Week 2 Agenda

- Week 2 Introduction to Python
  - Python 2 vs 3
  - Installing Python
  - IDLE
  - Other IDEs
  - Variables
  - Printing in Python
  - Math Operations
- Chapter 1 and 2 in Python Crash Course: A Hands-on, Project-Based Introduction to Programming

Python 2 vs Python 3

What are the differences?

Short version: Python 2.x is legacy, Python 3.x is the present and future of the language

Python 3.0 was released in 2008. The final 2.x version 2.7 release came out in mid-2010, with a statement of extended support for this end-of-life release. The 2.x branch will see no new major releases after that. 3.x is under active development and has already seen over five years of stable releases, including version 3.3 in 2012, 3.4 in 2014, 3.5 in 2015, and 3.6 in 2016. This means that all recent standard library improvements, for example, are only available by default in Python 3.x.

Python 2 vs Python 3 continued...

Guido van Rossum (the original creator of the Python language) decided to clean up Python 2.x properly, with less regard for backwards compatibility than is the case for new releases in the 2.x range. The most drastic improvement is the better Unicode support (with all text strings being Unicode by default) as well as saner bytes/Unicode separation.

Besides, several aspects of the core language (such as print and exec being statements, integers using floor division) have been adjusted to be easier for newcomers to learn and to be more consistent with the rest of the language, and old cruft has been removed (for example, all classes are now new-style, "range()" returns a memory efficient iterable, not a list as in 2.x).

The What's New in Python 3.0 document provides a good overview of the major language changes and likely sources of incompatibility with existing Python 2.x code. Nick Coghlan (one of the CPython core developers) has also created a relatively extensive FAQ regarding the transition.

However, the broader Python ecosystem has amassed a significant amount of quality software over the years. The downside of breaking backwards compatibility in 3.x is that some of that software (especially in-house software in companies) still doesn't work on 3.x yet.

Deciding Which Python Version to Use

Deciding which version of Python to use will depend on your specific circumstances. Mac and Linux/Unix operating systems general come pre installed with Python. For ease of use you may wish to use the preinstalled version. The Python community however, recommends that new developers start with Python 3.

Python 2 has better library support than Python 3 so depending on which libraries you need to use for your program may also influence the version of Python you use.

Additional Documentation regarding Python 2-3 differences

http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html

Checking for Python Installation

Before Installing Python it is advisable to determine if Python has been installed on your PC. From the command prompt simply type “Python”. You should see results similar to below:

```
C:\Users\mikee>python

Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>
```

Installing Python

See textbook “Python Crash Course” Chapter 1 for detailed instructions.

1. Download the Python installer
2. Run Python Installer
3. Add the Python Home directory to your Path
4. Test Python installation by running the “Python” command from the command line.

Python IDEs

An IDE is an Integrated Development Environment and make writing programs more effective by using certain language specific features such as syntax highlighting and completion.

Python comes with a built in IDE called IDLE. IDLE is great for short programs and for testing purposes however other IDEs such as ATOM and NOTEPAD++ are often more efficient for larger programs.

https://atom.io/

https://notepad-plus-plus.org/

IDLE

IDLE is Python’s Integrated Development and Learning Environment.

IDLE has the following features:

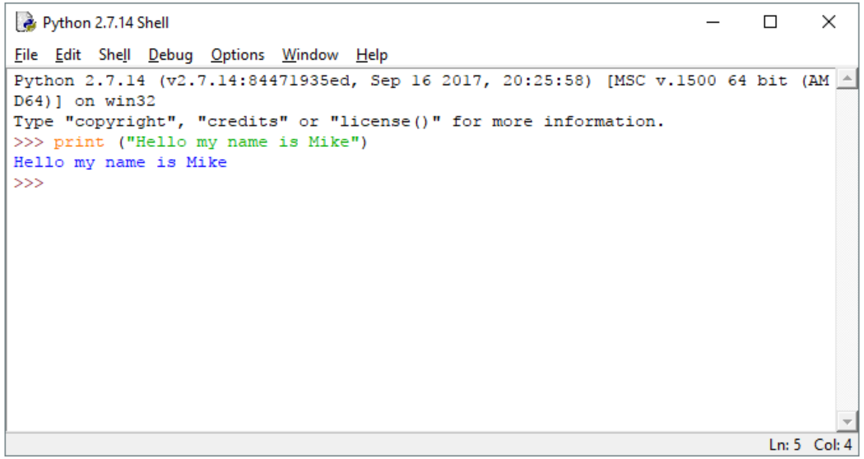
- coded in 100% pure Python, using the **tkinter** GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

