

Introduction to Computational and Algorithmic Thinking

LECTURE 2 – COMPUTER PROGRAMMING FUNDAMENTALS



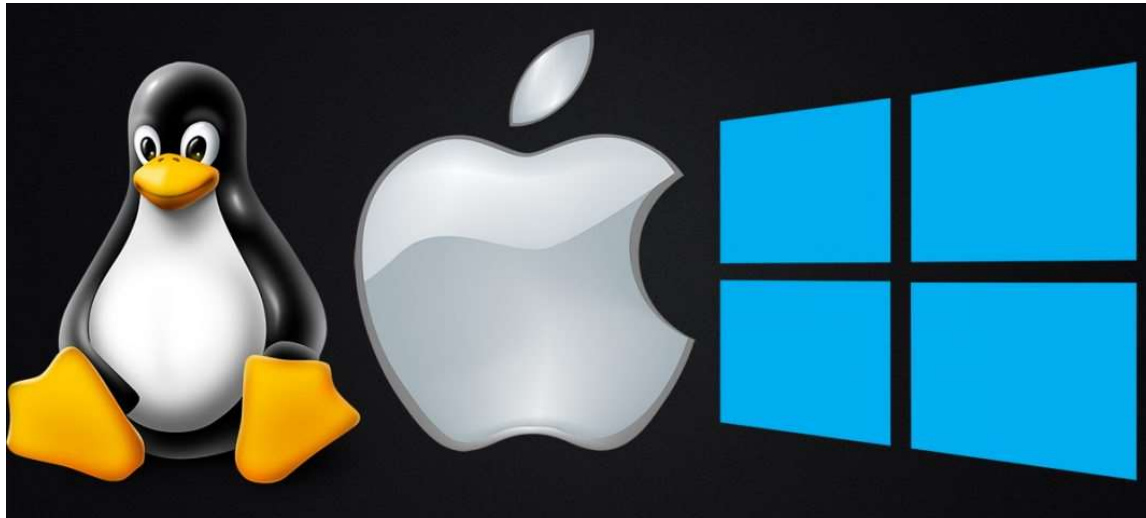
Announcements

This lecture: Computer Programming Fundamentals

Reading: Read Chapter 2 of Conery

What is an Operating System?

Operating System is a program that manages computer hardware and software resources, and provide common services for computer applications.



What is Python?

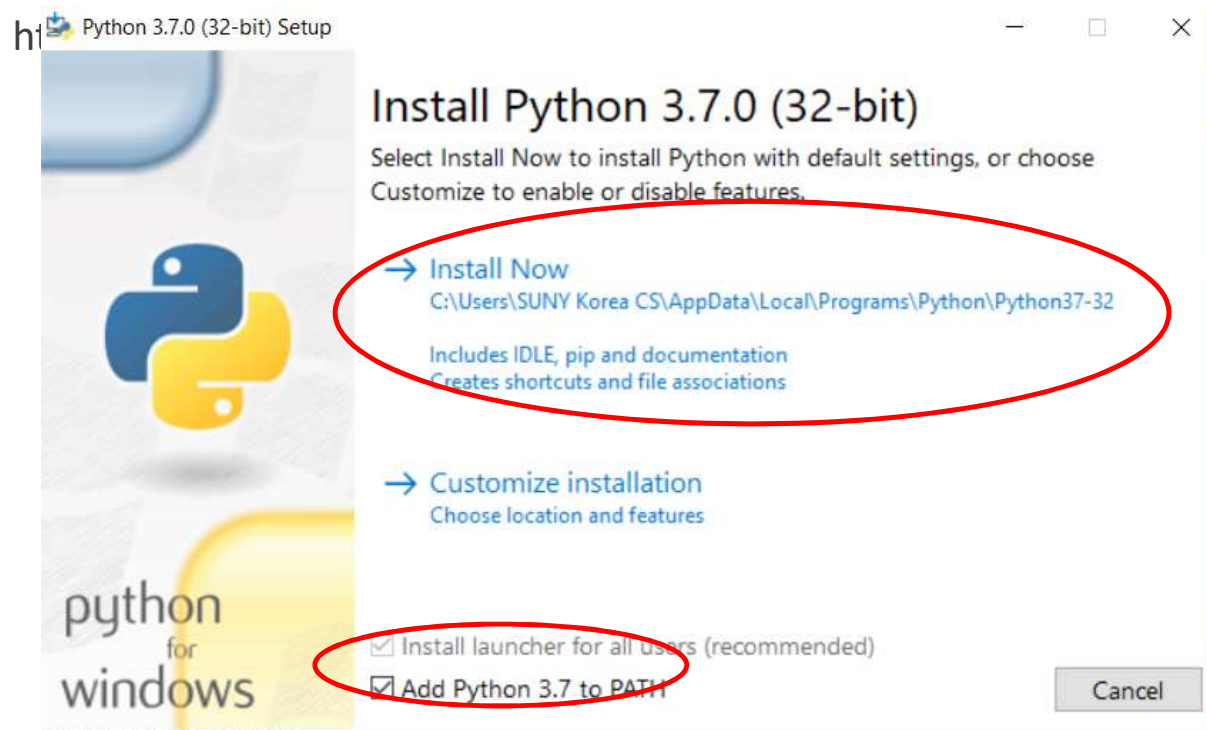
- Python is a computer programming language
 - Relatively simple **syntax** (set of rules programmers must follow when writing programs)
- Python can be used to write simple programs that do basic calculations or very complicated ones
 - Can write basic games!
 - Python is popular with scientists because they can do complex data analysis by writing short programs
- Python can be installed on a wide variety of computer types and operating systems

