Python

ECE 650
Methods & Tools for Software Engineering (MTSE)
Fall 2023

Presented by
Dr. Albert Wasef

Used by permission from Prof. Arie Gurfinkel

UNIVERSITY OF
WATERLOO

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

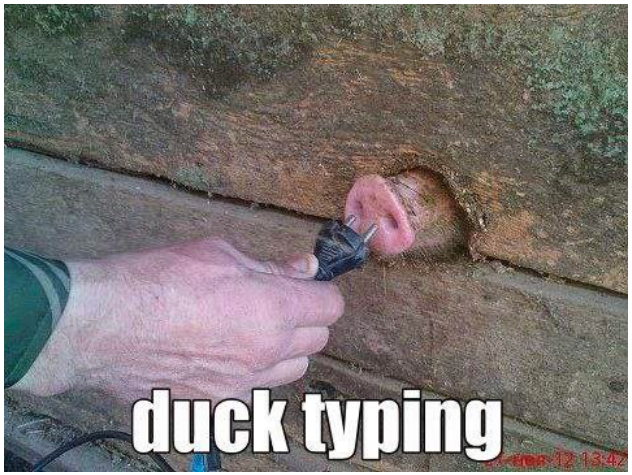
Python 3.0 released in 2008

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

We use Python v3 in the course

- January 2020 is EOL of Python2

Duck Typing



http://stereobooster.github.io/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)

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 interpretes 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

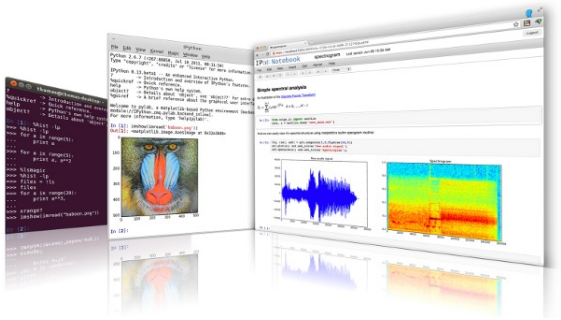
IP[y]: IPython
Interactive Computing

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 echo system

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

Runs in your shell

```
$ ipython
```

Jupyter Notebooks

A web-application for sharing computational documents

- http://jupyter.org

Powered by

- Python (with many libraries) for data processing
- Markdown for documentation

Great tool for keep track of documentation, data, and computation

Great tool to learn Python and learn new libraries

Supported by Visual Studio Code

Share through many cloud providers

- https://colab.research.google.com/

https://docs.python.org/3/tutorial/index.html

PYTHON TUTORIAL

https://git.uwaterloo.ca/ece650-f23/py

https://git.uwaterloo.ca/ece650-f23/calc-py-skel

CALCULATOR EXAMPLE

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

Calls to `assertXXX()` methods indicate test results

Entry point for the test when ran from command line

```
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()
```

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

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., #statements executed / # 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 nonesense

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 `htmlconv/index.html`

Regular Expressions

RegEx – a language to specify and discover patterns in strings

(Basic) Syntax

```
regex ::=   letter                  (exact match)
            ( regex )               (grouping)
            regex?                  (zero or one)
            regex+                  (one or more)
            regex*                  (zero or more)
            regex regex             (sequence)
            regex | regex           (choice)

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
```

Suggested 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

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 `eceubuntu`)

`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://git.uwaterloo.ca/ece650-f23/tutorials/-/blob/master/2020-08-25-virtualenv-intro.md

Python

Course Website

https://git.uwaterloo.ca/ece650-f23/tutorials/-/blob/master/2020-09-13-python.md

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/