments with `sys` library

main.py

```
import sys
def main():
    if len(sys.argv) <= 1:
        sys.exit(1)
    print(sys.argv)
if __name__ == '__main__':
    main()
```

First argument is name of executable



Normal exit returns code 0



CLI

```
> python3 main.py
> python3 main.py argument
['main.py', 'argument']
```

functools library

```
import functools
```

```
data = [1, 2, 3, 4]
```

```
functools.reduce(lambda x, y: x + y, data) # => 10
```

as keyword for renaming

```
import functools as ft
```



functools can be referred to as ft

```
data = [1, 2, 3, 4]
```


```
ft.reduce(lambda x, y: x + y, data) # => 10
```


Limited import with **from** keyword

```
from functools import reduce
```

```
data = [1, 2, 3, 4]
```

```
reduce(lambda x, y: x + y, data)  # => 10
```



Only import reduce function from the library

Installing more libraries

- Python has a few package managers
 - Most popular is `pip`
 - We're using `conda` for this class because it comes as part of the suite of Anaconda tools
- `conda install package-name` from the Anaconda prompt (Windows) or the terminal (macOS and Linux)

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

Report-style notebook

- Similar interface to Mathematica and Matlab
- Breaks down program into "cells" for organizational purposes
 - A cell's execution is persistent throughout the notebook
 - Cells do not necessarily execute in order, even though that is most often the logical way of using a notebook
- Consist of a mix of Markdown and Python
 - Markdown is a simple typesetting language
- Integrate well with plotting libraries and other graphics
- Useful for sharing results

Creating a notebook

```
conda install jupyter  
jupyter notebook
```



Only first time

Creating a notebook

- Launches a webapp with a file browser
- Create a new notebook with a Python 3 kernel

Demo

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

Installation

```
conda install numpy
```

Demo

See 'numpy (HTML)' on Canvas

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

Installation

```
conda install matplotlib
```

Demo

See 'matplotlib (HTML)' on Canvas

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

Installation

```
conda install pandas
```

Demo

See 'pandas (HTML)' on Canvas

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

Will cover in lab section...

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

click

```
import click
@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name',
              help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT
    times."""
    for x in range(count):
        click.echo('Hello %s!' % name)
if __name__ == '__main__':
    hello()
```

pygame (and pygamelet)



requests

```
>>> import requests
```

```
>>> r = requests.get('https://api.github.com/events')
```

scikit-learn

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics. — Examples

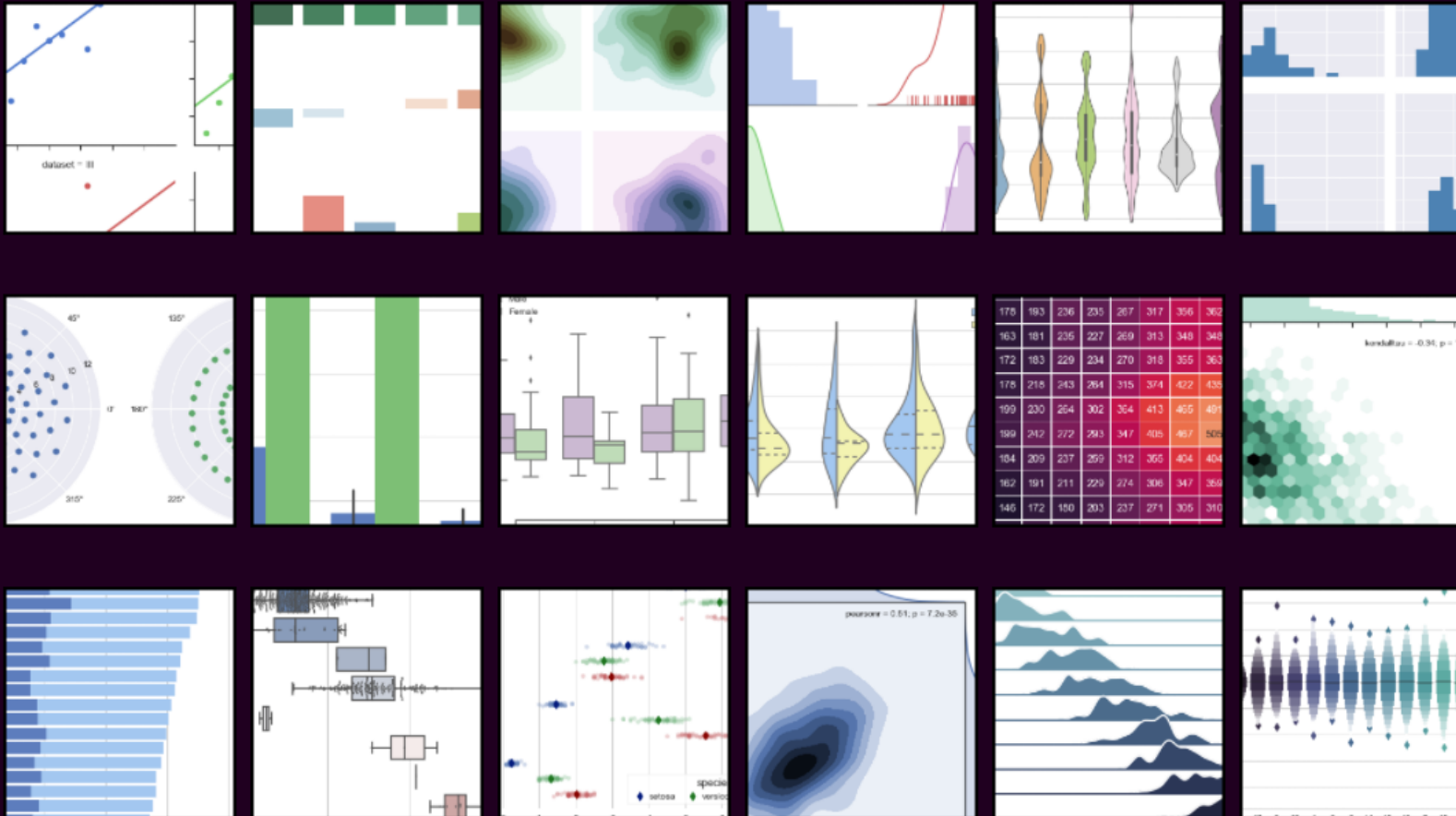
Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples

seaborn



...and many more

Overview

Recap

Using libraries

Jupyter notebooks

numpy

matplotlib

pandas

flask

Honorable mentions

Conclusion

Key insights

- There is *a lot* of support for Python
- Python is a flexible, clean, and portable language, which makes it conducive to doing anything
- Prototyping projects in Python (or using it exclusively if performance isn't very significant) is made easy by a vast set of libraries