Python Installation on Windows

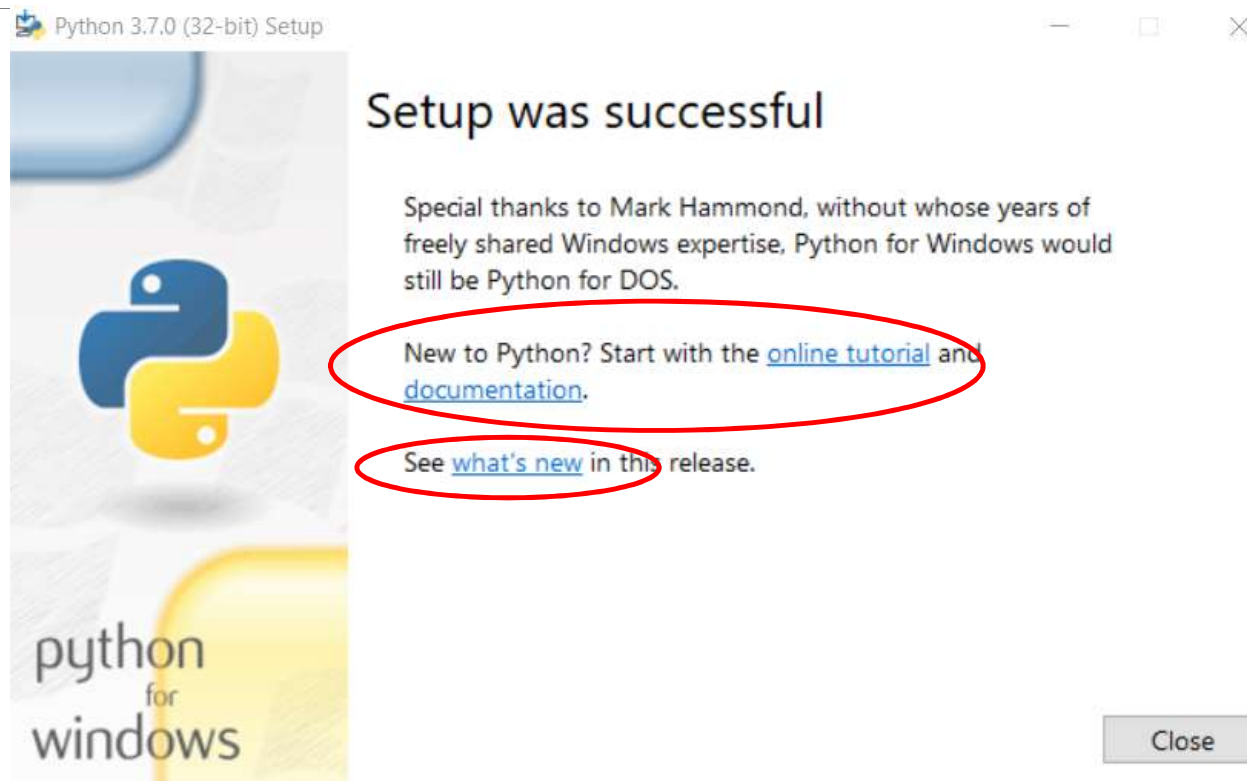
<https://www.python.org/downloads/>



Python Installation on Windows



Python Installation on Windows



Python Installation on MacBook

<https://www.youtube.com/watch?v=8BiYGIDCvvA>

What is a computer program?

