ITP 115
Programming in Python

Overview
Input and Output

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 2 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

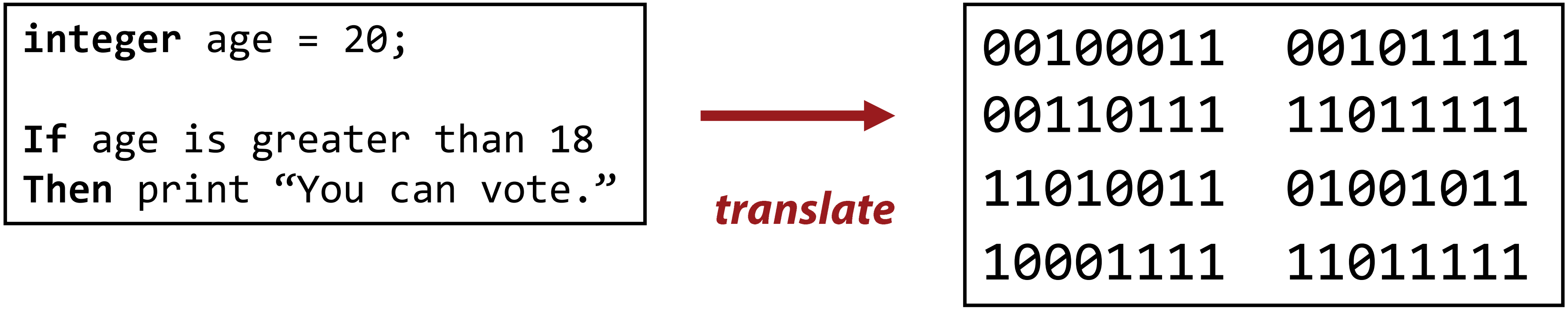
Why so many Programming Languages?

- Consider automobiles
  - 1) Sedan vs. passenger van
  - 2) Toyota vs. Honda
- Different languages are better at certain tasks
- Personal preference

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)

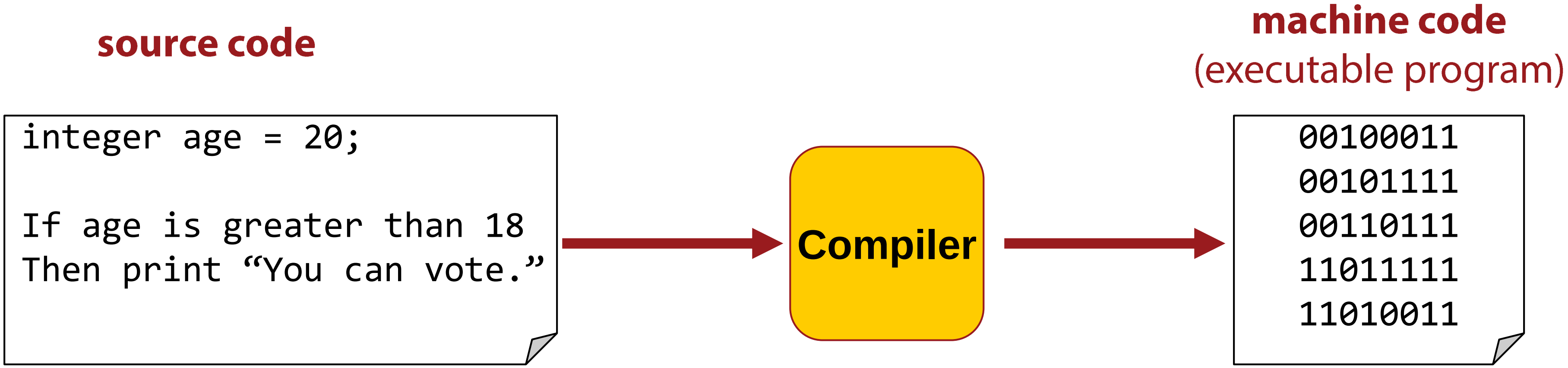
Compiled vs. Interpreted Languages

English / Programming Language → Machine Code

- Compiled Language
  - Entire source code is compiled once
  - This creates an **executable** program which can be run by a computer
  - Ex: Java, C++, C, Visual Basic

How Source Code is Compiled

Translating the Program



source code

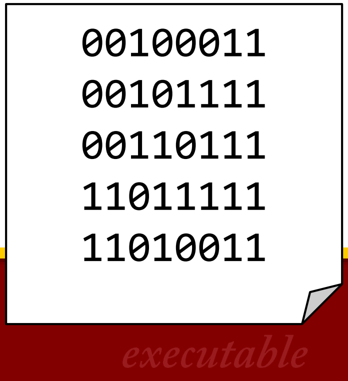
```
integer age = 20;

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

Compiler

machine code
(executable program)

