introduction (week 1+)

Ben Bolker

02 September 2019

Introduction

Administrative trivia

- Instructors: Ben Bolker and Weijie Pang
- TAs: Nik Počuča, Steve Cygu, TBA (marking)
- Grading
  - Assignments (10%)
  - Quizzes (5%)
  - Final project (5%)
  - Midterm tests (2 $\times$ 20%)
  - Final exam (40%)
- homework assignment announcements policy
  (web page, Avenue: **not** in class)

- Laptop policy
- Course material on Github: `https://bbolker.github.io/math1mp`
  and Avenue to Learn
- Expectations of professor and students
- Textbook (optional); Gries et al. *Practical Programming* 2d ed., available from the bookstore, the publisher (The Pragmatic Bookshelf), or amazon.ca.
- also see resources

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)

- conceptual foundations of computing/computer science
- context/culture of mathematical/scientific computing
- interesting applications

Installing Python

- CodeLab: `http://www.turingscraft.com/go.html`
- PythonAnywhere
- Everyone must have access to a computer with Python3 installed.
  - See installation instructions

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 and other examples); computer algebra
- applied math: cryptography, tomography, logistics, finance, fluid dynamics, ...
- applied statistics: bioinformatics, Big Data/analytics, ...
- discrete vs. continuous math

Running Python

Fun!

Hello, world (always the first program you write in a new computer language)

```
print('hello, python world!')
```

```
## hello, python world!
```

Python as a fancy calculator:

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

```
## 20760.6
```

reference: Python intro section 3.1.1

Interlude: about Python

- programming languages
  - Python: scripting; 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 overall (up from 8th in 2015); most popular for teaching
- well suited to mathematical/scientific/technical (NumPy; SciPy; Python in Finance)
- ex.: Sage; BioPython

the "prime walk" (from math.stackexchange.ca)

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 (`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 §2.4*)*

- 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 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**
  - 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 says in a similar context, "Ew, ew, ew! Don't do that. Forget I even mentioned it.")

Check out the Python tutor 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 §5.1)

- 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
- `not` has higher precedence than `and`, `or`. When in doubt use parentheses ...

From CodingBat:

> 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 problems

- squirrel_play
- cigar_party

String operations (PP chapter 4)

reference: Python intro 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
- make_tags

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
- Negative indices count backward from the end of the string (-1 is the last element)
- Indexing a non-existent element gives an error

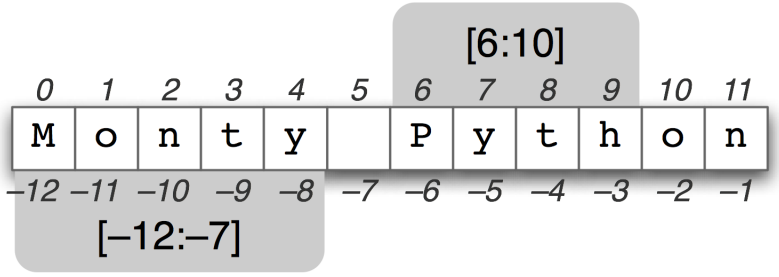

Figure 1: slicing

Slicing

- Extracting (consecutive) sets of elements is called **slicing**
- 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
- first_half