Instructor: Paul Nguyen (paul_nguyen [at] utexas [dot] edu)
TAs: Farzam Farahani, Eric Wei

Introduction to Python

THE APTAMER STREAM

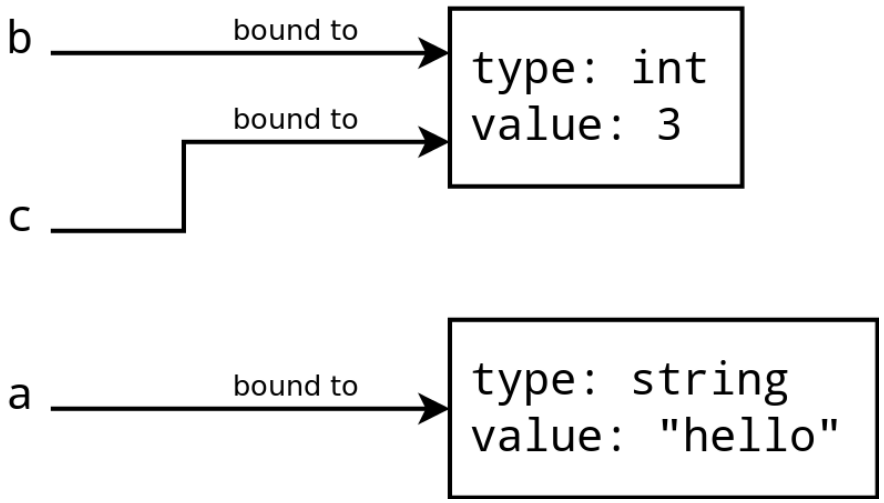
Typefaces: Futura, Consolas
Theme: Depth
Colors: Office
Spring 2016

Executed Code:
Variable Assignment

```
a = 3
b = a
c = a
a = "hello"
```

Variables

Values in Memory



http://swcarpentry.github.io/training-course/2013/05/python-variables-a-diagram-that-is-not-a-concept-map/

Week 1: Introduction, Computers, Variables and Types, Operations

Goals for this mini-course

By the end of this course, you should:

- Be able to **write** and **debug** programs in Python **independently**
- Be able to come up with programs to solve basic problems
- Be able to consider certain optimizations to improve program runtime
- Understand how computer science is useful for automation
- Understand basic **algorithms** (like sorting algorithms) and **data structures** (lists, dictionaries)

What is Python?

Python is a widely used **high-level**, general-purpose, interpreted, dynamic programming language. Its design philosophy **emphasizes code readability**, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable **clear programs** on both a small and large scale. Python supports multiple programming paradigms, including **object-oriented**, imperative and functional programming or procedural styles. It features a **dynamic type system** and **automatic memory management** and has a **large and comprehensive standard library**. *(from Wikipedia)*

Course overview

- Weekly problems (around 3 to 5 per week)
- Typing speed is not important
- Visit/email/message me if help is needed
- Slides marked with an asterisk (*) at the beginning are optional
- Topics: Python syntax, programming concepts, (very) basic data structures and algorithms, applications to biology
- Recommended books (this course is partially based on these):
  - Think Python, Downey (available online from author for free)
  - Python Programming for Biology, Stevens and Boucher (not free)
- Resources:
  - Bioinformatics practice on Rosalind (free)
  - Stack Overflow for help (but try not to become too reliant on it)
  - Highly recommended: Python 3 documentation (https://docs.python.org/3/, free)