- A computer program is a sequence of instructions the computer executes to solve a well-defined problem
- The instructions or steps the programmer writes constitute the **source code** of the program
- In Python, many of these instructions look like regular, everyday English with some extra punctuation thrown in
- There are two basic ways to give commands written in Python to the computer:
 1. Type individual instructions via a **shell**, an interactive program that executes the commands
 2. Write a complete, stand-alone **application** that we can run over and over

Python console / interactive shell

- The **console** (or interactive shell) is
 - a window where a single command or short set of commands can be typed to the computer
 - the computer tries to execute those command
- Python **interpreter**
 - Reads Python instructions typed into the console by the user
 - The interpreter converts them into a form the computer's hardware understands
 - The language that the hardware understands is called **machine language**
- No matter what language is used, at some point the source code must be translated into machine code for the computer to execute it

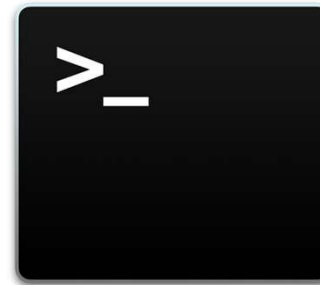
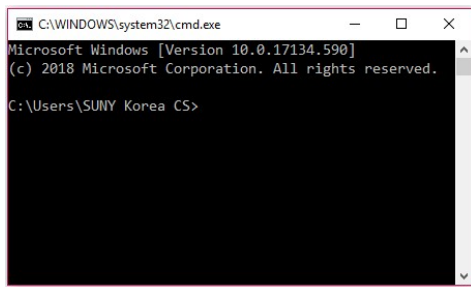
Opening a Terminal

Windows

- Press "Win-R," type "cmd" and press "Enter" to open a Command Prompt session using just your keyboard.

Mac OS

- Finder -> Applications -> Utilities -> Terminal



Some Python Statements

- `print ("helloworld")`
- `1 + 1`
- `a = 1;`
- `b = 2;`
- `a + b`
- `name = "SUNY"`
- `country = "Korea"`
- `print (name + country)`
- `Pi = 22/7`
- `print (type(name))`
- `print (type(Pi))`

The PyCharm IDE

- In this course, an **integrated development environment** (IDE) called PyCharm will be used
- PyCharm is industry-grade software used by professional software developers
 - still easy enough for novice programmers to use
 - First download and install Python from www.python.org
 - Go to www.jetbrains.com/pycharm to download and install the free **Community Edition** of PyCharm

