

Special Topics in Info Systems Management: **Python**

Fall 2014, with Nicholas Miceli



NYU

Class Overview



- Learn programming fundamentals, from the basics
- Learn Python from the basics
- Learn powerful & practical skills
- Build something cool!

Highlights from Syllabus



- Every week, last week's lab is due and next week's is assigned
- 80% of grade is labs
 - What to do to get full credit is clearly explained with each lab
- Strict deadlines & attendance policies
- Office hours available by request, via Google Hangouts

What is programming?



- Teaching a computer how to do a new task
- Programming languages offer the ability to speak to the machine in a way we both understand
- Programming is **logical, objective, and deterministic**
 - Input A will always produce output B

Important to note:



- **Everything** we write in code has a very logical purpose.
- Don't just copy+paste something that works. If you don't understand *why* it works, ask
- Try, try, try! Often, you learn more from playing than from just listening
 - In other words, don't just take my word for it!

- Code is not always "right or wrong"
 - It will either produce correct or incorrect output, but there's more to it
- Software will be modified, shared, reused
 - Is it easy to read? Even for someone unfamiliar?
 - If a bug appears, how hard would it be to fix?
 - Is it easy to make future improvements to?
- Two different programs may both give correct output, but with very different speed
 - Efficiency - very important

PYTHON

THIS IS PLAGIARISM.
YOU CAN'T JUST "IMPORT ESSAY."



JAVA

I'M TWO PAGES IN AND I STILL
HAVE NO IDEA WHAT YOU'RE SAYING.



C++

I ASKED FOR ONE COPY,
NOT FOUR HUNDRED.



UNIX SHELL

I DON'T HAVE PERMISSION TO
READ THIS.



ASSEMBLY

DID YOU REALLY HAVE TO REDEFINE EVERY
WORD IN THE ENGLISH LANGUAGE?



C

THIS IS GREAT, BUT YOU FORGOT TO ADD
A NULL TERMINATOR. NOW I'M JUST READING
GARBAGE.



LATEX

YOUR PAPER MAKES NO GODDAMN SENSE,
BUT IT'S THE MOST BEAUTIFUL THING
I HAVE EVER Laid EYES ON.



HTML

THIS IS A FLOWER POT.



What is Python?



- High-level Programming language
- Focuses on being readable, maintainable, and requiring fewer lines of code to achieve the same result as in other languages
- For scripting or production applications
- Supports many different "paradigms"
- Highly used professionally, such as at Google!

Why should I care?



- Super useful for anyone in tech, math, science, etc.
- Practical use in software engineering
- Great for rapid prototyping
- Great for simple scripts in many software-unrelated fields
- Powerful companion alongside other languages (i.e. Java)

Python Interpreter



- Run code instantly, no compiling needed!
- Available installed, or online
 - <http://ideone.com/> (Supports Python2 and Python3)
 - <http://repl.it/languages/Python> (Only supports Python2)
 - <http://python.codepad.org/> (Only supports Python2)

```
ngmiceli1:~$ python3
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 10
>>> x * 3
30
>>> print("So cool!")
So cool!
>>> "Hello".swapcase()
'hELLO'
>>> █
```

"Hello, world"

- First program! Open up interpreter and write:

```
print("Hello, World!")
```

The Zen of Python



In the same interpreter, write:

```
import this
```

Then be sure to read the output!

(Note: this is not a feature of the language, just an easter egg)

Hello World in Python vs Java



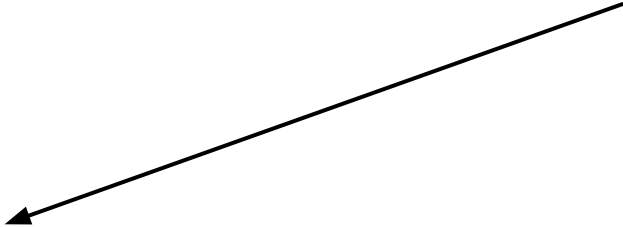
Python:

```
print("Hello, World!")
```

Java:

```
public class Foo {  
    public static void main(String args[]) {  
        System.out.println(  
            "Hello, World!");  
    }  
}
```

What the heck does all that mean??

A black arrow originates from the text "What the heck does all that mean??" and points diagonally down and to the left, ending at the opening curly brace of the Java code block.

Python is very readable



- Reads like you would describe code in English
- Without any prior knowledge of Python, can you figure out what the following scripts do?

