ITP 115
Programming in Python

Overview
Output
Variables

What Programming is

"…getting your computer to do stuff."

— *Michael Dawson*

What Programming is NOT

- Magic!
  - Anyone can learn to program with practice
- The computer assuming what you want
  - Must be precise

Program Sequence

- Algorithm
  - Think: **Recipe**
  - Logical sequence of steps to accomplish a task

Algorithm Example

Brew Coffee in a French Press

1. **Grind coffee beans**
2. **Put 4 tablespoons ground coffee in press**
3. **Boil hot water**
4. **Pour 12 ounces of water in press**

5. **Wait 4 minutes**

6. **Push plunger**
7. **Pour coffee in mug**

8. **Enjoy!**

Programming Languages

- Commands that are agreed upon between programmers
  - What commands are available
  - How they are formatted (**Syntax**)

- Syntax
  - Grammar for programming language

- Example

  "?Kaprielian Where is"

Types of Programming Languages

- Low-level language (directly understandable by computer)
  - Machine language – 0's and 1's
  - Assembly language – slightly more intelligible

- High-level language (written in English)
  - Java
  - C / C++
  - C#
  - Perl
  - Python

Translating High-Level Languages

- High-level languages must be **translated** to machine code so a computer can understand

English / Programming Language → Machine Code

```
integer age = 20;

If age is greater than 18
Then print "You can vote."
```

→ ***translate***

```
00100011  00101111
00110111  11011111
11010011  01001011
10001111  11011111
```

High-Level Language (human)

Machine Code (computer)

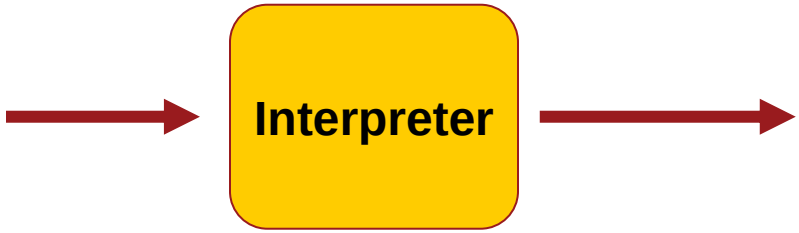
How Source Code is Interpreted

Translating AND Running the Program

source code

```
integer age = 20;

If age is greater than 18
Then print "You can vote."
```



machine code
(executable program)

```
00110111
11011111
```

Each line of source is individually translated and then executed

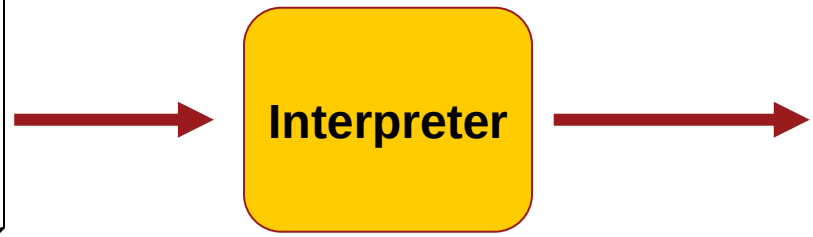
How Source Code is Interpreted

Translating AND Running the Program

source code

```
integer age = 20;

If age is greater than 18
Then print "You can vote."
```



machine code
(executable program)

```
11110111
00011111
```

Each line of source is individually translated and then executed

What is Python?

- Developed in the 1990s
- High-level language
- Interpreted language

python

Why Python?

- Simple syntax
  - Easy to pick up
- Powerful, full-featured
  - Python supports many libraries and applications
- Multi-platform
  - Programs can run on Windows, Mac, Linux, etc.
- Free and open-source

How to Use Python (Must Install Both)

1. Download and Install Python
   - This is needed to run program you write
   - **Version 3.x.x** (whatever the latest version is)

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

2. Download and Install PyCharm
   - PyCharm an IDE for creating Python programs
   - We'll be using this in class
   - Download **Free Community Edition**
   - http://www.jetbrains.com/pycharm/download/

How to Use Python (Optional)

- **repl.it**
  - If you don't have access to your computer or can't install software, you can run Python program directly from a web browser
  - https://replit.com/
  - This is not recommended because it makes submitting your work more difficult

STARTING WITH PYTHON AND PYCHARM

PyCharm

- PyCharm is like a word processor:
  - Use to create and edit code
- It does more!
  - Runs our programs
- Install it in the default location.