Installing Python

Visit https://www.python.org/downloads/

Directions online from me or from other websites

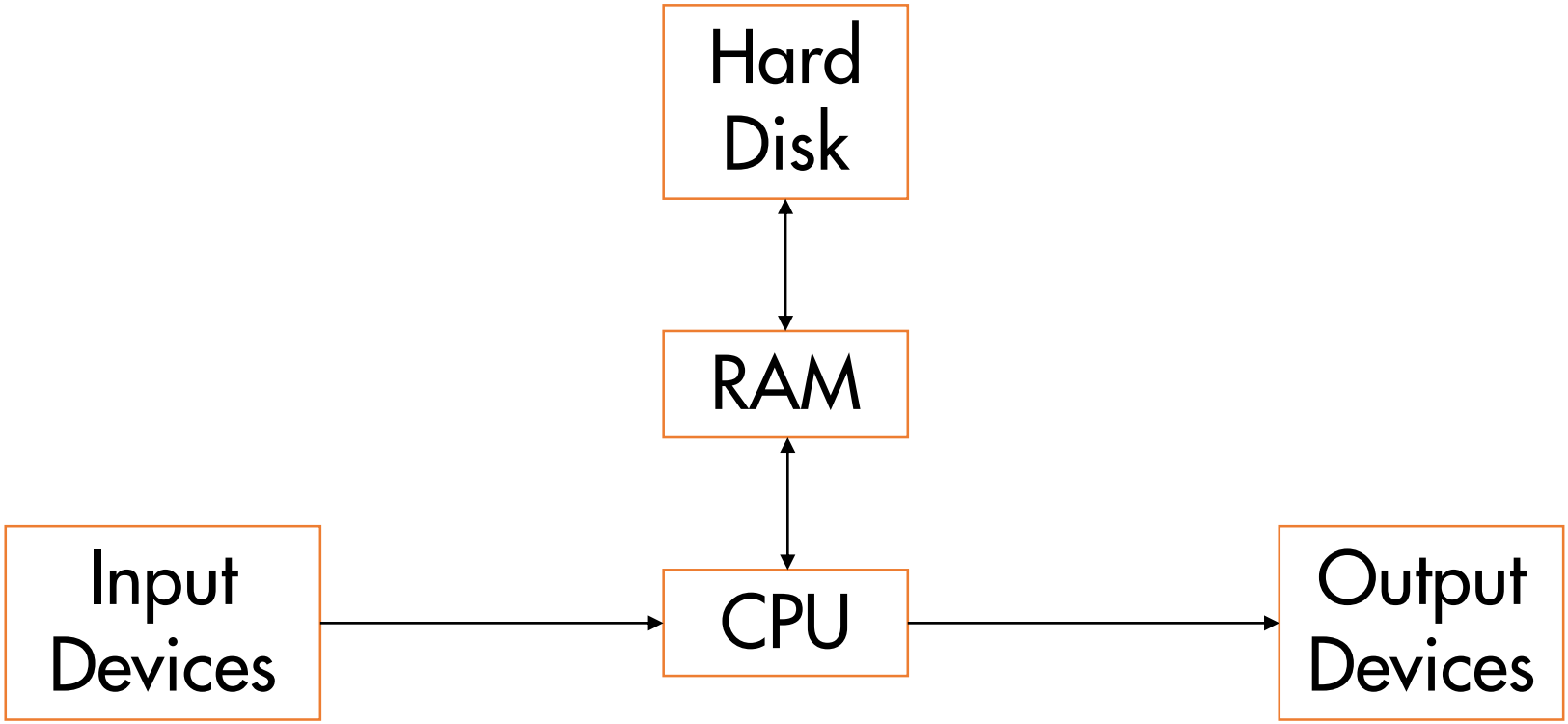
Mac OS and Linux users may already have Python

- However, do not use Python 2.x
- Type `python --version` in Terminal to check which version you have

Make sure to use **Python 3.x** (latest: **3.5.1** as of March 2016)

http://python.readthedocs.org/en/latest/using/windows.html

Introduction to computers



Interactive mode

In Command Prompt/Terminal: type `python` to enter Python's interactive mode

In IDLE: IDLE automatically launches in interactive mode

Expressions are combinations of values, variables, and operators

Expressions to try in interactive mode (use Python as a calculator!):

`2 + 2`	`2 ** 3`
`3.6 / 3`	`7 - 9 * 8`
`3.6 // 3`	`(7 - 9) * 8`
`17 % 4`	`9 + 2 // 3`

What do you think `//`, `%`, and `**` do?

Each expression in the table above **returns** a **value**

Two main types of numbers: **integers** and **floats** (floating-point numbers)

- Integers: 4, 99, -2, 0
  - Don't use commas for numbers (like 100000)
- Floating-point numbers: 3.3, 3.33, 7.0, -2.3, 7e9

*Notes on floating-point

*"On a typical machine running Python, there are **53 bits of precision available** for a Python float…" – Python Documentation (emphasis added)*

When you print floating-point numbers in Python, the output can be misleading because a) Python uses binary fractions (base 2 instead of base 10) to store floats and b) there are only a limited number of bits available.

From the documentation:

If Python were to print the true decimal value of the binary approximation stored for 0.1, it would display

```
>>> 0.1
0.1000000000000000055511151231257827021181583404541015625
```

This is a lot of digits, so Python displays a rounded number instead.

```
>>> 0.1
0.1
```

Try adding 0.1 to 0.2 and see what happens. Compare what happens when you add together ten copies of 0.1 and when you just multiply 0.1 by 10.