PyCharm Installation

<https://www.jetbrains.com/pycharm/download/#section=windows>

Download PyCharm

Windows

macOS

Linux

Professional

Full-featured IDE
for Python & Web
development

DOWNLOAD

Free trial

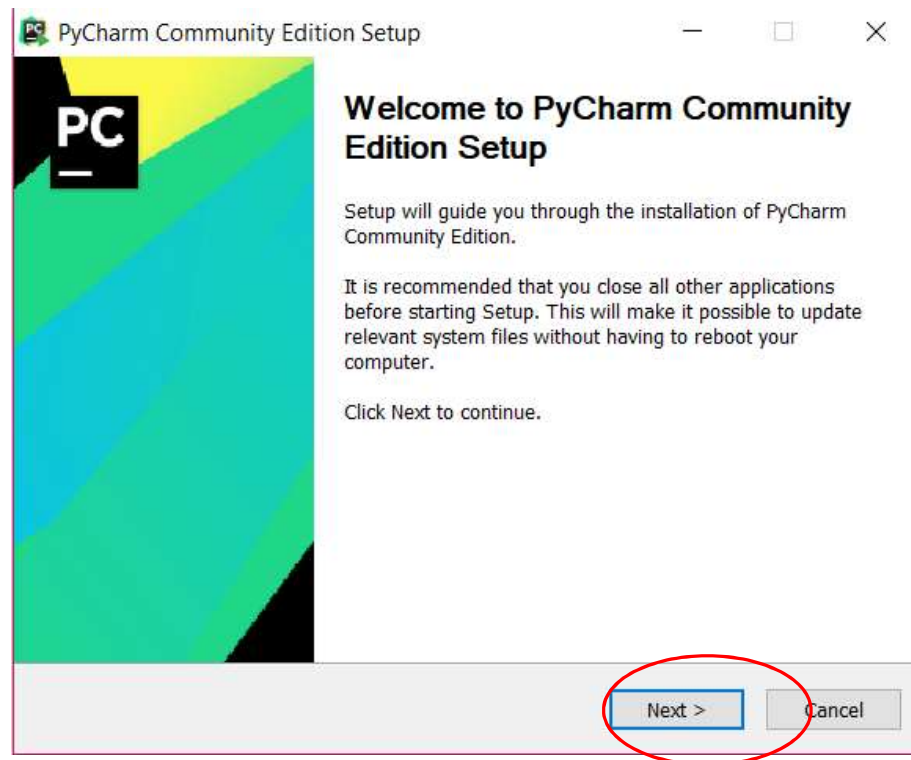
Community

Lightweight IDE
for Python & Scientific
development

