CSCE274 Robotic Applications and Design
Fall 2020
Introduction to Python

Ioannis REKLEITIS

Computer Science and Engineering

University of South Carolina

yiannisr@cse.sc.edu

Based on http://tdc-www.harvard.edu/Python.pdf

Outline

- Comparison of programming languages
- Brief overview of Python
- Basics
- Main elements

Programming Language Comparison

- Compiled vs. interpreted languages
- Static vs. dynamic types
- Metrics
  - CPU efficiency
  - Memory efficiency
  - Programmer efficiency, e.g.,
    - Memory manual vs. automatic
    - Libraries of code to use -- re-use

C/C++

- Static compile time type system
- Only dynamic type support = void*
- CPU/memory efficient
- Programmers have to explicitly manage memory
- Wide use in embedded systems and robotics (e.g., Robotic Operating System – ROS)

Java

- Flexible compile time type system
- Both static and dynamic types
- CPU efficiency -- 1-2x worse than C
- Mem efficiency -- 2x worse than C
- Programmers have the benefits of a statically typed language and at the same time flexibility of checking types at run time
- In robotics less commonly used

Compile Time Typing Pro/Con

- Pros
  - Errors detected at compile time
  - Good performance being the code compiled
  - Easier optimization
  - More control from the programmer
  - Enables easier performance optimization
  - …
- Cons
  - More verbose code
  - Maintaining type info can be burdensome
  - …

Python

- Interpreted language
- Dynamic language
- CPU/memory inefficient compared to compile time languages
- Easy to prototype
- Used also in robotics, e.g., ROS

Dynamic Typing Pro/Con

- Advantages
  - Less verbose code
  - More flexibility because of the dynamic typing structure
  - …
- Disadvantages
  - Less readable, because type info is missing
  - Worse performance
  - Worse compile time error detection
    - Compensate with unit tests
  - …

How to choose?

- According to the efficiency metrics
- “Legacy” driver
- Libraries that are available
- Standard/Open-source

Python

- Allows programmers to focus more on the algorithmic aspects rather than the intrinsic aspects of a language

- Open source general-purpose language
- Object Oriented, Procedural, Functional
- Interpreted language

- Downloads: http://www.python.org
- Documentation: http://www.python.org/doc/
- Free book: http://www.diveintopython.org

Python 2 or 3?

- Python 2.x is legacy, Python 3.x is the present and future of the language
- Python 3.x has slightly worse library support
- Python 2.x is still the default in many Unix-based operating systems

Binaries

- Python is pre-installed with Linux and Mac OS X
- Windows binaries from http://python.org

Python interpreter

- Interactive interface to Python

```
%python
Python 2.7.8 (default, Jun 30 2014, 16:03:49) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Python interpreter evaluates inputs

```
>>> 3*(7+2)
27
```

- Python prompts with '>>>'
- To exit Python:
  - CTRL-D in Linux and Mac OS X
  - CTRL-C in Windows

Running Python Programs

- Pass as argument the program

%python filename.py

Many standard libraries

- https://docs.python.org/2/py-modindex.html

A code sample

```
“““This is a code
sample”””
x = 34 - 23 # A comment.
y = “Hello” # Another one.
z = 3.45
if z == 3.45 or y == “Hello”:
    x = x + 1
    y = y + “ World” # String concat.
print x
print y
```

Variables

- Names are case sensitive and cannot start with a number.
  - They can contain letters, numbers, and underscores.

    bob Bob _bob _2_bob_ bob_2 BoB

  - Some words are reserved, e.g., and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

- Type not specified, evaluated at runtime

Variables – Data type

- Integers (default for numbers)

  z = 5 / 2 # Answer is 2, integer division.

- Floats

  x = 3.456

- Strings
  - Can use “” or ‘’ to specify.

  “abc” ‘abc’ (Same thing.)

  - Unmatched can occur within the string.

  “matt’s”

  - Use triple double-quotes for multi-line strings or strings than contain both ‘ and “ inside of them:

  “““a‘b“c”””

Variables – Assignment

- Variables are created when placed it on the left side of an assignment

  x=3

- Binding a variable in Python means setting a name to hold a reference to some object
  - Assignment creates references, not copies
- Names in Python do not have an intrinsic type, instead objects have types
  - Python determines the type of the reference automatically based on the data object assigned to it
- A reference is deleted via garbage collection after any names bound to it have passed out of scope

Variables - Reference semantics

- Assignment manipulates references
  - x = y does not make a copy of the object y references
  - x = y makes x reference the object y references

- Some data types are immutable, e.g., integer, float, stringThe data 3 we created is of type >>> x = 3

```
>>> x = x + 1
>>> print x
4
```

- 3 is of type integer, and is stored in memory, a new portion of the memory is allocated to 4

Variables - Reference semantics

- Some other data types (e.g., lists, dictionaries, user-defined types) are "mutable"
  - The change of the data directly happens in place
  - No copy of the data into a new memory address each time
  - If two variables are referencing to the same data, both variables are changed
- Example:

```
>>> a = [1, 2, 3] # a now references the list [1, 2, 3]
>>> b = a # b now references what a references
>>> a.append(4) # this changes the list a references
>>> print b # if we print what b references,
[1, 2, 3, 4] # It has changed…
```

Common errors

- Accessing non-existent names
  - If you try to access a name before it's been properly created (by placing it on the left side of an assignment), you'll get an error

```
>>> y
Traceback (most recent call last):
File "<pyshell#16>", line 1, in -toplevel-
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

Sequence types

- Tuple
  - A simple immutable ordered sequence of items
  - Items can be of mixed types, including collection types
- Strings
  - Immutable
  - Conceptually very much like a tuple
- List
  - Mutable ordered sequence of items of mixed types

Sequence types

- Tuples are defined using parentheses (and commas)

  >>> tu = (23, ‘abc’, 4.56, (2,3), ‘def’)

- Strings are defined using quotes (“, ‘, or “““)

  >>> st = “Hello World”

  >>> st = ‘Hello World’

  >>> st = ““““This is a multi-line

  string that uses triple quotes.””””