What will this print?



```
1 list_of_ingredients = ['pasta', 'tomato', 'beef', 'cheese']
2
3 if 'chicken' in list_of_ingredients or 'beef' in list_of_ingredients:
4     print("This meal is not vegetarian")
5 else:
6     print("This meal is vegetarian!")
7
8 if 'peanut' in list_of_ingredients:
9     print("Warning: contains nuts!")
10
11 if 'cheese' in list_of_ingredients:
12     print("Remember to take your lactose pill")
13
14 if 'bacon' not in list_of_ingredients:
15     print("This is not delicious enough")
```

</> source code

```
1  from random import randint
2
3  secret_num = randint(0,10)
4  guessed_correctly = False
5  while not guessed_correctly:
6      guess = input("Guess a number between 0 and 10")
7      guess_as_int = int(guess)
8      if guess_as_int == secret_num:
9          print("Correct!")
10         guessed_correctly = True
11     elif guess_as_int < secret_num:
12         print("Think higher...")
13     else:
14         print("Think lower...")
15
16  print("Thanks for playing!")
```


- In addition to Python3 interpreter...
- PyCharm IDE
 - IDE = Integrated Development Environment
 - Basically, makes writing code easier / faster
 - Allows us to write, organize, test, and execute code all from one application

Scripts vs Applications



- Old definition
 - Scripts = interpreted, Applications = Compiled
 - Not really meaningful anymore
- More common usage
 - Scripts = quick, dirty, solution to a task. You are the only consumer
 - Application = engineered, maintainable solution to a problem.
Hopefully, someone or something else is consuming your application
- Python can easily be used for either!

4 Common Ways to Run Python



1. Directly from the interpreter / command-line
 - Mostly used to quickly try something, or for *super basic* tasks
 - This is what we did with our Hello World statement
2. Invoke the interpreter on a Python file
 - Most common way for how **scripts** are run.
3. Directly from your IDE
 - For testing your code as you're writing it
4. As a compiled executable
 - For a finished application, so users can run it without needing to know about interpreters or IDEs

Run a Python file from the command line



- Open up any simple text editor (Notepad, TextEdit)
- Write `print("Hello, world!")`
- Save the file as "hello.py". Remember where you saved it!
- Open interpreter and write:

```
python hello.py
```

- You may need to specify the path to the file:

```
python somefolder/otherfolder/hello.py
```

Installing Components



- <https://www.python.org/download>
 - Make sure to get version 3+
- <http://www.jetbrains.com/pycharm/download/>
 - Community edition is certainly sufficient for our needs
 - Make sure the interpreter is installed *before* PyCharm for easier setup

Both of these are supported on Windows, Mac, and Linux.

Using PyCharm



- Start PyCharm. Click "Create New Project"
- Name your project - I named mine "Class1"
- Be sure to set the interpreter to Python3. Click OK

Creating our project



- Right-click on the "Class1" project and go to "New"
- Select "Python File"
- Name the file "Greetings"
- Add our print statement to a clean line at the bottom of the file. Your file should look something like:

```
__author__ = 'ngmiceli'  
print("Hello, world!")
```

Executing our code



- Under the top menu, click "Run", then "Run..."
- Choose your project (i.e. "Class1")
- Your output appears in the "Run Greetings" pane!
- You can use the green arrow in that pane to re-run your code. Try changing the text and re-running it to see the updated output

The PyCharm Interpreter



- You can launch an interpreter right from PyCharm.
- In the top menu, click "Tools", then "Run Python Console"
- The "Run Python Console" pane should now be open at the bottom.
- Test it out with another `print` statement.
- Works the same as the interpreter we ran in the command-line

- So far, the only value we've seen is `"Hello, World!"`
- Open up the interpreter (either in Eclipse or the command-line) and try out the following statements:
 - `"foo" + "bar"`
 - `100 + 20 / 5 * 2`
 - `1 / 9`
 - `True or False`
 - `True and False`

Data Types



- Integer (int)
 - `-10, 0, 50, 1000, 10000000000000000000`
- Float
 - `-5.123, 613.0, .11111111`
- String
 - `"Hello, World!", 'Hello, World!'`
- Boolean (bool)
 - `True, False`

Numeric Operations



- Python treats most numbers (int, float, etc) the same
- Standard operators
 - ()
 - + - / *
 - < <= > >= == (comparisons)
 - ** (exponent)
 - % (modulo - super useful)
- Follows order of operations
- Integers have no max size in Python3
- Floats are accurate to 15 decimal places

Boolean Operations



- Two boolean values: `True`, `False`
- Operators for Boolean values:
 - `and`
 - `or`
 - `not`
 - Follows "Boolean Logic"
- Try the following:
 - `5 < 4`
 - `5 < 4 or 5 < 10`
 - `5 < 4 and 5 < 10`
 - `not (5 < 4 and 5 < 10)`
 - `-3 < -2 < -1 and -1 > -2 > -3`
 - `True and True or False and True`