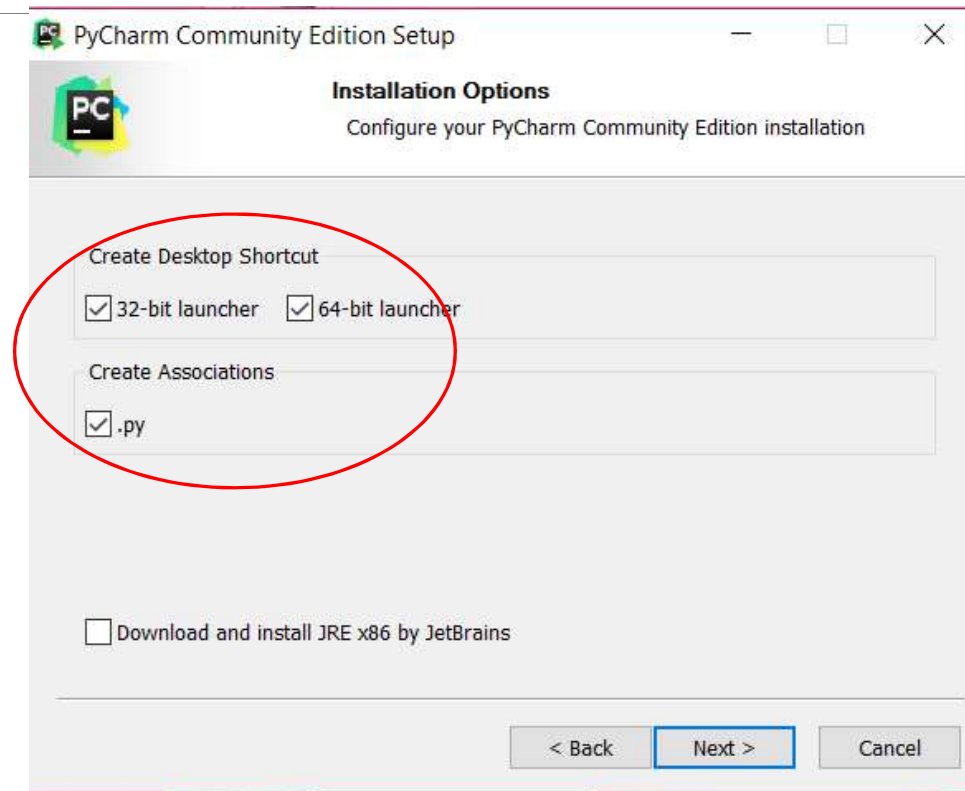
DOWNLOAD

Free, open-source

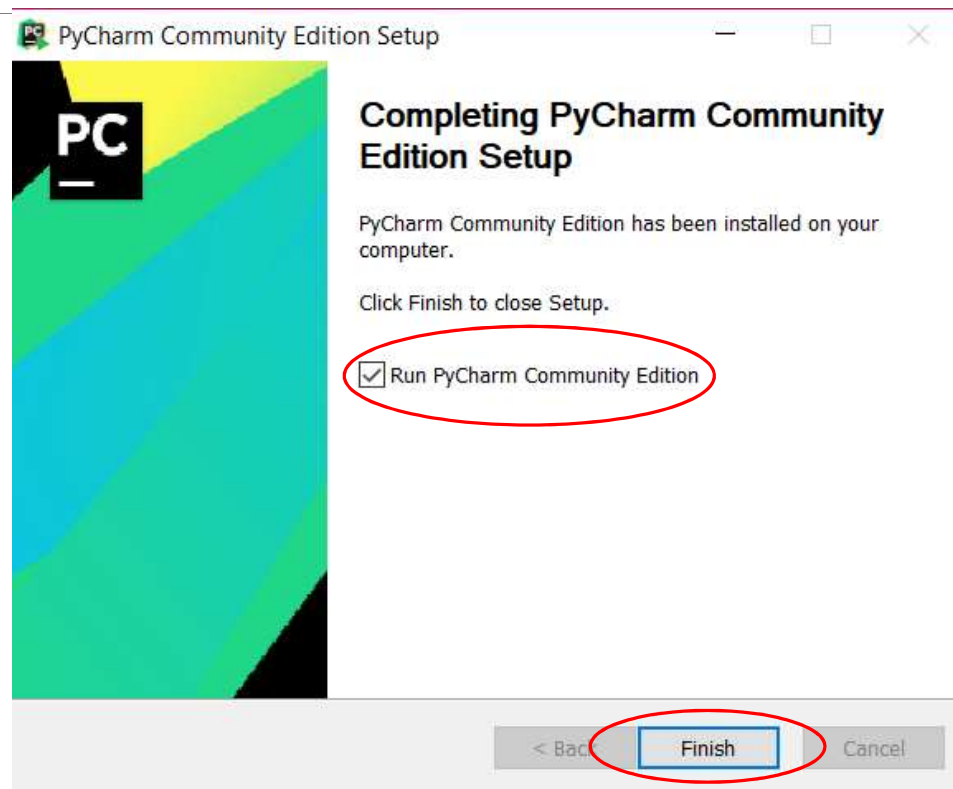
PyCharm Installation



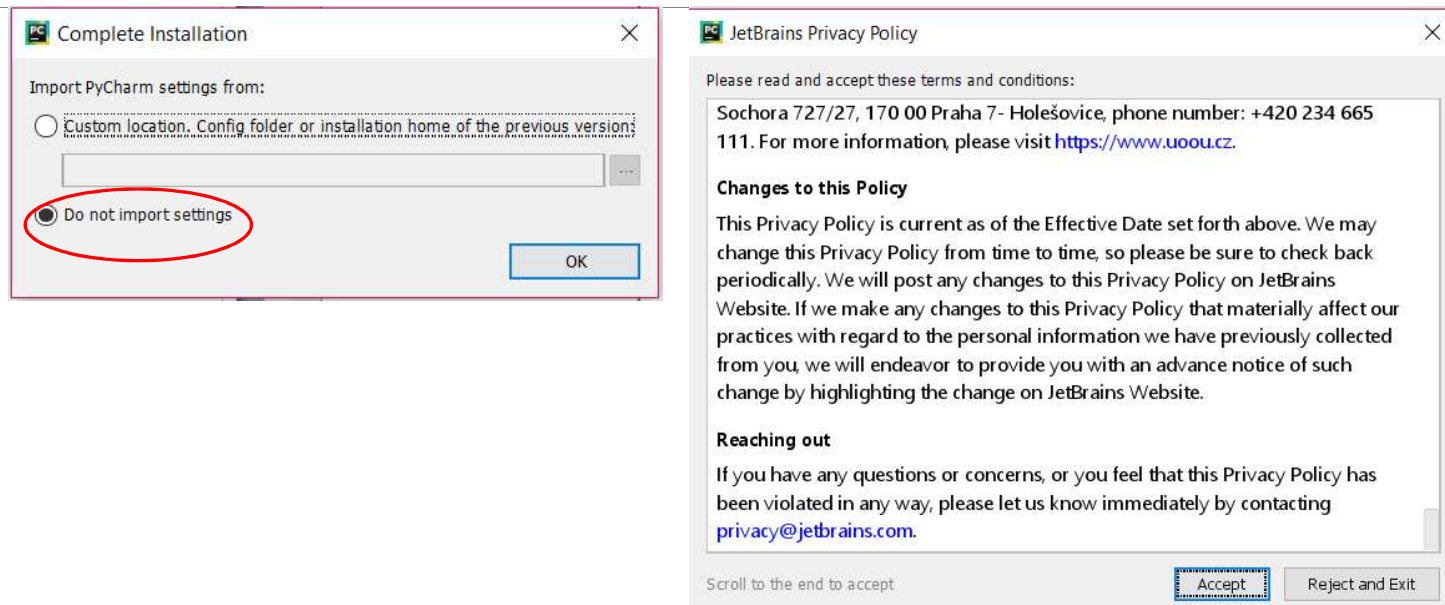
PyCharm Installation



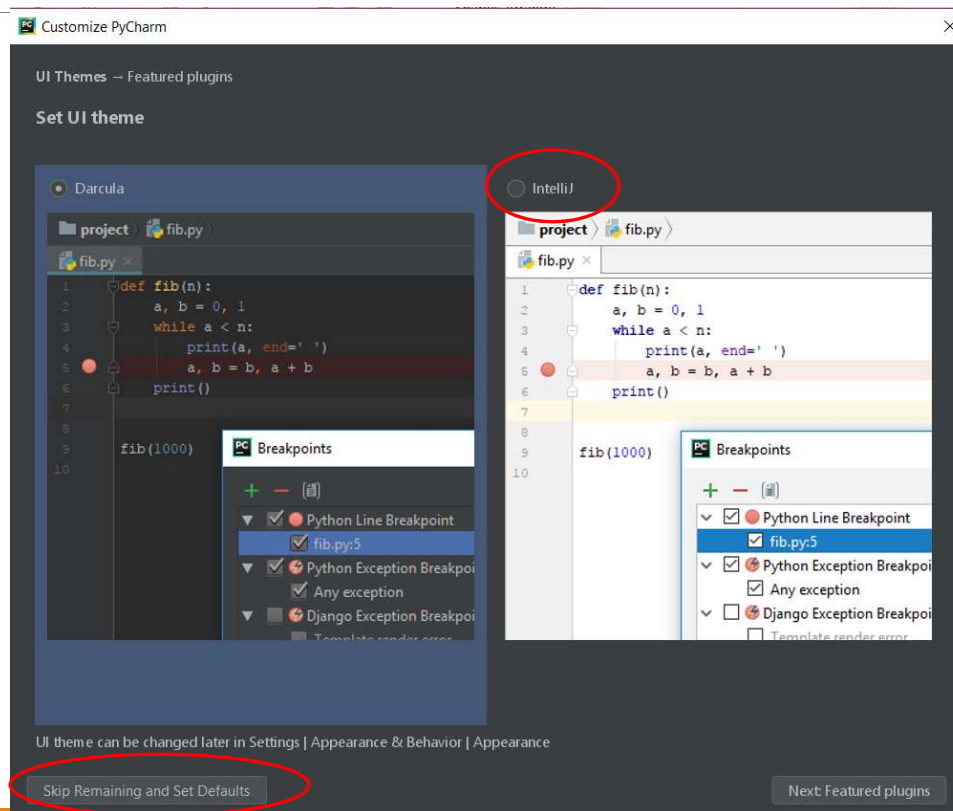
PyCharm Installation



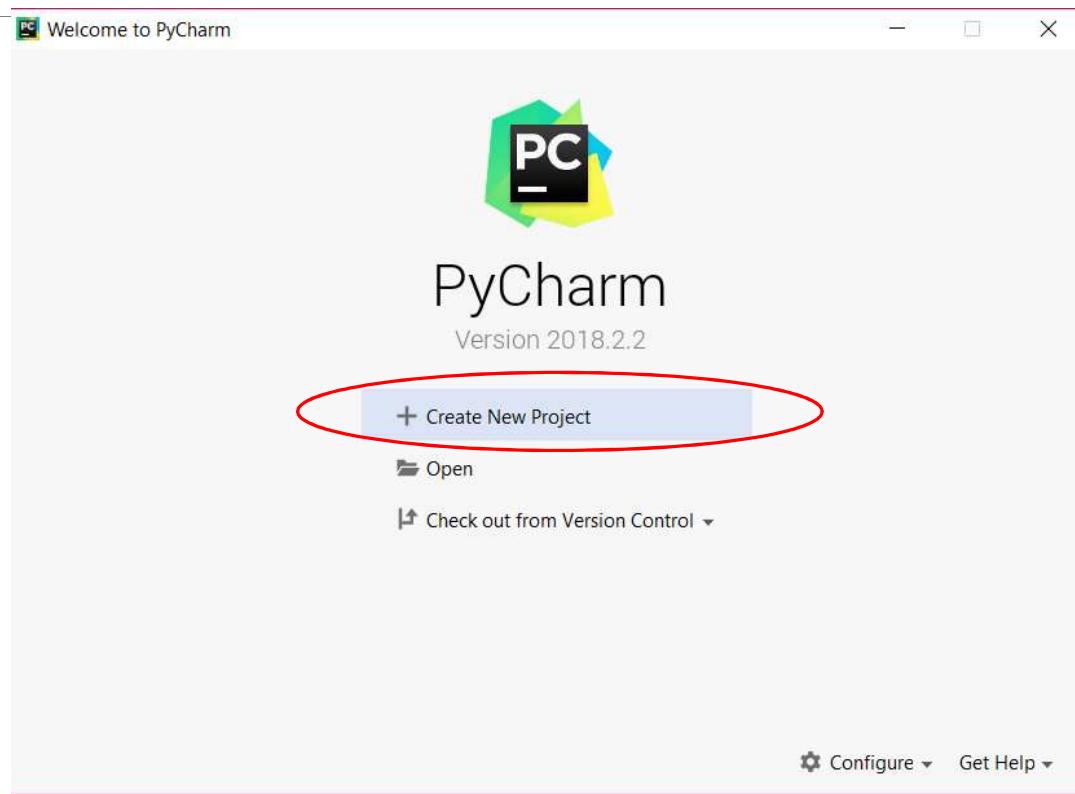
PyCharm Installation