Other languages have this problem, so just keep this in mind when dealing with precise numbers. Use integers when you only need to deal with integers!

First program (script mode)

In IDLE, go to File > New File to begin writing a new script. Alternatively, open up your favorite text editor (Sublime Text, Notepad, Notepad++, Brackets, Atom, emacs, vim…not Microsoft Word!)

Type the following line.

```
print("Hello world!")
```

Save this somewhere, like your Desktop, with a sensible name, like "hello.py". Then, run your program (in IDLE, go to the Run menu at the top).

`print()` is a **function.**

`"Hello world!"` is a **string** that is passed to the `print()` function as a **parameter** (more on functions later).

Strings

Strings are basically sequences of characters. They are surrounded by straight quotation marks (single or double, as long as they match on both sides of the string). Do not use curly quotes (which might inadvertently happen if you try to copy and paste someone else's code).

Type a string in interactive mode, and that string will be output to the screen.

Examples of strings:

```
"Hello"
"test string"
"12345"
"testing"
"2 + 2"
"ACTGGTCGATCGAT"
```

Variables

If you want to hold on to data, **assign** it to a **variable.** Assignments are examples of **statements** (other statements coming up: if statements, return statements).

Examples:

```
pi = 3.14159
```

```
greeting = "Bonjour."
```

Variable **identifiers** (names) are often lowercase and derived from words (typically from English), but you can use almost any sequence of letters, numbers, and underscores, as long as the sequence **begins with a letter** and is **not a reserved keyword** (a word reserved for the Python language).

Not valid names:

2chainz

#winning

space bar (though "space" and "bar" are each individually valid)

Variables and memory

Basically, values are stored in RAM (which serves as **main memory** or **primary memory**). There are many locations in RAM at which to save values, and each location has a **memory address**. A variable name is essentially a label for an address. When you reassign a variable, you are changing the address that the "label" is referring to.

Python takes care of memory management for our needs, so we are likely not going to talk about manual memory allocation in this course (specifically, Python uses a garbage collector to reclaim memory).

Relevant links:

https://en.wikipedia.org/wiki/Computer_data_storage#Primary_storage
https://en.wikipedia.org/wiki/Memory_address

Reserved keywords

`False`	`class`	`finally`	`is`	`return`
`None`	`continue`	`for`	`lambda`	`try`
`True`	`def`	`from`	`nonlocal`	`while`
`and`	`del`	`global`	`not`	`with`
`as`	`elif`	`if`	`or`	`yield`
`assert`	`else`	`import`	`pass`	
`break`	`except`	`in`	`raise`	

Don't try to use any of these for variable names!

Variables cont.

You can reassign values to variables you have already assigned values to.

Example:

```
number = 2
print(number)
number = 7 * 7
print(number)
number = "This is a string."
print(number)
```

Variable updating

You can update variables by adding a value to the value already stored by the variable.

Example:

```
x = 4
print(x)
x = x + 2
print(x)
x += 2
print(x)
```

Instead of adding, you can also subtract, multiply, divide, and even use the modulus operator.

String concatenation

The + operator is useful for more than adding numbers; you can also use it to **concatenate** (combine) strings, creating a new string.

```
string_one = "Hello "
string_two = "world!"
print(string_one + string_two)
```

You cannot "subtract" a string from another string the way you can concatenate strings together.

Question: What happens if you try to "multiply" a string by an integer? What about by a floating-point number?

Comments

Use the pound sign (#) to mark off the rest of the line as a comment that will not affect the running of the program. This is useful for leaving notes to yourself or for disabling code while debugging.

Example:

```
number = 3
# number = 4
print(number)
# TODO: add new features
```

Commenting code with notes/explanations is very important! You might leave a project for a few months, then come back to it and not remember why you wrote the original code that way. Commenting is also important if you work on a team where other people may have to build upon your code but have not seen it before and need to get up to speed quickly.

Getting user input

You can prompt the user to type in something, like his/her name, and save the value that is given. This is useful for numerous things; one possible use is for text-based games.

Use the `input()` function; like `print()`, `input()` accepts a string **parameter** (or **argument**)

```
name = input("Enter your name: ")
print("Hello " + name + "!")
```

The string passed as a parameter to in `input()` is the prompt that will be displayed to the user.

Try to come up with some creative ideas for using user input!

Thanks for coming to Week 1!

Feedback is appreciated!