Boolean Logic Reference



and	T	F
T	T	F
F	F	F

True and True \rightarrow True
True and False \rightarrow False
False and True \rightarrow False
False and False \rightarrow False

or	T	F
T	T	T
F	T	F

True or True \rightarrow True
True or False \rightarrow True
False or True \rightarrow True
False or False \rightarrow False

not True \rightarrow False
not False \rightarrow True

- Strings are a collection of characters
 - What is a character? The detailed answer comes later, but for now think of it as 'K' or '5' or '深'
- Strings are always wrapped in quotes
 - Double quotes (") or single quotes (') mostly don't matter in Python.
 - Whatever you open a string with, you must use the same symbol to close the string
 - Prefer single-quotes whenever possible

The following are examples of strings:

```
'This is all just some text'
```

```
"This is all just some text"
```

```
'"Thank you", he said.'
```

```
"Don't try that again!"
```

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

The above print statement is **not** read as code to the interpreter. It's just a collection of characters.

Strings (cont.)



- If you use a quote to open and close the string, how do you put an apostrophe in the string?
 - You can use double-quotes to contain a string with an apostrophe, or you can "escape" it
- Escaping a character means to suppress meaning it might have and read it as a regular character
- Escape a special character by adding a backslash before it

`'This isn\'t "the end", but this is'`

`"This isn't \"the end\", but this is"`

Moar strings!



- If backslash means "escape the next character", how do you write a backslash?
 - Escape the backslash, of course!

'This prints with one slash: \\'

- Backslash can also be used to make special chars:
 - `\n` makes a new line
 - `\t` makes a tab

Multiline strings



- It is a "syntax error" to end a line without closing a string

```
"This code will not run
```

- To have multiple "lines" in your string, you can use `\n` or a multiline string.
 - Open and close them with three quotes: `'''` or `"""`

```
'''This is perfectly  
valid Python!'''
```

Operations on Strings



- Use + to append two strings together

`'My name is' + 'Nick Miceli'`

- Use `in`, `not in` to see if one string contains another
 - Results in a boolean

`'d b' in 'Good bye' and 'foo' in 'foobar'`

`'d b' not in 'Good bye'`

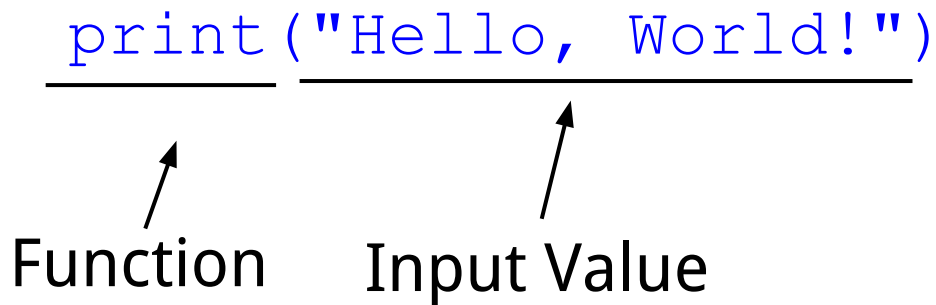
- Lots of other operators and power with Strings. We'll come back to them later

- A "comment" is any code on a line following a #
- As soon as the interpreter sees the beginning of a comment, it ignores the rest of the line
- Comments are for programmers to leave notes & documentation
- Remember, **strings are data**, comments are just notes. Strings *take up memory* in the application, comments *are ignored* by the application

Functions

- A function is an operation. You provide it certain inputs, and it provides an output

```
print ("Hello, World!")
```



Function Input Value

A little about Functions



- Every function has a definition, or "contract", that describes how it operates. This definition describes:
 - Exactly how many inputs it expects, and what types it expects for each
 - What type the output of the function is
- You "call" (execute) a function by providing the inputs (parameters) in parentheses:

```
someFunction(param1, param2, param3)
```

The "type" function



- There are a few functions "built-in" to Python.
 - So far, we've seen `print`
- The `type` function takes in 1 value of any type, and outputs what Type the provided value is.
 - `type(1)`
 - `type(1.5)`
 - `type("Hello")`

Comment example



```
# The following code is an example of the
#   usage of comments
print(10*500) # Prints 5000 to the screen
# print('This will not print')
print('This will print')
```

- A variable is a container which holds a value
- In Python *any* variable can hold *any* type of value
 - This is called **dynamic typing**
 - Java, by contrast, is **static typed**. You declare a given variable is for a certain type
- Create variables with the **assignment operator**, =

```
name = "Nick Miceli"
```

```
print("My name is " + name)
```

