Introduction to Programming for Public Policy Week 1 (Python)

Eric Potash

March 29, 2018

Introduction to Programming

Python

For the remainder of class we will be learning a high-level programming language called **Python**.

What is a Programming language?

Assembly

- Computers run programs written in low-level language called *assembly*
- Assembly code is very fast and efficient but:
  - The code is not portable between different hardware
  - The code is difficult to write and read

Assembly "Hello, World"

This is a program for printing the text "Hello, World" in assembly:

```
global    _start

section   .text
_start:   mov       rax, 1
mov       rdi, 1
mov       rsi, message
mov       rdx, 13
syscall
mov       rax, 60
xor       rdi, rdi
syscall

section   .data
message:  db        "Hello, World", 10
```

Programming languages

- People created "higher level" programming languages to make our (programmers') lives easier:
  - Stata, SAS, SPSS
  - R
  - Shell
  - Python
  - Java
  - C, C++

Interpretter vs Compiler

- There are two basic ways of translating high-level languages into low-level languages:
  - Interpretter: Code is read and translated and executed line by line (e.g. shell, Python, R)
  - Compiler: Code is read all at once and translated before it is executed
- Generally interpretted languages are easier to read and write but may be slower

Interpretted "Hello, World"

- Shell:

```
echo Hello, World
```

- Python:

```
print('Hello, World')
```

- R:

```
print("Hello, World')
```

Compiled "Hello, World"

- C:

```c
#include <stdio.h>
int main()
{
   printf("Hello, World");
   return 0;
}
```

Language choice

- People have preferences for programming languages just like they do for anything else

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria
  - e.g. speed or efficiency or portability

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria
  - e.g. speed or efficiency or portability
- But often they're subjective

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria
  - e.g. speed or efficiency or portability
- But often they're subjective
  - e.g. many people think that python code is very easy to read and write

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria
  - e.g. speed or efficiency or portability
- But often they're subjective
  - e.g. many people think that python code is very easy to read and write
- And there are many myths

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria
  - e.g. speed or efficiency or portability
- But often they're subjective
  - e.g. many people think that python code is very easy to read and write
- And there are many myths
  - e.g. you'll hear people say that python is faster than R

Language choice

- People have preferences for programming languages just like they do for anything else
- Sometimes these preferences are based on objective criteria
  - e.g. speed or efficiency or portability
- But often they're subjective
  - e.g. many people think that python code is very easy to read and write
- And there are many myths
  - e.g. you'll hear people say that python is faster than R
  - it's not exactly true

Python

Why Python?

In this class we'll use python because it has proven over time to be:

- Easy (relatively) to learn
- Works well for many policy tasks (data analysis, text mining, visualization, modeling, etc.)
- Scales to large applications/datasets
- Has a large developer community

Interactive Interpreter

- Simple way of running python
- Interactively enter text commands like the command line itself

```
$ python
Python 3.6 (default, Sep 16 2015, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more
>>>
```

Calculator

```
$ python
Python 3.6 (default, Sep 16 2015, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more i
>>> 1+1
2
>>>
```

Scripts

- As with the shell, we can write python scripts. E.g. save this to a file called `hello_world.py`:

```
print('Hello, World')
```

- Then execute the script with the python interpreter:

```
$ python hello_world.py
Hello, World
$
```

Types

- So far we've seen three different values in python: `1`, `2`, and `"Hello, World"`
- These have different *types*:
  - 1 and 2 are *integers*
  - "Hello, World" is a *string*

Type inspection

- If you want to know the type of a value in python, you can ask the interpreter:

```
>>> type(1)
<type 'int'>
>>> type('Hello, World')
<type 'str'>
>>> type(3.1415)
<type 'float'>
>>> type('3.1415')
<type 'str'>
```

Variables

- Like in math, variables are an essential part of programming
- We assign values to variables like so:

```
>>> pi = 3.1415
>>> greeting = "Hello, World"
```

Arithmetic with variables

We can do arithmetic on numeric variables like so:

```
>>> pi = 3.1415
>>> r = 2
>>> circumference = 2*pi*r
>>> circumference
12.566
```

More arithmetic

- Basic mathematical operations include: +, -, *, /, and ** (exponeniate)
- Use parentheses to specify an order of operations:

```
>>> 1/(2*pi)
```

Strings

There are also operations on strings. Plus (+) concatenates:

```
>>> 'Hello' + ', ' + 'World'
Hello, World
```

String indexing

- Strings are composed of characters:

0	1	2	3	4
H	e	l	l	o

- You can access a single character through indexing:

```
>>> greeting = 'Hello, World'
>>> greeting[0]
'H'
>>> greeting[5]
','
>>> greeting[-1]
```

String slicing

- You can access multiple characters in a string through slicing:

```
>>> greeting[1:5]
'ello'
```

Debugging

- Mistakes in code result in `errors`:

```
>>> pi = 3.1415
>>> Pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Pi' is not defined
```

- *Debugging* is the process of sorting through errors and fixing them
  - In this case the issue is that variable names are case sensitive

Comments

Python uses the hash (#) for comments, just like the shell:

```
pi = 3.1415     # define the math constant pi
r = 2           # circle's radius
A = 2*pi*r**2   # area of a circle
```