```
00100011
00101111
00110111
11011111
11010011
```

Running the Program



```
00100011
00101111
00110111
11011111
11010011
```

executable

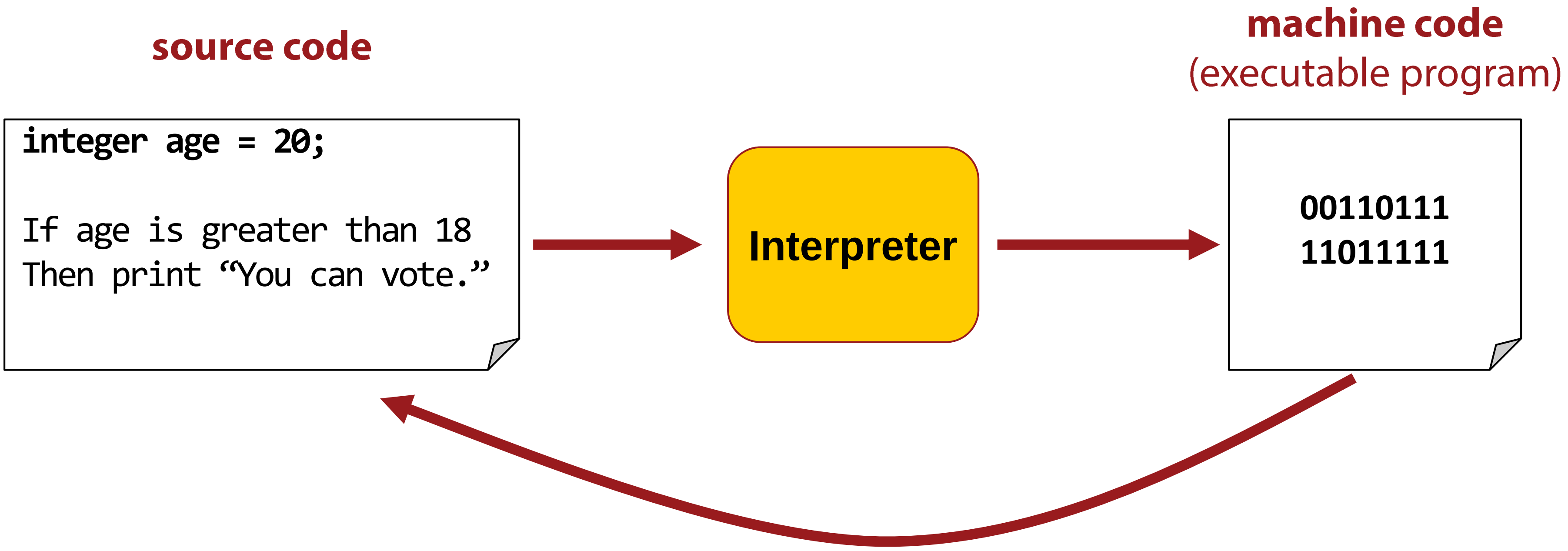
Compiled vs. Interpreted Languages

English / Programming Language → Machine Code

- Interpreted Language
  - Each line of source code is interpreted every time the program runs
  - Ex: Python, Perl, PHP, MATLAB, JavaScript

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.”
```