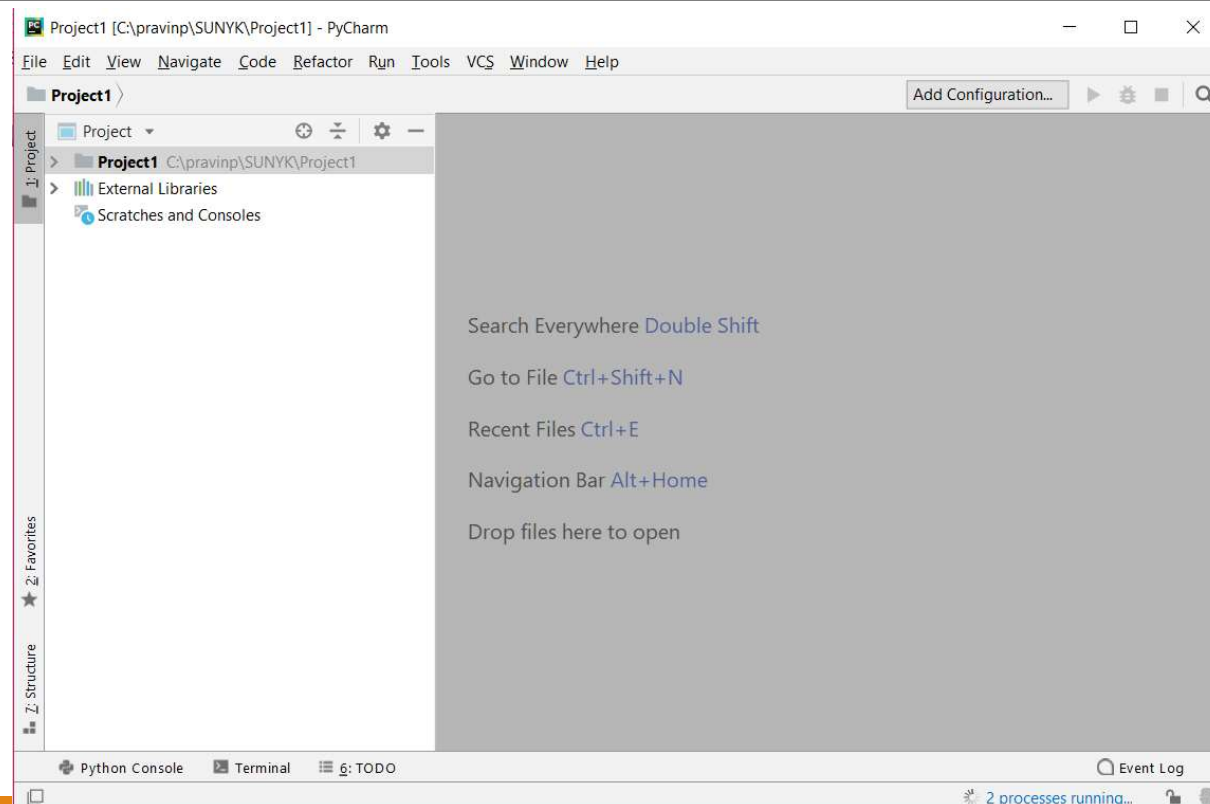
PyCharm Installation



PyCharm Project



PyCharm IDE

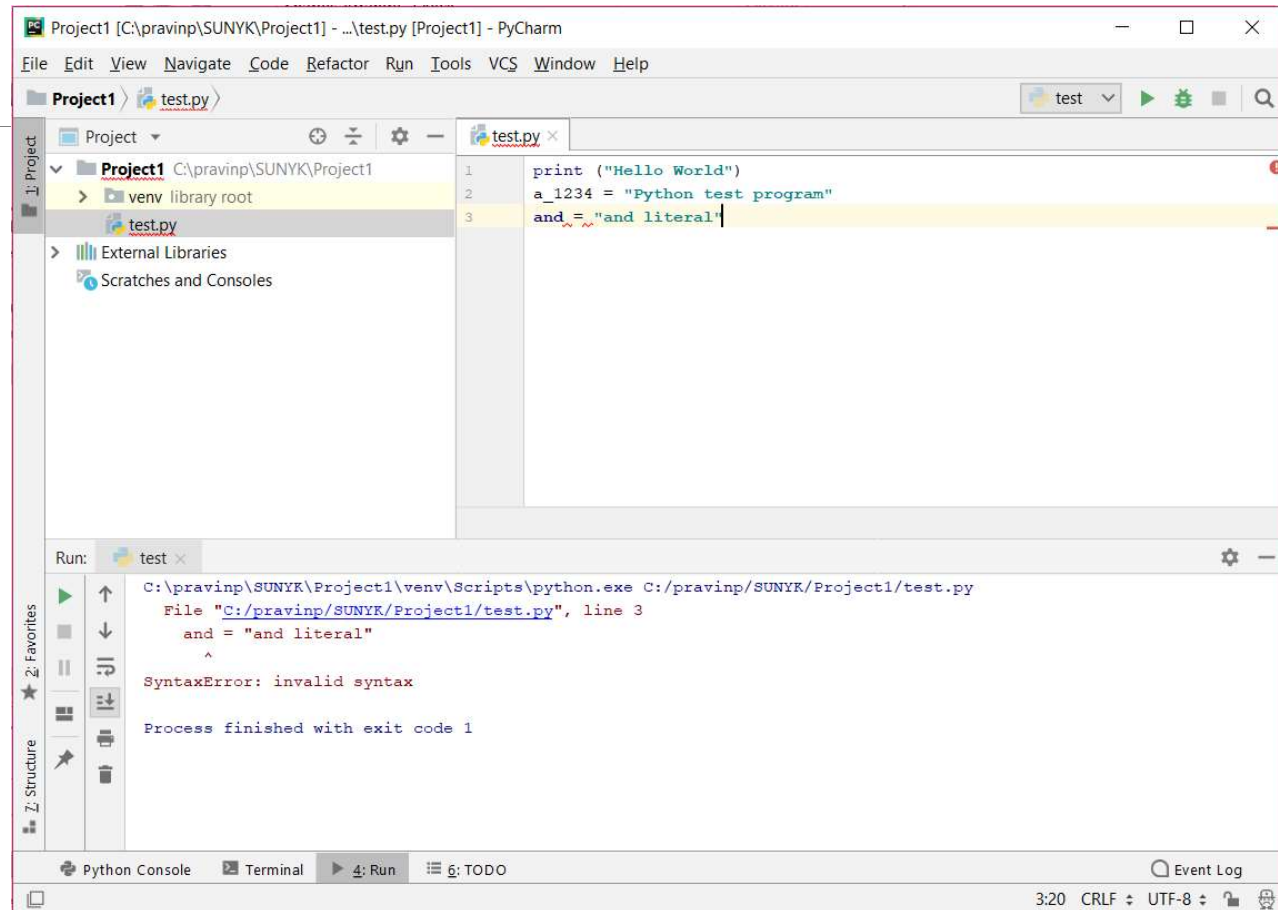


PyCharm Installation on Mac

<https://www.youtube.com/watch?v=wb4HNqQtIII>



PyCharm IDE



Setting Default Python Interpreter in PyCharm

Step 1: Find out installation location of Python program:

