CIS192 Python Programming

Introduction

Robert Rand

University of Pennsylvania

August 27, 2015

Outline

Grading

- Homeworks: 65% of grade.
- Final Project: 30% of grade.
- Participation: 5% of grade.

Homework

- Due Sunday at midnight, ten days after class
- One late week
- Can discuss with a partner
- Code must be your own
- Cite partner / all references used
- Submit via Canvas

Final Project

- Work with up to one partner
- Proposal due Nov 8th
- Demo ready version due last day of class
- Final version due last day of exams

Office Hours

- Instructor: Robert Rand
  - Thursday 1:30 - 3:30pm, Levine 513
- TA: Joseph Cappadona
  - Tuesday 3:00 - 5:00pm, DRL 4E9
- TA: Raymond Yin
  - Friday 3:00 - 5:00pm, Towne 303

Shared Lecture

- Class: CIS 19X Shared Lecture
  - Room: Towne 100
  - Time: Tuesdays 6:00 - 7:30
  - Only a few meetings per semester
  - Includes Unix skills, version control
- Instructor: Swapneel Sheth

Programming Environment

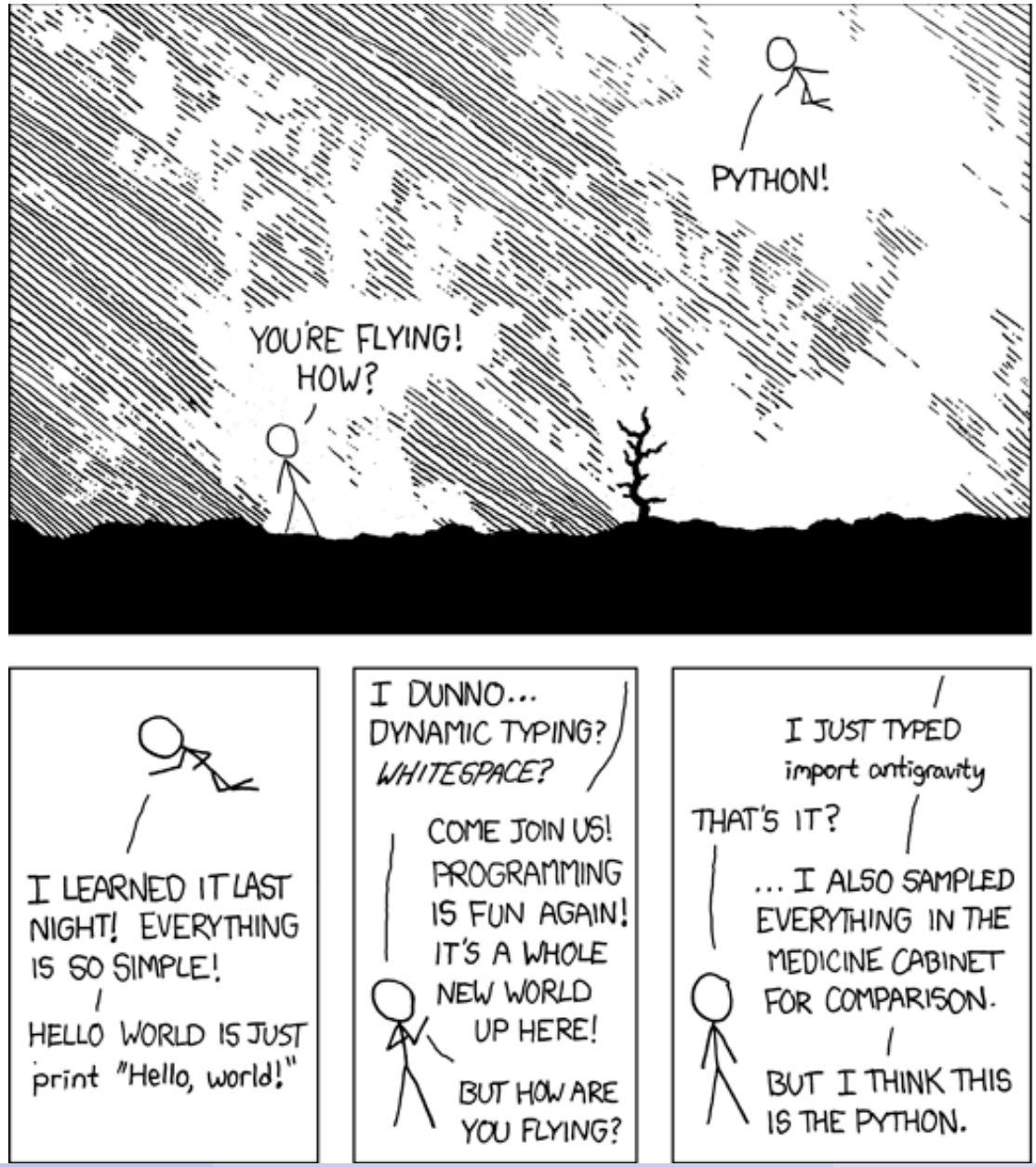
- PyDev for Eclipse Recommended
  - Emacs is also good
- *Make sure your editor interprets TAB as four spaces*
  - Python is whitespace-sensitive!

