

Python

ECE 650
Methods & Tools for Software Engineering (MTSE)
Fall 2017

Prof. Arie Gurfinkel



Makeup Class This Saturday (Tomorrow!)

Date: Saturday, September 23, 2017

Location: RCH 307

Time: 10am – 12:20pm

AccessAbility Services Volunteer Notetaker Required

Interested? Complete an online application using your WATIAM:

<https://york.accessiblelearning.com/UWaterloo/>

More information:

Website: <https://uwaterloo.ca/accessability-services/current-students/notetaking-services>

Email: notetaking@uwaterloo.ca

Phone: 519-888-4567, ext. 35082



UNIVERSITY OF
WATERLOO

Assignments

Assignment 0

- let's do it together 😊

Assignment 1

- let's take a look

Python

Created by Guido van Rossum in early 90s

- simple and elegant syntax emphasizing readability
- dynamic type system ("duck" typing)
- automatic memory management
- dynamically interpreted



Python 2.0 released in 2000

Python 3.0 released in 2008

- many new features
- NOT backward compatible to Python 2.0
- both version 2 and 3 are still actively used

We use Python v2.7 in the course



Duck Typing

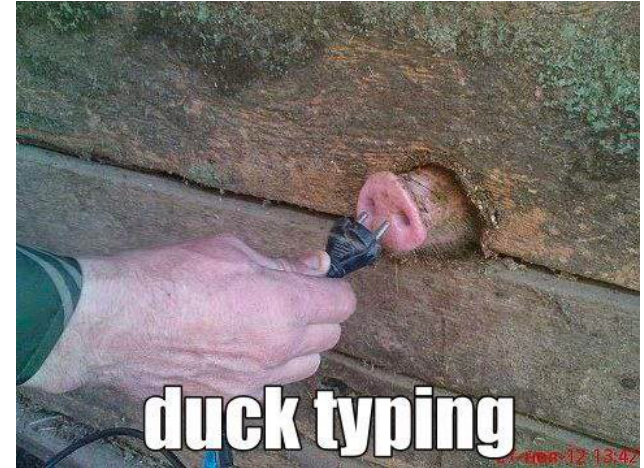
“if it walks like a duck and quacks like a duck, then it must be a duck”

A type of any object / expression is determined dynamically based on what operations (methods / functions) the objects involved support

- if the code works then it is typed correctly

This means that there are very few checks that can be done before the code is executed

- thus, a poorly tested program might contain hidden code paths that do not are not even executable (i.e., do not produce any answer)



<http://stereobooster.github.io/duck-typing>

Many Flavors of Python

CPython (a.k.a. Python)

- the official implementation of Python in C
- a defacto standard of the language



PyPy

- an alternative implementation
- based on RPython framework for developing interpreters for dynamic languages



Jython

- a Java-based implementation
- compiles Python into Java bytecode



Cython

- a C-based implementation
- compiles Python into C for more efficient execution



Don't forget that there is version 2 and version 3 of everything!

IPython

IP[y]: IPython
Interactive Computing

An interactive shell for Python

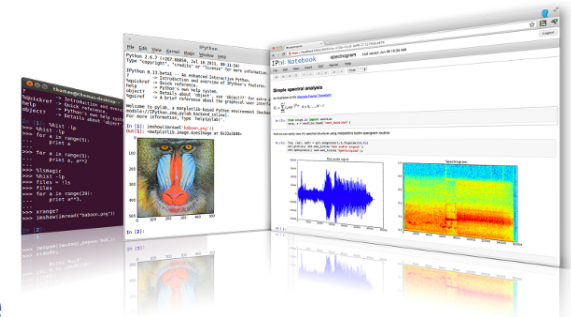
- written in Python

Much more user friendly than the standard Python interpreter

- many helpful features to discover available modules, methods
- easy access to documentation
- good way to learn the language by trying

Part of a bigger ecosystem

- Jupyter, Python Notebooks, graphs, and many more
- <https://ipython.org/>



We will only use the interactive shell

<https://docs.python.org/2/tutorial/index.html>

PYTHON TUTORIAL

Regular Expressions

RegEx – a language to specify and discover patterns in strings

(Basic) Syntax

regex ::=	letter	(<i>exact match</i>)
	(regex)	(<i>grouping</i>)
	regex?	(<i>zero or one</i>)
	regex+	(<i>one or more</i>)
	regex*	(<i>zero or more</i>)
	regex regex	(<i>sequence</i>)
	regex regex	(<i>choice</i>)

letter ::= (*see next slide*)

Regular Expressions (Cont'd)

letter ::=	char	(exact match)
	.	(matches any character)
	[char+]	(any char in the group)
	[^ letter+]	(any char not in a group)

char ::= (a single character)