NOTE: Additional Information on IDLE usage can be obtained here: https://docs.python.org/2/tutorial/interpreter.html

First Python Program using IDLE

Open IDLE

At the IDLE Command Prompt type ***print ("Hello my name is <your name>")*** and hit the Enter key



```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print ("Hello my name is Mike")
Hello my name is Mike
>>>
```

First Python Program using Atom

Open the Atom IDE
Create a new File
Type **print (“Hello my name is <your name>”)** on the first line

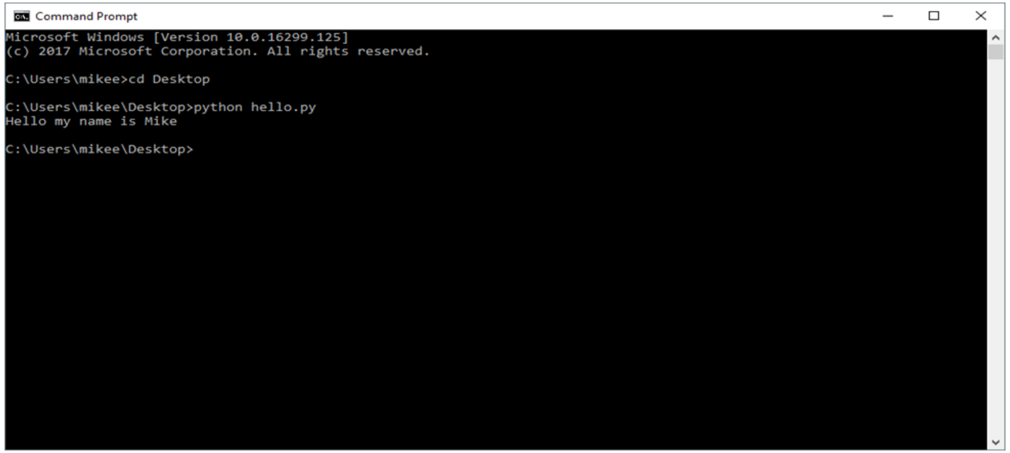
- Notice that once you type the first parenthesis or double quote that Atom will add another one for you. This is one of the conveniences of using a text editor such as Atom. This is an example of Atom’s autocomplete capability.

Save the program as “hello.py”

- Notice that once you save the file with a “.py” extension that Atom will automatically highlight in different colors the syntax of your program. This is an example of Atom’s syntax highlighting capability.

Running your Atom program

- Open a command prompt
- Navigate to the location that you saved your program created in Atom
- Type, “**python hello.py**” to run your python program. By typing python before the name of the program you are calling the python executable which is needed to run the program.



```
Microsoft Windows [Version 10.0.16299.125]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\mikee>cd Desktop

C:\Users\mikee\Desktop>python hello.py
Hello my name is Mike

C:\Users\mikee\Desktop>
```

Analyze the hello.py Program

Let’s take a closer look at our hello.py program:

```
        print ("Hello my name is Mike")
```

When you run this code, you should see this output:

```
    Hello my name is Mike
```

When you run the file hello.py, the ending.py indicates that the file is a Python program. Your editor then runs the file through the Python interpreter, which reads through the program and determines what each word in the program means. For example, when the interpreter sees the word print, it prints to the screen whatever is inside the parentheses.

As you write your programs, your editor highlights different parts of your program in different ways. For example, it recognizes that print is the name of a function and displays that word in blue. It recognizes that "Hello my name is Mike" is not Python code and displays that phrase in a different color. This feature is called syntax highlighting and is quite useful as you start to write your own programs.

Python Variables

A variable is a name (identifier) that is associated with a value and is a way of referring to a memory location used by a computer program. A variable is a symbolic name for this physical memory location. This memory location contains values, like numbers, text or more complicated types.

A variable can be seen as a container values. While the program is running, variables are accessed and sometimes changed, i.e. a new value will be assigned to the variable.

