introduction (week 1+)

Ben Bolker

03 September 2019

# Introduction

## Administrative trivia

* Instructors: Dr. Benjamin Bolker and Dr. Weijie Pang
* TAs: Nik Počuča, Steve Cygu, Aghigh Farhadi (marking)
* course web page: <http://bbolker.github.io/math1mp>
* course outline: <http://bbolker.github.io/math1mp/admin/outline.html>
* Grading
	+ Assignments (10%)
	+ Quizzes (5%)
	+ Final project (5%)
	+ Midterm tests (2 $×$ 20%)
	+ Final exam (40%)
* homework assignment announcements policy
(web page, Avenue: **not** in class)
* Laptop policy
* Course material on web page and Avenue to Learn
* Expectations of professor and students
* Textbook (optional); Gries et al. *Practical Programming* 3d ed. (see outline)
* also see [resources](https://github.com/bbolker/math1mp/misc/resources.md)

## Course content

reasonable balance among

* nitty-gritty practical programming instruction:
* … I just sat down in front of a text editor, with nothing but thoughts, and ended up with a program that did exactly what I wanted it to a few hours later … ([ankit panda](https://web.archive.org/web/20160421222949/http%3A//www.ankitpanda.com/pythonicity/))
* conceptual foundations of computing/computer science
* context/culture of mathematical/scientific computing
* interesting applications

## Installing Python

* CodeLab: <http://www.turingscraft.com/go.html>
* [PythonAnywhere](https://www.pythonanywhere.com/)
* Everyone must have access to a computer with Python3 installed.
	+ See [installation instructions](./install_python.html)

# Overview of math/sci computing

## Using computers in math and science

* math users vs. understanders vs. developers
* develop conjectures; draw pictures; write manuscripts
* mathematical proof (e.g. [four-colo(u)r theorem](http://en.wikipedia.org/wiki/Four_color_theorem) and [other examples](http://math.stackexchange.com/questions/1084230/what-are-some-theorems-that-currently-only-have-computer-assisted-proofs)); computer algebra
* applied math: cryptography, tomography, logistics, finance, fluid dynamics, …
* applied statistics: bioinformatics, Big Data/analytics, …
* discrete vs. continuous math

## Running Python

* via **notebooks** (<http://mcmaster.syzygy.ca> or on your own computer)
* via **scripts** + **console** (<http://mcmaster.syzygy.ca/jupyter/user-redirect/lab>)

## Fun!

[**Hello, world**](https://en.wikipedia.org/wiki/%22Hello%2C_World%21%22_program) (always the first program you write in a new computer language)

print('hello, python world!')

## hello, python world!

Python as a fancy calculator (**REPL**, Read-Evaluate-Print-Loop)

print(62\*\*2\*27/5+3)

## 20760.6

*reference*: [Python intro section 3.1.1](https://docs.python.org/3/tutorial/introduction.html)

## Interlude: about Python

* [programming languages](http://crashworks.org/if_programming_languages_were_vehicles/)
	+ Python: [scripting](http://en.wikipedia.org/wiki/Scripting_language); high-level; glue; general-purpose; flexible
	+ contrast: *domain-specific* scripting languages (MATLAB, R, Mathematica, Maple)
	+ contrast: *general-purpose* scripting languages (Perl, PHP)
	+ contrast: general-purpose *compiled* languages (Java, C, C++) (“close to the metal”)
* relatively modern (1990s; Python 3, 2008)
* currently the [5th most popular computer language](http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html) overall (up from 8th in 2015); [most popular for teaching](http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext)
* well suited to mathematical/scientific/technical ([NumPy](http://www.numpy.org); [SciPy](http://www.scipy.org); [Python in Finance](https://www.safaribooksonline.com/library/view/python-for-finance/9781491945360/ch01.html))
* ex.: [Sage](http://www.sagemath.org); [BioPython](http://www.biopython.org)

## the “prime walk” (from [math.stackexchange.ca](http://tinyurl.com/primewalk))

1. start at the origin, heading right, counting up from 1
2. move forward one space, counting up, until you find a prime
3. turn 90$^{∘}$ clockwise
4. repeat steps 2 and 3 until you get bored

code [here](code/primewalk.py) (bbolker.github.io/math1mp/code/primewalk.py)

**Note**:

* easier to understand/modify than write from scratch
* build on existing components (*modules*)

## Interfaces

* integrated development environment (IDE), command line/console (Spyder)
* programming editor
* notebooks
* **not** MS Word! 