- *Just need to memorize this:*

This assigns the value 5 to a variable `x`

```
x = 5
```

This checks if the variable `x` is "equal to" 5

```
x == 5
```

Using Variables



- You can reference a variable anywhere you would normally put a value
 - The interpreter looks up the value of the variable at runtime and substitutes it into the statement

$x = 25 * 6$

$y = x / 2$

$z = x + y$

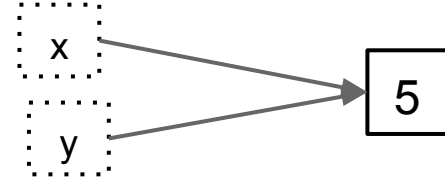
(What is the value of z here?)

Variables and memory

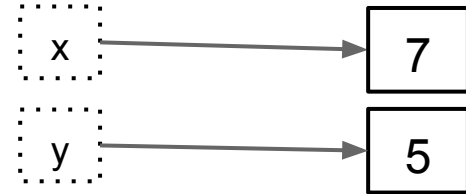
$x = 5$



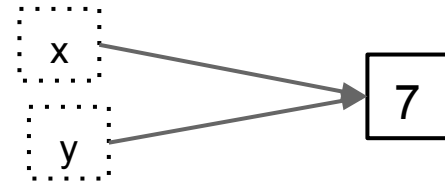
$y = x \dashrightarrow y = 5$



$x = 7$



$y = y + 2 \dashrightarrow y = 5 + 2$
 \downarrow
 $y = 7$



Working between types



Imagine the following code:

```
points = 5000  
message = 'Your points: ' + points  
print(message)
```

This will produce a "TypeError".

Python doesn't know how to use the + operator between a string and a number.

We would need to convert the number into a string!

Type conversion



Each type has a function used for conversion:

- `int()`
- `float()`
- `bool()`
- `str()`

Technically, these are "constructors". More about that in a future session!

Integer conversion



Ways to use the `int()` function:

- With a float parameter, drops the decimal point
 - Does **NOT** round.
- With a string parameter, tries to interpret the contents as a base-10 integer
- Not very useful with boolean or integer parameter...

`int(4.9)` # becomes 4 (not 5!)

`int('15')` # becomes 15

More Conversions



Float conversion works very similarly.

```
float("10.593")
```

And same with string conversion:

```
str(10.5)
```

In fact, you can use `str` on objects of many types to get human-readable representations

- `None` is a special keyword that can be used to represent the absence of a value.
- You can create a variable but give it no value:

```
x = None
```

- `None` has its own Type, called `NoneType`

```
type(None)
```

- Every function* technically returns (outputs) a value. If a function just "does something" without creating output, it returns `None`.

Statements and Syntax



What do the following statements do? Are they legal Python statements, or will they cause errors?

```
x = 5 + 10
```

```
x = 5 + y
```

```
x = 5 == y
```

```
x = y = 10
```

```
x = 10 = z
```

```
x + y = 10
```

Statements and Syntax



- Programming languages have strict, formulaic grammar.
- We call that grammar, "syntax"
- If a statement doesn't follow Python's grammar, it will result in a "SyntaxError"

Assignment Syntax



- Rule: Value goes on the right, what's being assigned that value goes on the left.

Valid:

```
x = 100
```

Invalid, because "100" can't be given a new value:

```
100 = x
```

- Values like `100` or `"hello"` are called "literals".
Literals cannot be assigned new values.

More built-in functions



Some more built-in functions:

- `max(number, number, ...)`
 - Takes in 2 or more numbers and returns the largest
- `min(number, number, ...)`
 - Takes in 2 or more numbers and returns the smallest
- `round(number)`
 - Takes in 1 number and rounds it to the nearest integer
- `len(string)`
 - Takes in 1 string and returns the number of characters in it

Statements within statements...