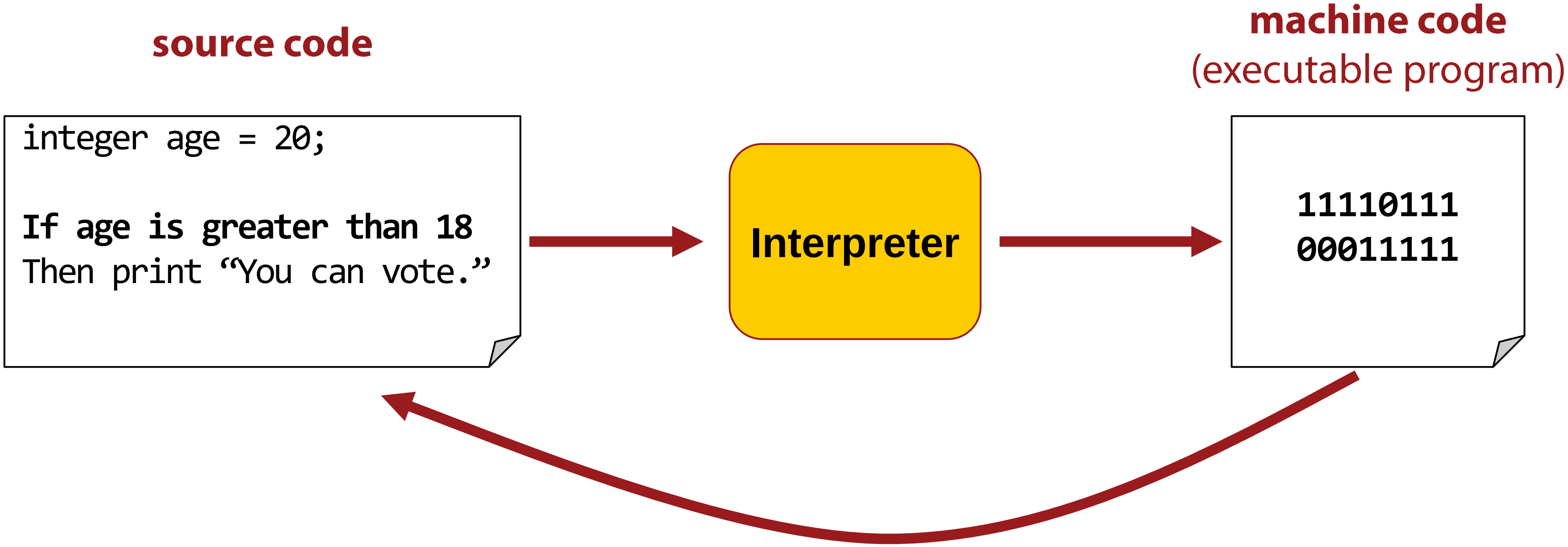
machine code
(executable program)

```
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.”
```

Interpreter

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

GRAHAM CHAPMAN · JOHN CLEESE · TERRY GILLIAM
ERIC IDLE · TERRY JONES · MICHAEL PALIN
MONTY PYTHON and the HOLY GRAIL
"Marvelously zany humor."
– NEWSWEEK

Why Python?

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

How to Use Python

- Download and Install Python
  - **Version 3.5.2** (must use this version)

  https://www.python.org/downloads/release/python-352/

How to Use Python

- Integrated Design Environment (IDE)
  - Software program used to write code
  - Think: "Microsoft Word for programming"

- Python comes with **IDLE**
  - Limited functionality
  - Other IDEs are supported (e.g. Eclipse, PyCharm)