Piazza

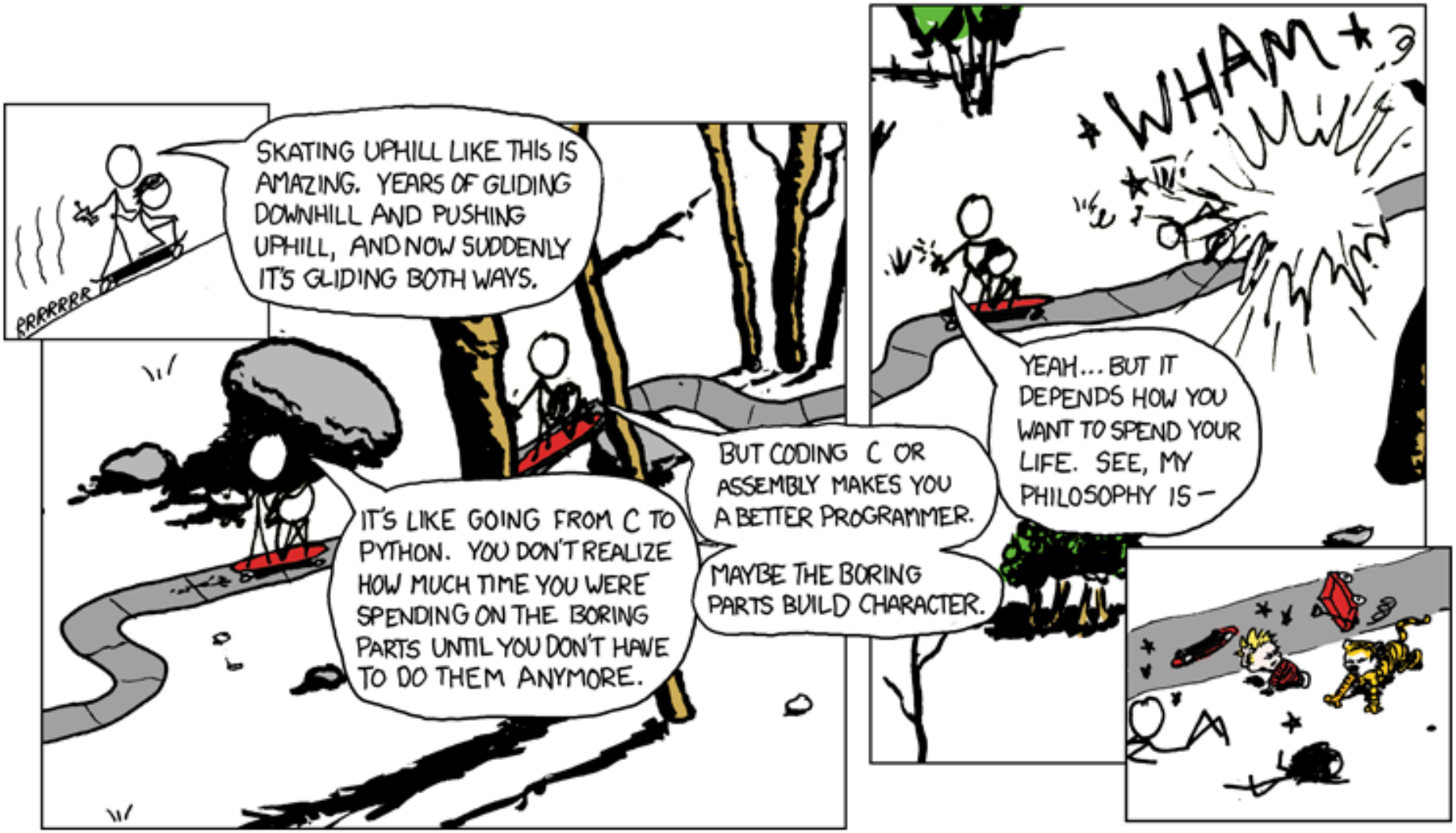
Use Piazza to find project partners and discuss lectures/homeworks with your peers.