Let's walk through how the interpreter handles this:

```
x = round(min(1+2, max(0, 4, 1), 2.5)) - 1
```

```
x = round(min(1+2, 4, 2.5)) - 1
```

```
x = round(min(3, 4, 2.5)) - 1
```

```
x = round(2.5) - 1
```

```
x = 3 - 1
```

```
x = 2
```

Returning vs Printing

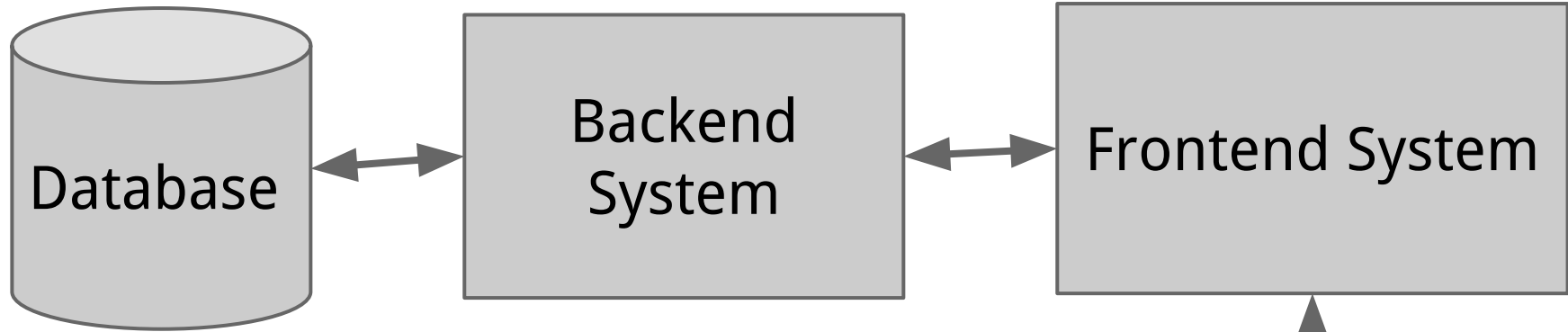


- Print = Only for consumption by human eyeballs
- Return / Resolve = Passing data onwards through the program
 - Return = output of a function
 - `len('foo')` returns the integer 3
 - Resolve = outcome of a statement
 - `'foo' in 'foobar'` resolves to the boolean `True`

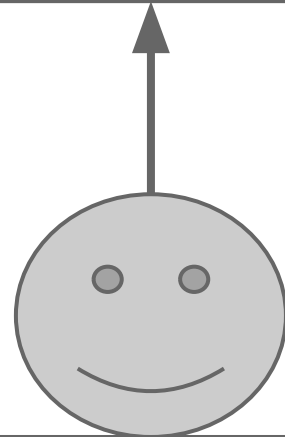
Most of what we will do will be returning & resolving

You **only** print when you would show something on a UI

Inventory Management Scenario



Keep data as data!
Only the frontend should ever
"print" or display anything!



Why print?



- The only time `print` is really used is for simple scripts
- Most applications will likely have a form of UI, or communicate with another application
- Next we'll learn the way to read from the command-line: `input`
 - Sort of the opposite of print
- Again, really only used for scripting...
 - ... and intro to programming lessons :)

The `input` function



The `input` function takes in a string prompt, which it prints to the screen.

It then waits for user input, and returns that input.

```
>>>  
>>> user_message = input('Write a message: ')  
Write a message: This is a nice message  
>>> print('Your message was: ' + user_message)  
Your message was: This is a nice message  
>>> █
```

- Conditional: How to say "if-then" in code

`if` ***condition*** *that resolves to a boolean:*

what to do in that case



Whitespace has meaning in Python. It defines a "block"

Whenever you see a colon, it must be followed by a block

- Block: Set of code grouped together.

Simple Conditional Example



```
if x < 10:  
    print('x is less than 10')  
    print('still in the block')  
print('Outside the block')
```

- The first two print statements are both in the same block, so they're both part of the same "if" statement.
- The third print statement is unindented, so it's not in the block. It will print no matter what value `x` has.

Else statements

Let's convert the following English to Python

- Assume two variables, x and y, both have numeric values
- If x is greater than 10, set z to 5 and print "Foo!"
- If not, check if y is greater than 10 and set z to 3 and print "Bar!"
- If neither of those statements are true, set z to 1 and print "Zap!"
- No matter what, finally print "Done!"

```
# Assume x and y are numbers
if x > 10:
    y = 5
    print('Foo!')
elif y > 10:
    z = 3
    print('Bar!')
else:
    z = 1
    print('Zap!')
print('Done!')
```

Blocks within blocks...



Nested logic is easier to represent in Python than English!

```
if attack_mode:
    if enemies_near:
        attackEnemies()
    else:
        prepareForAttack()
else:
    sleep()
    doADance()
```

What happens if you say `if 5`? Actually, it works fine...

Most values are considered "true" in Python

- False Values:

- `False`
- `0`
- `''` (the empty string - nothing inside the quotes)
- `None`

To test it for yourself, try passing a value into `bool()`

Why did Python do it that way?

You can determine if a variable has been given a value

```
x = input('Enter a value')  
if x:  
    print('You entered ' + x)  
else:  
    print("You didn't enter anything!")
```