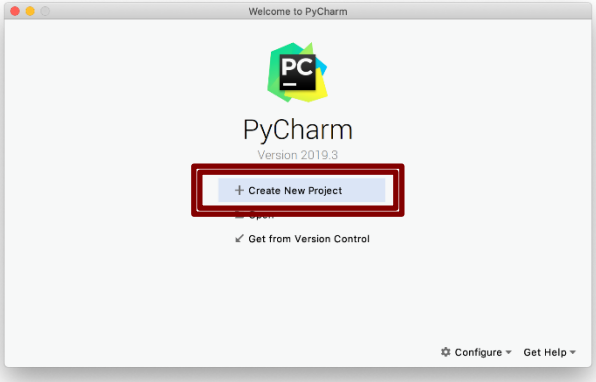
Code

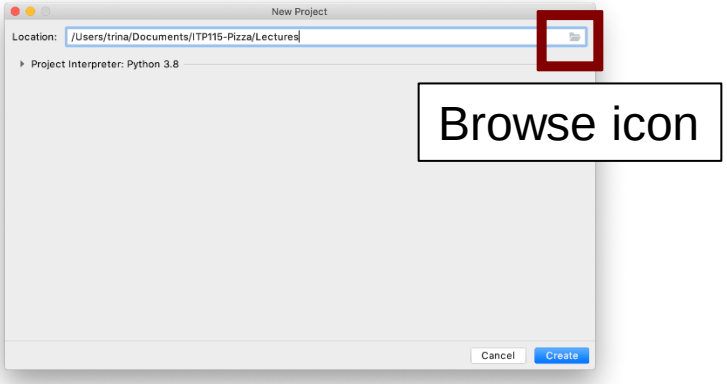
- As coders, we need to know where our code is.
- Create a folder on your computer called ITP115.
  - It can be in your Documents folder, Desktop, etc.

- This is different from where you installed Python and PyCharm.
  - You do not store your Word files under the folder where Word is installed. The same is true for PyCharm.

PyCharm – New Project

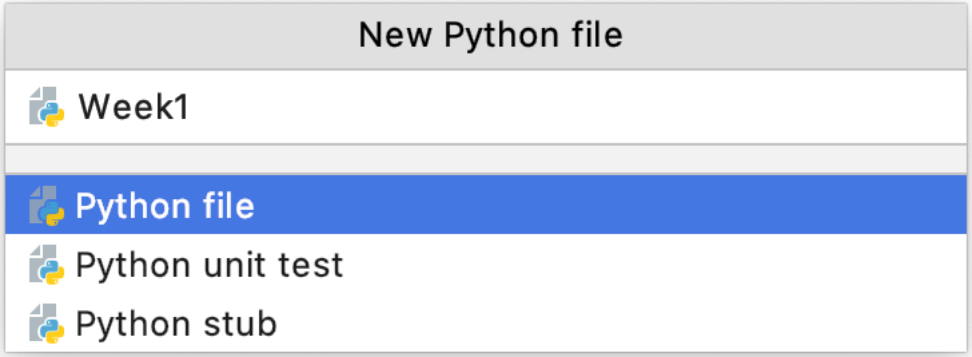
- In the Welcome to PyCharm window, click on the **Create New Project** option.
- In the New Project window, click on the Browse icon (looks like a folder) to find where you created your ITP115 folder.
- If you do not have one, create a folder named **Lectures**.
- Make sure that the correct path is in the Location textfield.





PyCharm – New File

- To create a new Python file, select **File → New...** from the main menu.
- In the little window, select the **Python File** option (not File).
- In the New Python File window, enter a name such as Week1 and press the return key.



Lessons - [C:/Users/R/Dropbox (uscItp)/_ITP 115/Lectures] - ..._working\in class.py - PyCharm 5.0.1

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Lectures _working in class.py

Project

Lectures [Lessons] (C:\U

PyGame Demo

Week 15 - Exception

Week 2 - Variables, I

Week 3 - Operators

Week 4 - Loops, Stri

Week 5 - Lists and T

Week 6 - Functions

Week 7 - Midterm

Week 8 - files

Week 9 - Dictionarie

Week x - OS

Week x - packages, (

in class.py

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

Folders ("directories" show files on your computer)

Code window (where you type commands)

Run in class

```
C:\Users\R\AppData\Local\Programs\Python\Python35\python.exe "C:/Users/R/Dropbox (uscItp)/_ITP
Hello world

Process finished with exit code 0
```

Output window (where things are "printed to" and where you can type input)

4: Run Terminal 6: TODO 0: Messages Python Console Event Log

5:1 n/a UTF-8

Starting Your Program

- `main()` is the container for the code you will be writing

- Examples

```
def main():
    # this is where your code will go

main() # needs to be at the end of file
```

Output

- **print** is the command we use to place text on the output window (basically what the user will see)

Syntax: **print(*sometext*)**

- Examples:

```
def main():
    print("Hello World")
    print("Python is awesome")
main()
```

```
Hello World
Python is awesome
```

String

- Whenever we are dealing with *text* we need to surround it with double quotation marks (`""`)
- In programming we refer to *text* as a **string**
  - Like a "string of characters"

```
print("Hello World")
```

```
Hello World
```

```
print(Hello world)
```

```
Generates error
```

Fun with Printing!

```
print("Some text")
print("Where does this line go?")
```

```
Some text
Where does this line go?
```

- What is happening?

Fun with Printing!

```
print("Some text")
print("Where does this line go?")
```

```
Some text
Where does this line go?
```

- By default, the print command automatically moves the output to the next line
- It does this by printing a *hidden* character called a **newline** (*basically hitting Enter on your keyboard)*
- But we can change this!