Outline

Easy to Learn



Easy to Use


PRRRRRR
SKATING UPHILL LIKE THIS IS AMAZING. YEARS OF GLIDING DOWNHILL AND PUSHING UPHILL, AND NOW SUDDENLY IT'S GLIDING BOTH WAYS.
IT'S LIKE GOING FROM C TO PYTHON. YOU DON'T REALIZE HOW MUCH TIME YOU WERE SPENDING ON THE BORING PARTS UNTIL YOU DON'T HAVE TO DO THEM ANYMORE.
BUT CODING C OR ASSEMBLY MAKES YOU A BETTER PROGRAMMER.
MAYBE THE BORING PARTS BUILD CHARACTER.
WHAM
YEAH... BUT IT DEPENDS HOW YOU WANT TO SPEND YOUR LIFE. SEE, MY PHILOSOPHY IS—

Easy to Abuse

```
for thing in something:
    if type(thing) == str:
        string_handle()
    elif(thing == None):
        ...
```

REPL

```
MacBook-Pro:~ Reuven$ python
Python 2.7.10 (default, Jul 14 2015, 19:46:27)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.39)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World!"
Hello World!
>>>
```

- Read Evaluate Print Loop
- Type “Python” at the terminal
- Test out language behavior here
- Get information with `dir()`, `help()`, `type()`

2.7 vs. 3.4

Python 2.7	Python 3.4
print “hello world!”	print(“hello world!”)
1/2 = 0	1/2 = 0.5
	1 // 2 = 0
No Unicode support	Unicode support
More popular	Gaining popularity
Better library support	Good library support

Solution

- `from __future__ import absolute_import, division, print_function`
- Changes Python 2.7's behavior to mimic 3.x's
- Best of both worlds
- Add to the top of every .py file.

