Python Workshop

Topics

1. Overview
2. Setting Up Python
3. Data Types
4. Flow Control
5. Exercise
6. Applications

Overview

What is Python?

- High-level
- Dynamic typing
- Strongly typed
- Objected oriented
- Interpreted

Why use Python?

- Easy to pick up
- Extensive documentation
- Lots of packages and frameworks
- Widely used in industry
- Easy to read
  - Indentation dictates scope
  - No curly braces {} or semicolons ;

Setting up the Python Environment

- Python 2.7 should be installed by default on macOS and Windows (Windows Subsystem for Linux)
  - This version is deprecated as of the beginning of 2020. Avoid using it!
- Install the latest (stable) version, Python 3.8.1, through python.org
- Package management and installation through `pip`
- If you don't have the environment installed, you can follow today's code on an online environment such as: https://www.onlinegdb.com/online_python_compiler
- All of our sample code can be found on Github: https://tinyurl.com/CruzPython

Data Types

int and float

- What you would expect: integers and floating point numbers
- `int` has unlimited precision (as big as you want)
- Implicit conversion to `float` when necessary
  - `1 - 1   # Returns 0, an int`
  - `1 + 1.0 # Returns 2.0, a float`
  - `10 / 2  # Returns 5.0, a float`

list

- Mutable sequence of objects, put anything it it
- Dynamically sized (grows as you put more stuff in it)
  - ```
    mylist = ['foo', 'bar', 'baz']
    for i in range(len(mylist)):   # traditional for loop
        print(mylist[i])           # foo bar baz
    ```
    ```

list (cont.)

- `for ele in mylist:  # iterate through list elements`
  `print(ele)  # foo bar baz`
- Cleaner way to iterate through lists (get index and elements)
  - `for index, ele in enumerate(mylist):`
    `print(index, ele)  # 0 foo 1 bar 2 baz`

list (cont.)

- List comprehensions are cool
  - ```
    single = [1, 2, 3]
    doubled = [x * 2 for x in single]  # [2, 4, 6]
    ```
- List splicing is also pretty cool
  - ```
    mylist = ['thank', 'you', 'kanye', 'very', 'cool']
    mylist[3:]   # ['very', 'cool']
    mylist[1:3]  # ['you', 'kanye']
    ```

dict

- An unordered list of **key : value** pairs
- Access values by indexing via keys
  - ```
    mydict = {mykey: 'myvalue'}
    mydict['foo'] = 'bar' # Add a key-value pair
    print(mydict['foo'])  # Access val associated with foo
    ```
- Uses a hashtable internally to handle mappings
    ```

set

- Unordered collection of objects
- Implemented using a hashtable
- O(1) membership test
  - ```
    myset = set()
    myset.add('foo')
    if 'foo' in myset:
        print('foo in set')
    ```
    ```

str

- Strings are immutable (read-only) sequences of characters
- There is no `char` data type, only `str` of length 1
- `mystring = 'foo'`
  - ```
    print(mystring[0]) # prints f
    mystring[0] = 'b'  # Wrong, can't modify strings
    ```
- Encoded as UTF-8 by default
    ```

Flow Control

bool operators

- Similar to other languages (`==, >, <, >=, <=, etc.`)
- None is the NULL of Python
  - ```
    x = None
    if x is None:
        print('x is None')
    ```
- (`and, or, not`) are the python keywords for (`&&, ||, !`)
    ```

Conditionals: if

- the `if` keyword exists in Python:
  - ```
    age = 20
    if age < 21:
        print("You can't buy alcohol")
    ```
    ```

Conditionals: while

- `while` loops also exist
  - ```
    rainy = True
    while rainy:
        print("Don't forget umbrella!")
    ```
    ```

for

- For loops also exist in Python
- Easy-to-implement list traversal
  - ```
    mylist = ['thank', 'you', 'kanye', 'very', 'cool']
    for word in mylist:
        print(word)
    ```
- We can also control the number of times we iterate
  - ```
    for i in range(1,10):
        print(i)          # Prints numbers 1-9
    ```
- If we need to index AND traverse a list at the same time,
  - ```
    for index, ele in enumerate(mylist):
        print(index, ele)
    ```
    ```

Functions

- No need to define data type of arguments
- ```
  def nameAndAge(name, age):
      print(name, age)

  jon_name = ‘Jon Chong’
  jon_age = 21

  myFunc(jon_name, jon_age)
  ```
  ```

Classes

- Python is an object-oriented language
- (Almost) everything is a class
- Classes are easily implemented:
- ```
  class Student:
      ucsc_student = True
      # Constructor
      def __init__(self, user_name, user_age):
          self.name = user_name
          self.age = user_age
  ```
  ```

Classes (cont.)

- Class functions are defined much like regular functions
- Must pass in `self` to all class functions

```
class Student:
    # Constructor
    def __init__(self, user_name, user_age):
        self.name = user_name
        self.age = user_age

    def birthday(self):
        self.age += 1
```

Exercise!

Exercise

Print out the first 10 numbers of the Fibonacci Sequence

- Hints:
  - The Fibonacci Sequence is defined as:
    - The first number is 0. The second number is 1.
    - From then on, the next number is the sum of the previous two numbers
      - ex. Third Fibonacci number = First + Second = 0 + 1 = 1
  - You can either use two `int` variables as trackers, or use a list. Use `.append(number)` to add more elements to the end of a list.
  - Some starter code can be found in our Github repository: https://github.com/david3de/cruzhacks-python

Applications

How is Python used in industry?

- Scripting
- Data science
- Machine learning
- Server-side code
- Test automation
- ...and much more

How you can use Python at CruzHacks

- Backend for a web app using Flask or Django
- Add machine learning using scikit-learn
- Build a desktop app using PyQt
- Natural Language Processing using NLTK

Additional Resources

- Official Python docs: docs.python.org/3/
- CodeAcademy: codecademy.com/catalog/language/python
- Crash Course: grahamwheeler.com/posts/python-crash-course.html

SASE

SOCIETY OF ASIAN SCIENTISTS & ENGINEERS

https://www.facebook.com/UCSCSASE

Good luck at the hackathon!