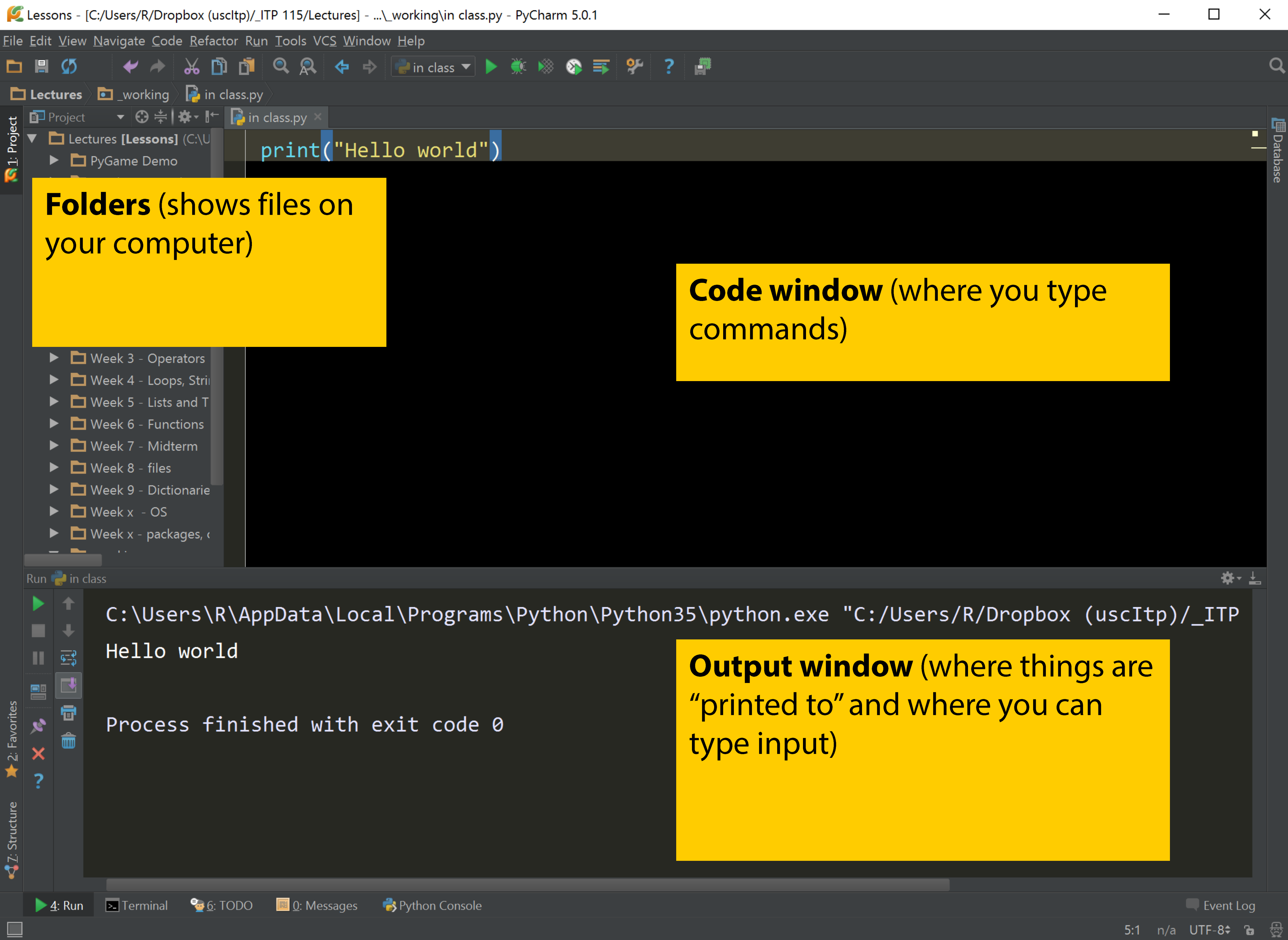
Eric Idle

Required: PyCharm

- PyCharm is an IDE for creating Python programs
  - Easier to use than IDLE
  - We'll be using this in class

- Download Free Community Edition
  - http://www.jetbrains.com/pycharm/download/

STARTING WITH PYTHON AND PYCHARM



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

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


Process finished with exit code 0
```

Folders (shows files on your computer)

Code window (where you type commands)

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

Note

- 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
```

Output

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

Syntax: `print(some_text)`

- Examples

```
print("Hello World")
                               Hello World
print("Python is awesome")
                         Python is awesome
```

More Print Phun

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

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

- What is happening?

More Print Phun

```
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?

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
  """
  ```

Comments

- What 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

```
# Hermione Granger
# ITP 115
# Lab 1
# 1/17/2075

# Description:
# This program simulates a quidditch match
```

End of lecture

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_identifier = 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_identifier = 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

Python Keywords

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

Can't use these keywords as variable names.

More on strings

```
lName = "Steinbeck"
```

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

Parts of a string

```
lName = "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

```
lName = "Steinbeck"
print(lName)
```

```
Steinbeck
```

```
print(lName.upper())
```

```
STEINBECK
```

- The string **`lName`** 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.

Getting User Input

- Use the **input** method

- **It always returns a string**

- Examples:

```
input("Press the enter key to exit.")

name = input("What's your name?")
print("Hi, " + name)
```

Reading in Numbers

- Enter the following code

```
num1 = input("Please enter a number: ")
num2 = input("Please enter another number: ")
print (num1 + num2)
```

- What is the output?

```
Please enter a number: 3
Please enter another number: 3
33
```

Reading in Numbers

- input() **always returns a string**

- **+** combines two strings together

- **Solution**
  - When reading in numbers, you need to **convert** the
    `string` → **`int`**
    or
    `string` → **`float`**

Reading in Numbers

- Enter the following code

```
num1 = int(input("Please enter a number: "))
num2 = int(input("Please enter another number: "))
print( num1 + num2)
```

- What is the output?

```
Please enter a number: 3
Please enter another number: 3
6
```

Conversion Functions

Function	Description	Example	Returns
`float(x)`	Returns a floating-point value by converting x	`float("10.0")`	`10.0`
`int(x)`	Returns an integer value by converting x	`int("10")`	`10`
`str(x)`	Returns a string value by converting x	`str(10)`	`'10'`

How to compress / submit assignments

- Example
- Lab