

# Programming Basics

Kelly Rivers and Stephanie Rosenthal

15-110 Fall 2019

# Last Time: Algorithms

A **precise rule (or set of rules)** specifying how to solve some problem.  
It should specify what is needed (input) to solve the problem, and what the solution looks like (output)

A good algorithm should  
**produce the correct outputs** for any set of legal inputs.  
**execute efficiently** (few steps when possible) and be specific.  
designed in a way that others will be able to **understand it and modify it**.

# Algorithms vs Programming

Computers only do what we tell them to do

We write programs to tell the computer what to do

Good algorithms are the same irrespective of language

We can use the same algorithm over and over

We translate it into different programming languages

Program structure, words, and syntax will be different

# Programming Languages

C

C++

C#

Java

JavaScript

Matlab

**Python**

...

We choose Python for algorithm implementation  
The concepts in the class are valid for any language

# Python

Interpretable: executed line by line

Values readability and simplicity

Has a huge standard library and many additional packages with useful code

Very popular for fast coding exercises and data analytics

Not good for code that has specific time or space requirements

# Numbers: Python as a Calculator

`+` : addition

`-` : subtraction

`*` : multiplication

`/` : division

`//` : division rounding down to nearest whole number

`%` : remainder left over when dividing and rounding down (modulo)

`()` : use parentheses to tell the computer what to execute first

`**` : power ( $2^{**}3 = 8$ )

**Integers**: whole numbers

**Float**: real numbers (with decimal places)

# Strings

## Valid Strings

`"hello"`

`'hello'`

`'hello"`  (type of quotation marks must match)

## Concatenation

`"A"+"B"+"C" -> "ABC"`

`"1"+"2" -> "12"`

`"1"+2`  `TypeError: can't add a string to an integer or float`

# Booleans – True/False

True

False

True == False -> False

1 == 2 -> False


1 == 1 -> True

'H' != 'h' -> True

1 < 2 -> True

2 > 2 -> False

1 < "2"  TypeError: can't compare an integer to a string

True == false  NameError: no name false  
(must capitalize False and True)



# Types

Python keeps track of the type of data

Boolean (**bool**) – True/False

Integer (**int**) – whole numbers

Floating point (**float**) – real numbers with decimal points (including .0)

String (**str**) – sets of characters enclosed in "..." or '...'

<b>type</b> (5) -> int	type("H") -> str	type(1>2) -> bool
type(5.0) -> float	type("hello") -> str	type(15-110) -> int
type(5/3) -> float	type("") -> str	type("15-110") -> str
type(5//3) -> int	type(True) -> bool	type(4-2.0) -> float

# Python Files vs Interpreter

The **interpreter (or shell)** allows you to type one line of code at a time.

If you have multiple lines of code that you want to run or if you don't want to type each line of code every time it needs to run, use a file.

**Python files** end in .py

When you run a python file, the interpreter runs it line by line (for now)

Python files do not output the answers to each line

you need to explicitly print out values if you want to see the answer


# Print Statements

Output the value of what is in parentheses. Use a + or , to concatenate two strings

```
print(4 - 2) -> 2
print(4.0 - 2.0) -> 2.0
print(1 < 2) -> True
print(15-110) -> -95
print("15-110") -> 15-110
print(5/3) -> 1.66666666667
print(5+3) -> 8
print("5"+"3") -> 53
```

# Print Statements

Output the value of what is in parentheses. Use a + or , to concatenate two strings

```
print("My favorite class is 15-110") -> My favorite class is 15-110
print("My favorite class is "+ 15-110)  TypeError: can't add a str and
int
print("My favorite class is ", 15-110) -> My favorite class is -95
print("My favorite class is ", str(15-110)) -> My favorite class is -95
print("My favorite class is "+"15-110") -> My favorite class is 15-110
print("My favorite class is ", "15-110") -> My favorite class is 15-110
```

# Syntax and Spacing


Python cares about spaces and tabs and capitalization


## Correct:


```
print("Hello")  
print ("hello")
```

## Incorrect:

```
    print("Hello")  
        print("Hello")  
p r i n t("Hello")
```

 IndentationError: unexpected indent

 IndentationError: unexpected indent

 SyntaxError: invalid syntax

# Comments

Comments help

- make your code **more readable**

- allow you to make a **line of code not run** without deleting it

Anything on a line after a **#** is a comment

Anything in triple quotes is a comment **""" ... """**

Anything not in comments will be treated like code

# Comments

## Examples


```
print("Hello") #this prints Hello
```

```
# now I will print it lowercase
```

```
print ("hello")
```

```
''' If I write three quotes
```

```
Then my sentence will not be run'''
```

 There will be an error on this line because my text isn't in a comment

# Variables

Variables allow you to **save the value of computation** to use later in a program

We set or assign a variable to a literal or another variable using the **=** sign

```
VariableToSet = ValueToSet
```

Note: Unlike math, only the variable to set can go to the left of the **=**

Rules:

Variables cannot start with a number and cannot ever use a special character other than `_`

Variables can use Upper and lower case letters, numbers, and underscores

Spell the variable name exactly the same each time you use it!



# Variables

```
x = 5
```

```
y = x+5 #y=10
```

```
z = x+y #z=15
```

```
a = x<y
```

```
print(z) #this should print 15
```

```
print(a) #this should print True
```

# Variables

```
print(hello)
```

```
#NameError. We have not set a variable hello
```

```
#hello = ...
```

```
5 = x
```

```
#NameError. The variable getting assigned goes on left
```

```
#5 is not a valid variable
```

```
d - 1 = 5
```

```
#SyntaxError: can't assign to operator.
```

```
#Unlike in math, only d can be on the left!
```

# Testing and Debugging

An error in a program is called a **bug**. Removing bugs is called **debugging**.

We must test our code to ensure that there are no bugs

good algorithms produce valid output on all valid input

# Testing and Debugging

Testing requires that we

- 1) think about **different possible inputs** and try them out
- 2) correct code when a valid input produces an **error** or the **wrong answer**

Use **print statements** to check whether the variables in our code are set to values we expect

If an error occurs, look at the **error message** to find the line of code to fix and the type of error that occurred (Indentation vs Syntax vs TypeError)

# Announcements

- Get Python3 and Pyzo running on your computer. Try out the code from class
- Waitlist – Keep coming to class. We'll work with you to find space
- Homework 1 Check-in due on Monday 9/2 noon
- Homework 1 is due Monday at 9/9 noon!

You should already be able to do some of the problems from each!