More Print Phun

```
print("Some text", end="***")
print("Where does this line go?")
```

```
Some text***Where does this line go?
```

```
print("Some text", end=" ")
print("Where does this line go?")
```

```
Some text Where does this line go?
```

```
print("Some text", end="")
print("Where does this line go?")
```

```
Some textWhere does this line go?
```

Two Ways to Combine Strings

- *Concatenate* two strings together with the **+** operator

```
print("I love " + "pumpkin")
```

```
I love pumpkin
```

- Use commas

```
print("I love", "pumpkin")
```

```
I love pumpkin
```

- What is the difference?
- When should you use one method instead of other?

Two Ways to Combine Strings

- *Concatenate* two strings with commas automatically adds spaces in between*

```
print("I love", "pumpkin")
```

```
I love pumpkin
```

- Either method is fine
- This method makes it easier to combine numbers and texts (later)

**It is possible to change this behavior as we did with newlines in print!*

Programming interlude…

- How would you display…

```
"Python" comes from a comedy troupe
```

Try it yourself

Programming interlude…

- How would you display…

```
"Python" comes from a comedy troupe
```

- Problem: The computer needs to be told that the quotation marks are not the beginning or end of the string but should be printed

Programming interlude…

- How would you display…

```
"Python" comes from a comedy troupe
```

```
print("\"Python\" comes from a comedy troupe")
```

Escape characters

- An *escape character* is…
  - Preceded by a backslash
  - Deviance (or escape) from normal meaning
  - Indicated by 2 characters (backslash + character)
    - But read by computer as 1 character
- Examples

`\"`	Prints double quote (")
`\\`	Prints backslash (\)
`\n`	Prints newline
`\t`	Prints a tab

Comments

- Comments are skipped by Python
  - So they can contain non-code text
  - Like English sentences!

- Intention is to provide reader (or maintainer) extra information to understand the code

Comments

```
# This is a single line comment

"""
This
is a
multiline
comment
"""
```

Triple quote is *technically* called a **docstring**, not a comment

Comments

- What you need to include in comments
  - Name, date, course/company (at beginning)
  - Identify key sections
  - Explain difficult or confusing section
  - Complicated solutions to problems that might not be obvious later

Comments at the Beginning of your Assignments

```
# Tommy Trojan, tommy.trojan@usc.edu
# ITP 115, Spring 2021
# Assignment 1
# Description:
# This program prints some Monty Python quotes
```

End of session 1

Variables

- Think: a bucket that stores **something**
- Represents a small piece of reserved memory
- Contents can change or **vary**
- Variables are the way we label and access information (data)

Variables

- Syntax

```
variable = expression
```

- Example

```
age = 12
```

- `=` is called **assignment**

Variables

```
age = 12
```

"Take the number **12**
and store it in a
variable *(container / bucket)*
called **age***"*

Variable Data Types

- **Integers**

  `int` `3` `-1` `0` `2011`

- **Real Numbers**

  `float` `3.14` `0.094` `-12.0`

- **Character Strings**

  `str` `"Hi"` `""` `"a"` `"44"`

- **Boolean**

  `bool` `True` `False`

Creating Variables

- Syntax

```
variable = expression
```

- Example

```
age = 12
name = "Rob"
tax = 0.0825
isItRaining = False
```

Variable Naming Guidelines

- Name can contain only numbers, letters and underscores
- Name cannot start with a number
- Names are cAsE-sEnSiTiVe
- Choose descriptive names
  - ex. `score` instead of `s`
- Use camelCase (conv)

Python Keywords

and	elif	if	print
as	else	import	raise
assert	except	in	return
break	exec	is	try
class	finally	lambda	while
continue	for	not	with
def	from	or	yield
del	global	pass	

Can't use these keywords as variable names.

More on strings

```
lastName = "Steinbeck"
```

- In Python strings are a special type of variable called an ***object***

Parts of a string

```
lastName = "Steinbeck"
```

This string has 2 parts…

- Its data
  - Its contents: "Steinbeck"
- Its commands (aka ***methods***)
  - Its operations or abilities
  - A method is "*called*" with parenthesis
  - To access a method use the dot **.** operator

String methods

```
lastName = "Steinbeck"
print(lastName)
```

```
Steinbeck
```

```
print(lastName.upper())
```

```
STEINBECK
```

- The string **`lastName`** has "Steinbeck" as its data
- The method **`upper()`** *returns* the data with all capital letters

Common String Methods

- Ex: `s = "tacos"`

Method	Description
`s.upper()`	Returns the uppercase version of the string.
`s.lower()`	Returns the lowercase version of the string.
`s.swapcase()`	Returns a new string where the case of each letter is switched.
`s.capitalize()`	Returns a new string where the first letter is capitalized and the rest are lowercases.
`s.title()`	Returns a new string where the first letter of each word is capitalized and all others are lowercase.
`s.strip()`	Returns a string where all the white space (tabs, spaces, and newlines) at the beginning and end is removed.
`s.replace(old, new)`	Returns a new string where occurrences of the string old are replaced with the string new.

Concatentation

- We can build a string using concatenation (+)

```
language = "python"

message = "I love programming in " + language
```

- *Note: since strings are immutable, you are really creating a new string every time you use the concatenation operator (more in two weeks)*