Static Types

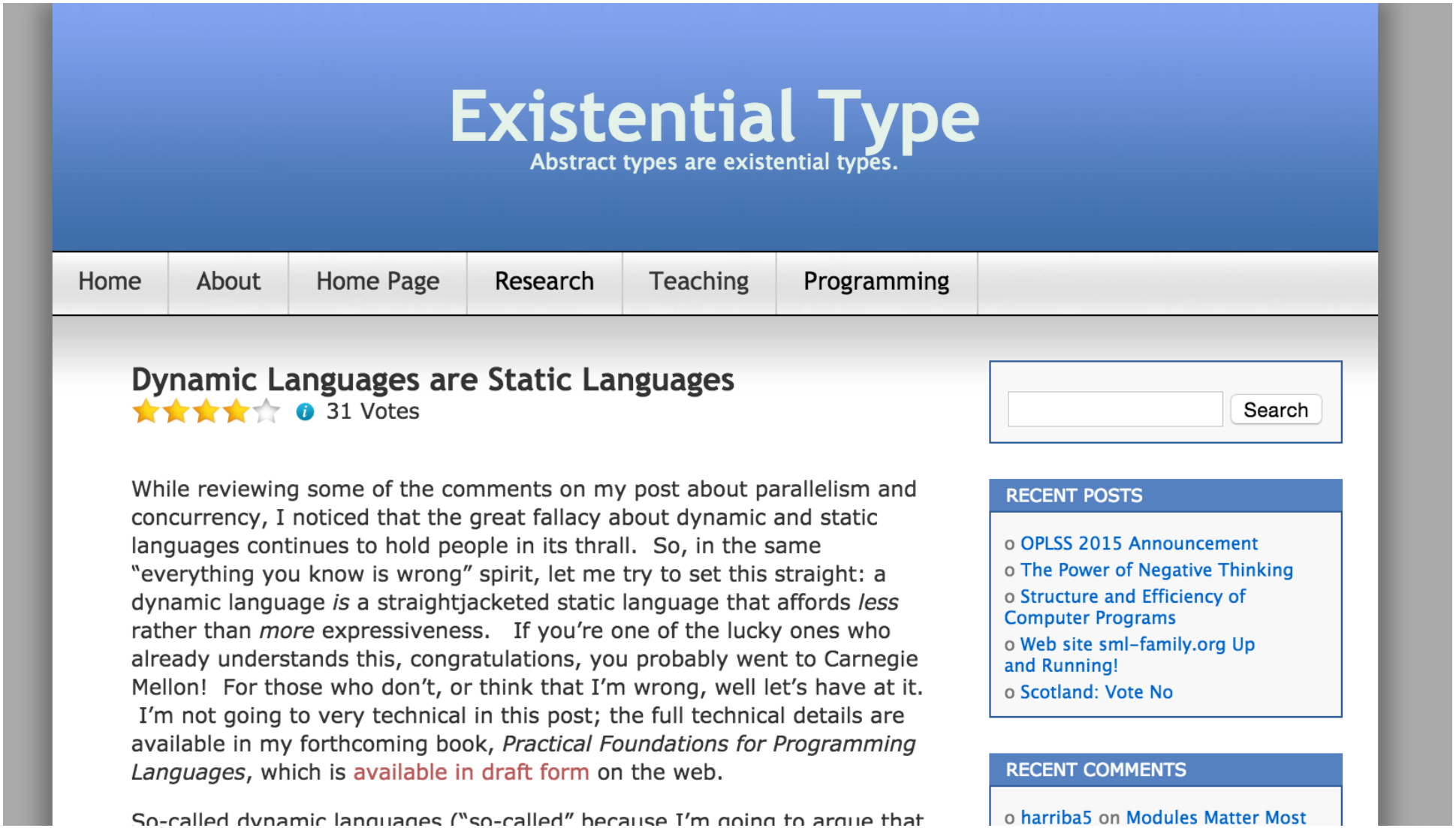
```haskell
-- Type annotation (optional)
fib :: Int -> Integer

-- With self-referencing data
fib n = fibs !! n
        where fibs = 0 : scanl (+) 1 fibs
        -- 0,1,1,2,3,5,...
```

What are *static types*?

- A form of *integrated specification*
- Enforced at compile time.
- Verbose
- *Python does not have these.*

```java
public static int fibonacci(int fibIndex) {
    if (memoized.containsKey(fibIndex)) {
        return memoized.get(fibIndex);
    } else {
        int answer = fibonacci(fibIndex - 1) + f
        memoized.put(fibIndex, answer);
        return answer;
```

Untyped?


Existential Type
Abstract types are existential types.
Home
About
Home Page
Research
Teaching
Programming
Dynamic Languages are Static Languages
31 Votes
While reviewing some of the comments on my post about parallelism and concurrency, I noticed that the great fallacy about dynamic and static languages continues to hold people in its thrall. So, in the same "everything you know is wrong" spirit, let me try to set this straight: a dynamic language *is* a straightjacketed static language that affords *less* rather than *more* expressiveness. If you're one of the lucky ones who already understands this, congratulations, you probably went to Carnegie Mellon! For those who don't, or think that I'm wrong, well let's have at it. I'm not going to very technical in this post; the full technical details are available in my forthcoming book, *Practical Foundations for Programming Languages*, which is available in draft form on the web.
So-called dynamic languages ("so-called" because I'm going to argue that
Search
RECENT POSTS
o OPLSS 2015 Announcement
o The Power of Negative Thinking
o Structure and Efficiency of Computer Programs
o Web site sml-family.org Up and Running!
o Scotland: Vote No
RECENT COMMENTS
o harriba5 on Modules Matter Most

Dynamic Types

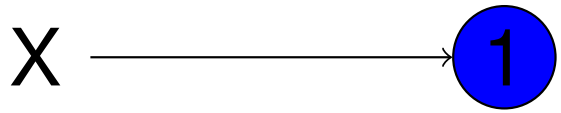
- Basically a system of tags.
- Let's see how they work in practice.