## Features

* syntax highlighting, bracket-matching, hot-pasting
* integrated help
* integrated debugging tools
* integrated project management tools
* **most important**: maintain reproducibility; well-defined **workflows**

## Assignment and types (PP $\S2.4$)

* superficially simple
	+ set aside *memory* space, create a symbol that *points to* that space
	+ = is the **assignment operator** (“gets”, not “equals”)
	+ <variable> = <value>
	+ variable names
		- what is legal? (names include letters, numbers, underscores, must start with a letter)
		- what is customary? [convention](https://www.python.org/dev/peps/pep-0008/#id30) is variables\_like\_this (“snake case”)
		- what works well? v vs. temporary\_variable\_for\_loop
		- same principles apply to file, directory/folder names
* variables are of different [**types**](https://docs.python.org/3/library/stdtypes.html)
	+ built-in: integer (int), floating-point (float), complex, **Boolean** (bool: True or False),
	+ *dynamic* typing
		- Python usually “does what you mean”, converts types when sensible
	+ *strong* typing
		- try print(type(x)) for different possibilities (x=3; x=3.0; x="a")
		- *what happens if you try x=a?*
		- **don’t be afraid to experiment!**

**Examples**

x=3
y=3.0
z="a"
q=complex(1,2)
type(x+y) ## mixed arithmetic
type(int(x+y)) ## int(), float() convert explicitly
type(x+z)
type(q)
type(x+q)
type(True)
type(True+1) ## WAT

[^2](As [Dive into Python](http://www.diveintopython3.net/native-datatypes.html) says in a similar context, “Ew, ew, ew! Don’t do that. Forget I even mentioned it.”)

Check out the [Python tutor](http://pythontutor.com/visualize.html#mode=edit) for these examples

## Arithmetic operators, precedence

* exponentiation (\*\*)
* negation (“unary minus”) (-)
* multiplication/division (\*,/,//=integer division,%=remainder (“modulo”))
* addition/subtraction (+, - (“binary”))

Use parentheses when in doubt!

**Puzzle:** what is -1\*\*2? Why?

## Logical operators (PP $\S5.1$)

* comparison: (==, !=)
* inequalities: >, <, >=, <=,
* basic logic: (and, or, not)
* remember your truth tables, e.g. not(a and b) equals (not a) or (not b)

a = True; b = False; c=1; d=0
a and b
not(a and not b)
a and not(b>c)
a==c ## careful!
not(d)
not(c)

**operator precedence**

* remember [order of operations in arithmetic](http://xkcd.com/992/)
* not has higher precedence than and, or. When in doubt use parentheses …

From [CodingBat](http://codingbat.com/prob/p120546):

We have two monkeys, a and b, and the parameters a\_smile and b\_smile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return True if we are in trouble.

monkey\_trouble(True, True) → True
monkey\_trouble(False, False) → True
monkey\_trouble(True, False) → False

## Truth tables

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | A and B | A or B | not A |
| True | True | True | True | False |
| True | False | False | True | False |
| False | True | False | True | True |
| False | False | False | False | True |

## Logical expressions

* The logical expression: not not a and not b or a is equivalent to ((not (not a)) and (not b)) or a since the operator not takes precedence over the operators and and or.
* So if a = True and b = False this evaluates to True
* Since not not a is equivalent to a, we can simplify the expression to just (a and not b) or a.
* Can we simplify this further?

What can we do with not a and not b ?

## More [CodingBat](http://codingbat.com) problems

* [squirrel\_play](http://codingbat.com/prob/p135815)
* [cigar\_party](http://codingbat.com/prob/p195669)

## String operations (PP chapter 4)

*reference*: [Python intro](https://docs.python.org/3/tutorial/introduction.html) section 3.1.2

* Less generally important, but fun
* + concatenates
* \* replicates and concatenates
* in searches for a substring

a = "xyz"
b = "abc"
a+1 ## error
a+b
b\*3
(a+" ")\*5
b in a

CodingBat problems:

* [make\_abba](http://codingbat.com/prob/p182144)
* [make\_tags](http://codingbat.com/prob/p132290)

One more useful string operation: len(s) returns the length (number of characters)

# Indexing and slicing

## Indexing

* Extracting elements is called **indexing** a list
* Indexing [starts from zero](http://xkcd.com/163/)
* Negative indices count backward from the end of the string
(-1 is the last element)
* Indexing a non-existent element gives an error



slicing

## Slicing

* Extracting (consecutive) sets of elements is called [**slicing**](http://stackoverflow.com/questions/509211/explain-pythons-slice-notation)
* Slicing non-existent element(s) gives a truncated result
* Slicing specifies *start*, *end*, *step* (or “stride”)
* Leaving out a bit goes from the beginning/to the end
* Slicing works on strings too!

x[:] # everything
x[a:b] # element a (zero-indexed) to b-1
x[a:] # a to end
x[:b] # beginning to b-1
x[a:b:n] # from a to b-1 in steps of n

* generate a list of odd numbers from 3 to 15
* reverse a string?

## String slicing practice

From CodingBat:

* [first\_two](http://codingbat.com/prob/p184816)
* [first\_half](http://codingbat.com/prob/p107010)

## Methods

* Objects in Python have **classes** (string, integer, etc.)
* Classes have **methods** - things you can to do the objects
* You use a method by calling **.()**
	+ yes, this seems weird at first.
* methods may or may not have **arguments**

## String methods: examples

Strings have lots of [methods](https://docs.python.org/3/library/stdtypes.html#string-methods), for example:

x = "abcdef"
x.upper()

## 'ABCDEF'

x.capitalize()

## 'Abcdef'

x.endswith("f")

## True

x.startswith("qrs")

## False

x.islower()

## True