One of the main differences between Python and strongly-typed languages like C, C++ or Java is the way it deals with types. In strongly-typed languages every variable must have a unique data type. E.g. if a variable is of type integer, solely integers can be saved in the variable. In Java or C, every variable has to be declared before it can be used. Declaring a variable means binding it to a data type.

Python just like all languages has multiple types of variables that can be used including numbers (integers and floats), strings, and lists.

Integers

Integers

Integer numbers (or simply, *integers*) are counting numbers like 1, 2, 3, but also include 0 and negative numbers. The following are examples of data that are expressed as integers:

- Number of people in a room
- Personal or team score in a game
- Course number
- Date in a month
- Temperature (in terms of number of degrees)

Floats

Floating-point numbers (or simply, *floats*) are numbers that have a decimal point in them. The following are examples of data that are expressed as floating-point numbers:

- Grade point average
- Price of something
- Percentages
- Irrational numbers, like pi

Strings

Strings (also called *text*) are any sequences of characters. Examples of data that are expressed as strings include the following:

- Name
- Address
- Course name
- Title of a book, song, or movie
- Sentence
- Name of a file on a computer

Booleans

Booleans are a type of data that can only have one of two values: True or False. Booleans are named after an English mathematician named George Boole, who created an entire field of logic based around these two-state data items. The following are some examples of data that can be expressed as Booleans:

- The state of a light switch: True for on, False for off
- Inside or outside: True for inside, False for outside
- Whether someone is alive or not: True for alive, False for dead
- If someone is listening: True for listening, False for not listening

Python Variables

Declaration of variables is not required in Python. If there is need of a variable, you think of a name and start using it as a variable.

Another remarkable aspect of Python: Not only the value of a variable may change during program execution but the type as well. You can assign an integer value to a variable, use it as an integer for a while and then assign a string to the variable.

In the following line of code, we assign the value 42 to a variable:

```
i = 42
```

The equal "=" sign in the assignment shouldn't be seen as "is equal to". It should be "read" or interpreted as "is set to", meaning in our example "the variable **i** is set to 42". Now we will increase the value of this variable by 1:

Variable Overview

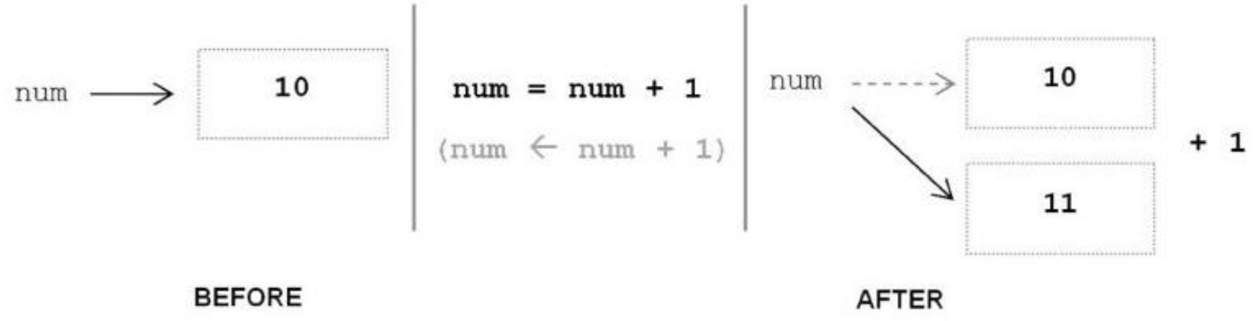
A variable can be assigned different values during a program's execution—hence, the name "variable." Wherever a variable appears in a program (except on the left-hand side of an assignment statement), it is the value associated with the variable that is used, and not the variable's name,

num + 1 → 10 + 1 → 11

Variables are assigned values by use of the assignment operator, =,
num = 10 num = num + 1