Identifiers, Names, Variables

- All 3 mean the same thing
- [A-Za-z0-9_] First character cannot be a number
- Variable naming convention
  - Functions and variables: lower_with_underscore
  - Constants: UPPER_WITH_UNDERSCORE

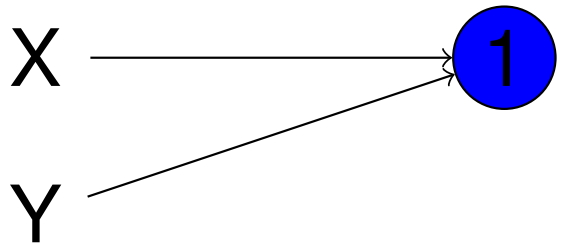
Binding

- x = 1
- y = x
- x = 'a'



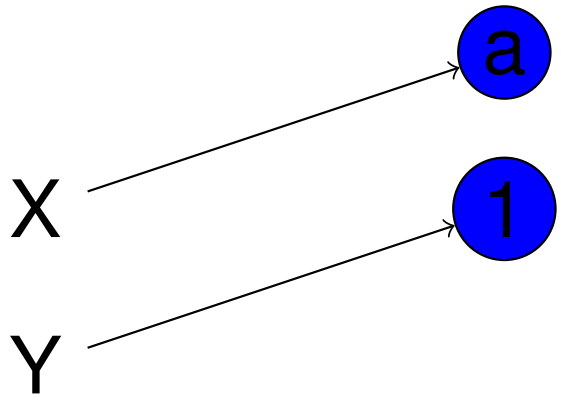
Binding

- x = 1
- y = x
- x = 'a'



Binding

- x = 1
- y = x
- x = 'a'



Types

- Every object has a type
- Inspect types with `type(object)`
- `isinstance(object, type)` checks type heirarchy
- Types can be compared for equality but usually want isinstance()
- Some types:
  - int, float, complex
  - str, bytes, tuple
  - list,bytearray
  - range, bool, None
  - function

Objects

- Python treats all data as objects
- Identity
  - Memory address
  - Does not change
- Type
  - Does not change
- Value
  - Mutable $\rightarrow$ [1,2]
  - Immutable $\rightarrow$ (1,2)

Math

- Literals
  - Integers: `1, 2`
  - Floats: `1.0, 2e10`
  - Complex: `1j, 2e10j`
  - Binary: `0b1001`, Hex: `0xFF`, Octal: `0o72`
- Operations
  - Arithmetic: + - * /
  - Power: **
  - Integer division: //
  - Modulus: %
  - Bitwise: `<< >> & | ^`
  - Comparison: `<, >, <=, >=, ==, !=`
- Assignment Operators
  - `+= *= /= &=` ...
  - No ++ or `--`

Strings

- Can use either single or double quotes
- Use single to show double flip-flop "'" $\rightarrow$ ' and '"' $\rightarrow$ "
- Triplequote for multiline string
- Can concat strings by sepating string literals with whitespace
- All strings are unicode
- Prefixing with r means raw. No need to escape: r'\n'

Conditionals

- One if block
- Zero or more elif blocks
- Zero or one else block
- Booleans: `True False`

Sequences

- Immutable
  - Strings, Tuples, Bytes
- Mutable
  - Lists, Byte Arrays
- Operations
  - `len()`
  - Indexing
  - Slicing
  - `in`
  - `not in`

Range

- Immutable sequence of numbers
- `range(stop), range(start,stop) range(start,stop,step])`
- start defaults to 0
- step defaults to 1
- All numbers in [start,stop) by incrementing start by step
- Negative steps are valid
- Memory efficent: Calculates values as you iterate over them

Loops

- For each loops
  - Iterate over an object
- While loops
  - Continues as long as condition holds
- Both
  - else: executes after loop finishes
  - break: stops the loop and skips the else clause
  - continue: starts the next iteration of the loop

Functions

- Functions are first class
  - Can pass them as arguments
  - Can assign them to variables
- Define functions with a def
- return keyword to return a value
- If a function reaches the end of the block without returning
  It will return None (null)

Imports

- Allow use of other python files and libraries
- imports: `import math`
- Named imports: `import math as m`
- Specific imports: `from math import pow`
- Import all: `from math import *`