Python RE library

- <https://docs.python.org/2/library/re.html>
- provides many additional “characters” and extra operators to refine and simplify the matching
- provides API to find matches in strings

Regular Expressions by Example

Single Digit: `[0-9]`

Non-Digit: `[^0-9]`

Non-Space: `[^]`

Natural number: `[0-9]+`

Integer: `[-]?[0-9]+`

Decimal: `[0-9]+(\.[0-9]+)?`

In Python

```
import re
r = re.compile(r'[0-9]+')
v = r.findall('555-4567 ext. 3483')
print v
```

Unit Testing

A *unit test* exercises a unit of functionality to test its behavior

A *unit test framework* provides a standard mechanism for

- specifying a test (setup, execution, expected result, teardown)
- executing a test
- generating test reports

Python includes a Unit Test framework called *unittest*

- <https://docs.python.org/2/library/unittest.html>

It is important to design your code with testing in mind

- e.g., a code that simply reads and writes to standard input and output is harder to test than code that provides a more structured interaction

Anatomy of a Unit Test

include module

A test case is a collection of tests

A method is a test

```
import unittest
```

```
class TestStringMethods(unittest.TestCase):
```

```
    def test_upper(self):  
        self.assertEqual('foo'.upper(), 'FOO')
```

```
    def test_isupper(self):  
        self.assertTrue('FOO'.isupper())  
        self.assertFalse('Foo'.isupper())
```

```
    def test_split(self):  
        s = 'hello world'  
        self.assertEqual(s.split(), ['hello', 'world'])  
        # check that s.split fails when the separator is not a string  
        with self.assertRaises(TypeError):  
            s.split(2)
```

```
if __name__ == '__main__':  
    unittest.main()
```

Calls to
assertXXX()
methods indicate
test results

Entry point for the
test when ran from
command line

Designing for Testing

Factor the program into meaningful units / components

- e.g., parser, command processor, components, data structures, etc.

Each unit should have a well defined specification

- what are legal inputs
- what are legal outputs
- how inputs and outputs are passed around

Avoid monolithic design that reads standard input and writes standard output

Good design requires more work

- additional functionality specifically for testing / debugging purposes
- but ultimately will save time of the overall development

Design for A1

Command Parser

- input: line of text
- output: command or error

Street Database

- a list of streets and their line segments
- interface: add/delete/change/check street

Graph

- a store for edges and vertices

Graph Generator

- input: Street Database
- output: Graph

Graph Printer

- input: a graph
- output: a graph in the output format of A1

coverage.py

A *test coverage* is a metric identifying how much of a program has been executed by a given test (or a set of inputs)

- e.g., $\text{\#statements executed} / \text{\# total statements}$

Statement coverage measures the number of statements executed

Branch coverage, in addition, measures the number of branches taken

- a branch is covered if both true- and false-branches are taken in some execution

In Python (or any interpreted language) statement/branch coverage are especially important

- a code that is not covered is never executed; it might be (almost) complete nonsense

Coverage.py is a widely used coverage tool for Python

- <https://coverage.readthedocs.io/en/coverage-4.4.1/>

coverage.py usage

`coverage run PYTHON_PROGRAM`

- executes the program and monitors which statements are executed

`coverage run -branch PYTHON_PROGRAM`

- executes the program and monitors which statements are executed and which branches are followed

`coverage html`

- generates an HTML report showing coverage of the last run
- can only be executed after coverage-run as shown above
- the result is placed in `htmlcov/index.html`

Virtualenv

It is hard to maintain consistent development environment

- your code might require 3rd party libraries and specific versions of these
- different environments might provide different libraries and these might change as system administrator updates the system
- you might want to develop on one machine but make sure that it works on another (i.e., develop on personal machine, run on ecelinux[1-3])

virtualenv simplifies the management of virtual python environment

- not a virtual machine! no overhead! (except for extra space)
- maintains local copies of desired libraries
- multiple virtual environments can co-exist together
- see course web site for setup details
 - <https://ece.uwaterloo.ca/~agurfink/ece650/tutorial/2017/08/25/virtualenv-intro>

Python

Course Website

<https://ece.uwaterloo.ca/~agurfink/ece650/tutorial/2017/09/13/python>

The Python Tutorial

<http://docs.python.org/tutorial/>

Think Python, 2nd edition

<http://www.greenteapress.com/thinkpython/>

Data Programming course notes

<http://courses.cs.washington.edu/courses/cse140/13wi/calendar/lecturelist.html>

Python Tutor

<http://www.pythontutor.com/>