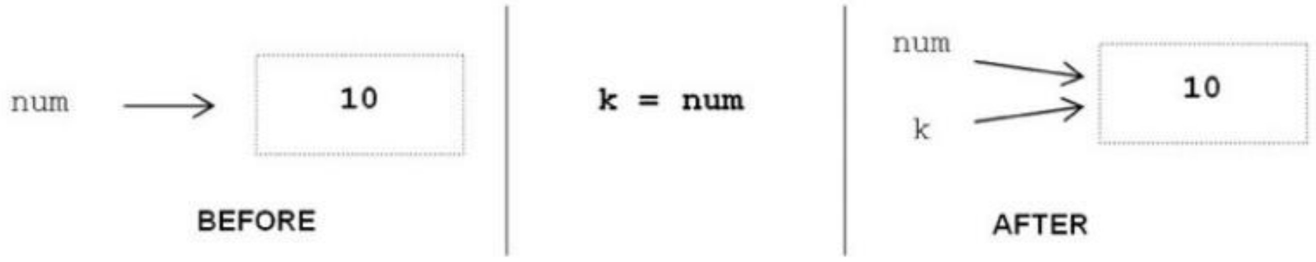
Variable Assignment Statements

Assignment statements often look wrong to novice programmers. Mathematically, num = num + 1 does not make sense. In computing, however, it is used to increment the value of a given variable by one. It is more appropriate, therefore, to think of the = symbol as an arrow symbol,



Variable Assignment Statements

When thought of this way, it makes clear that the right side of an assignment is evaluated first, then the result is assigned to the variable on the left. An arrow symbol is not used simply because there is no such character on a standard computer keyboard. Variables may also be assigned to the value of another variable (or expression, discussed below) as depicted below.



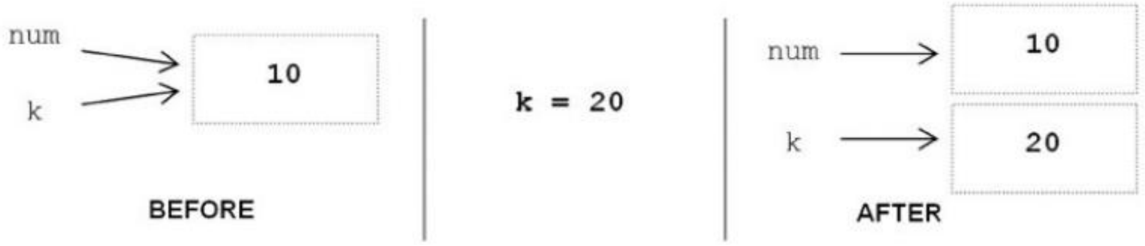
Variable Assignment Statements Continued...

Variables num and k are both associated with the same literal value 10 in memory. One way to see this is by use of built-in function id,

```
>>> id(num)                    >>> id(k)
505494040                     505494040
```

ID Function

The id function produces a unique number identifying a specific value (object) in memory. Since variables are meant to be distinct, it would appear that this sharing of values would cause problems. Specifically, if the value of num changed, would variable k change along with it? This cannot happen in this case because the variables refer to integer values, and integer values are immutable. An immutable value is a value that cannot be changed. Thus, both will continue to refer to the same value until one (or both) of them is reassigned, as depicted below:



Variable References

If no other variable references the memory location of the original value, the memory location is deallocated (that is, it is made available for reuse).

Finally, in Python the same variable can be associated with values of different type during program execution, as indicated below.

var = 12 integer
var = 12.45 float
var = 'Hello' string

Working with Python Variables

Let's try using a variable in hello.py. Add a new line at the beginning of the file, and modify the second line:

```
message = "Hello my name is Mike"
print(message)
```

Run this program to see what happens. You should see the same output you saw previously:

```
Hello my name is Mike
```

We've added a variable named **message**. Every variable holds a value, which is the information associated with that variable. In this case the value is the text "Hello my name is Mike".

Adding a variable makes a little more work for the Python interpreter. When it processes the first line, it associates the text "Hello my name is Mike" with the variable message. When it reaches the second line, it prints the value associated with message to the screen.

Modifying our Program

Let's expand on this program by modifying hello.py to print a second message. Add a blank line to hello.py, and then add two new lines of code:

```
message = "Hello my name is <your name>"
print(message)
message = "Hello Python Crash Course world!"
print(message)
```

When you run the program you should see the following output:

```
Hello Python world!
Hello Python Crash Course world!
```

Identifiers

An identifier is a sequence of one or more characters used to provide a name for a given program element. Variable names line, num_credits, and gpa are each identifiers. Python is case sensitive, thus, Line is different from line. Identifiers may contain letters and digits, but cannot begin with a digit. Additional rules for defining variable names are provided on the following slide.