- Lists are defined using square brackets (and commas)

  >>> li = [“abc”, 34, 4.34, 23]

Sequence types

- Individual members of a tuple, list, or string can be accessed using square bracket "array" notation
  - 0-index based

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1] # Second item in the tuple.
 'abc'
>>> li = ["abc", 34, 4.34, 23]
>>> li[1] # Second item in the list.
 34
>>> st = "Hello World"
>>> st[1] # Second character in string.
 'e'
```

Sequence types

- Positive and negative indices

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Positive index: count from the left, starting with 0

```
>>> t[1]
'abc'
```

Negative lookup: count from right, starting with –1

```
>>> t[-3]
4.56
```

Sequence types

- Slicing

  Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying before the second index

  ```
  >>> t[1:4]
  ('abc', 4.56, (2,3))
  ```

  Negative indices can also be used when slicing

  ```
  >>> t[1:-1]
  ('abc', 4.56, (2,3))
  ```

Sequence types

- Slicing

  Omit the first index to make a copy starting from the beginning of the container

  ```
  >>> t[:2]
  (23, 'abc')
  ```

  Omit the second index to make a copy starting at the first index and going to the end of the container

  ```
  >>> t[2:]
  (4.56, (2,3), 'def')
  ```

Sequence types

- To make a copy of an entire sequence, you can use [:]

```
>>> t[:]
(23, 'abc', 4.56, (2,3), 'def')
```

- Note the difference between these two lines for mutable sequences:

```
>>> list2 = list1 # 2 names refer to 1 ref
# Changing one affects both
>>> list2 = list1[:] # Two independent copies, two refs
```

Whitespace

- Whitespace is meaningful in Python: especially indentation and placement of newlines
- Use a newline to end a line of code
  - Use \ when must go to next line prematurely
- No braces { } to mark blocks of code in Python… Use consistent indentation instead
  - The first line with less indentation is outside of the block
  - The first line with more indentation starts a nested block
- Often a colon appears at the start of a new block. (e.g. for function and class definitions)

Control of flow

```
if x == 3:
        print "X equals 3."
elif x == 2:
        print "X equals 2."
else:
        print "X equals something else."
print "This is outside the 'if'."
```

```
x = 3
while x < 10:
        if x > 7:
                x += 2
                continue
        x = x + 1
        print "Still in the loop."
        if x == 8:
                break
print "Outside of the loop."
```

```
for x in range(10):
        if x > 7:
                x += 2
                continue
        x = x + 1
        print "Still in the loop."
        if x == 8:
                break
print "Outside of the loop."
```

Exceptions

- Possible to use try/exception blocks

```
>>> try:
...         1 / 0
...     except:
...         print('That was silly!')
...     finally:
...         print('This gets executed no matter what')
...
That was silly!
This gets executed no matter what
```

Useful operators

- Assignment uses = and comparison uses ==
- For numbers + - * / % are as expected
- Special use of + for string concatenation
- Special use of % for string formatting (as with printf in C)
- Logical operators are words (and, or, not), not symbols
- The basic printing command is print

Functions

- def creates a function and assigns it a name
- return sends a result back to the caller

Functions

- Arguments are passed by assignment
- Passed arguments are assigned to local names
- Assignment to argument names don't affect the caller
- Changing a mutable argument may affect the caller

```
def changer (x,y):
    x = 2 # changes local value of x only
    y[0] = 'hi' # changes shared object
```

Functions

- Default values can be assigned to arguments

```
def func(a, b, c=10, d=100):
    print a, b, c, d
>>> func(1,2)
1 2 10 100
>>> func(1,2,3,4)
1,2,3,4
```

Functions

- All functions in Python have a return value
  - even if no return line inside the code
- Functions without a return return the special value
  - None
- There is no function overloading in Python
  - Two different functions can't have the same name, even if they have different arguments
- Functions can be used as any other data type. They can be:
  - Arguments to function
  - Return values of functions
  - Assigned to variables
  - Parts of tuples, lists, etc

Classes and objects

- Being object-oriented, classes are available

```
class Robot(object):
    def __init__(self):
        “““ this is the constructor. ”””
        # The code goes here.
```

Files

- Closing a file is implicit with with block

```
with open('file_path', 'r') as input_file:
    content = input_file.read()
    for line in content:
        print line
```

Modules

- Code reuse
  - Routines can be called multiple times within a program
  - Routines can be used from multiple programs
- Namespace partitioning
  - Group data together with functions used for that data
- Implementing shared services or data
  - Can provide global data structure that is accessed by multiple subprograms

Modules

- Modules are functions and variables defined in separate files
- Items are imported using from or import

```
from module import function
function()

import module
module.function()
```

- Modules are namespaces
  - Can be used to organize variable names, i.e.

```
atom.position = atom.position - molecule.position
```

Comments

- Start comments with # – the rest of line is ignored.
- Can include a “documentation string” as the first line of any new function or class that you define.
- The development environment, debugger, and other tools use it: it’s good style to include one.

```
def my_function(x, y):
    """This is the docstring. This
    function does blah blah blah."""
    # The code would go here...
```

CODING STANDARD

- When possible, follow ROS style guides: http://wiki.ros.org/PyStyleGuide
- Important: PYTHON IS WHITESPACE SENSITIVE
  - Always use spaces instead of tabs
  - 4 spaces per indentation level preferred
- Naming conventions:
  - package_name
  - ClassName
  - method_name
  - field_name