Python Variable Naming Rules

- Variable names can contain only letters, numbers, and underscores. They can start with a letter or an underscore, but not with a number. For instance, you can call a variable `message_1` but not `1_message`.
- Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. For example, `greeting_message` works, but `greeting message` will cause errors.
- Avoid using Python keywords and function names as variable names; that is, do not use words that Python has reserved for a particular programmatic purpose, such as the word print. (See "Python Keywords and Built-in Functions" on page 489.)
- Variable names should be short but descriptive. For example, `name` is better than `n`, `student_name` is better than `s_n`, and `name_length` is better than `length_of_persons_name`.
- Be careful when using the lowercase letter `l` and the uppercase letter `O` because they could be confused with the numbers 1 and 0.

Print Statements

The print statement is very "general purpose." You ask it to print something and it prints whatever you ask it to print into the Shell window. The general print statement looks like this:

```
print <whatever you want to see>
```

Here are some examples in the Shell:

```
 >>> eggsInOmlette = 3
>>> print eggsInOmlette
3
>>> knowsHowToCook = True
>>> print knowsHowToCook
True
>>>
```

Assignment and Print Statements

Here are more examples of assignment statements and print statements, using all four types of data:

```
>>> numberInADozen = 12
>>> print 'There are', numberInADozen, 'items in a dozen'
There are 12 items in a dozen
>>> learningPython = True
>>> print 'It is', learningPython, 'that I am learning Python'
It is True that I am learning Python
>>> priceOfCandy = 1.99
>>> print 'My candy costs', priceOfCandy
My candy costs 1.99
>>> myFullName = 'Irv Kalb'
>>> print 'My full name is', myFullName
My full name is Irv Kalb
>>>
```

Print Statement in Python 2 vs Python 3

In Python 3, the print statement has a different form (syntax). In Python 3, the print statement must have parentheses around the item(s) that you want to print, as follows: print (<itemi>, <item2>, ...) This is perhaps the most noticeable difference between Python 2 and Python 3. If you see code elsewhere written in Python 3 that uses parentheses in print statements, you can often modify these statements to work in Python 2 by removing the outermost set of parentheses.

Math Operators

Now let's move on to some simple math for use in assignment statements. Python and all computer languages include the following standard set of math operators:

- + Add
- − Subtract
- / Divide
- * Multiply
- ** Raise to the power of
- % Modulo (also known as remainder)
- ( ) Grouping (we'll come back to this)

Simple Math

Let's try some very simple math. For demonstration purposes, I'll just use variables named x and y. In the Shell, try the following:

```
>>> x=9
>>> y=6
>>> print x + y
15
>>> print x - y
3
>>> print x * y
54
>>> print x / y
1
>>>
```

Division in Python 2 vs Python 3

In Python 3, the division operator works differently. In Python 3, if you use the slash to do a division, you will always get a floating-point answer. If you want to do an explicit integer division, you must use a new operator; two slashes, for example:

```
forcedIntegerAnswer = integer1 // integer2
```

Division with Floats

As humans, we represent integers using base 10 (digits from 0 to 9). Computers represent integers using base 2 (using only ones and zeroes). But there is an exact mapping between the two bases. For every base 10 number, there is an exactly equivalent base 2 number. However, because of the way that computers represent floating-point numbers, this is not the case for floating-point numbers. There is no such mapping between base 10 fractions and base 2 fractions. When representing floating-point fractional numbers, there is often some small amount of "rounding"; that is, floating-point fractional numbers are a close approximation of the intended number. For example, if we attempt to divide 5.0 by 9.0, we see this:

```
>>> print 5.0 / 9.00.555555555556
```

The decimal values goes on forever, but when represented as a float, the value gets rounded off.

Order of Operations

Math operations in programming follow the same order of operations that we learn in elementary school. Using a common acronym PEMDAS we see the following order of operation:

1. Exponents
2. Multiplication
3. Division
4. Addition
5. Subtraction

References

The Basics of Compiled Languages, Interpreted Languages, and Just-in-Time Compiliers, https://www.upwork.com/hiring/development/the-basics-of-compiled-languages-interpreted-languages-and-just-in-time-compilers/

The End

You may close this Window and return to the course.