Windows terminal command

- where python

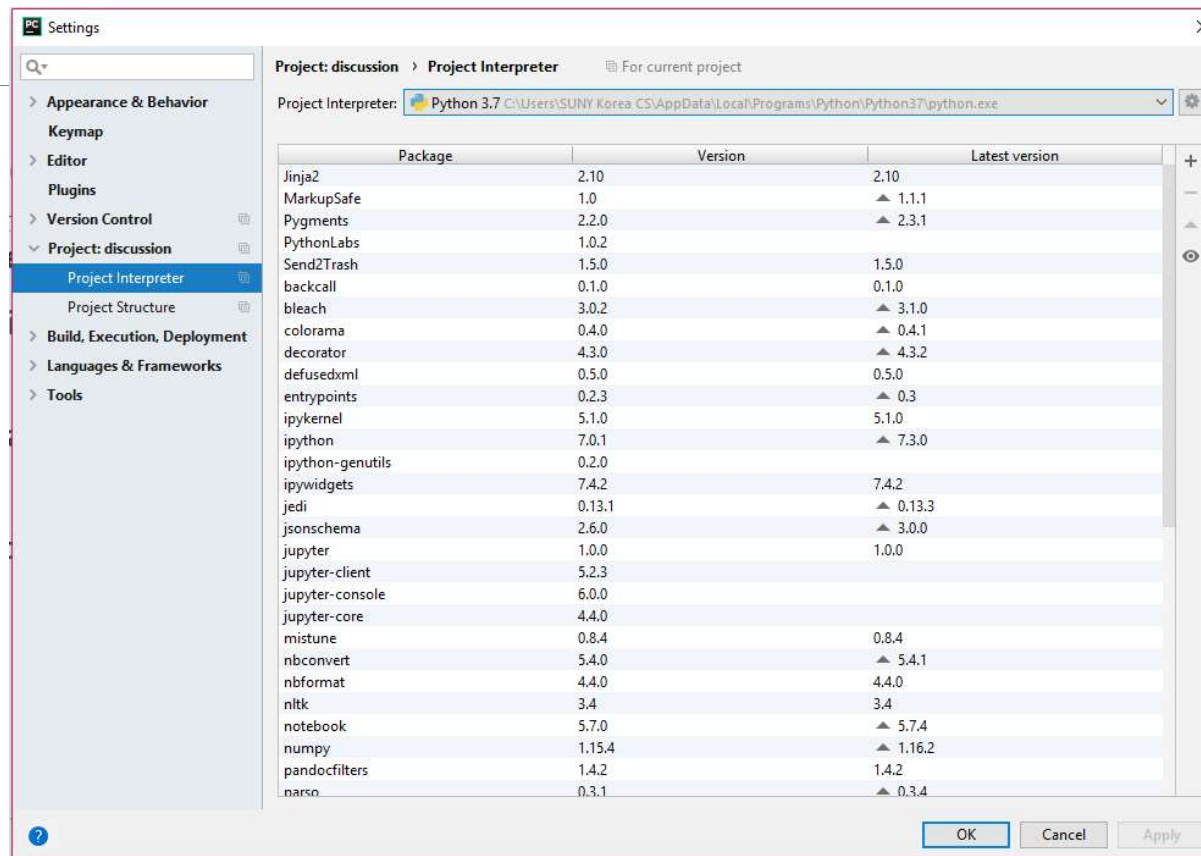
Mac terminal command

- which python3

Note down the paths of python installation.

Setting Default Python Interpreter in PyCharm

Step 2: Change project settings (and New Project Settings) in PyCharm:



PyCharm basics

- To create and run a stand-alone Python program:
 1. Start PyCharm and press the “Create New Project” button.
 2. Pick a “Location” and name for the Project (e.g., “CSE 101”).
 3. Select File Menu > New > Python File and enter the name of the file for the source code.
 4. Write the program and save the file.
 5. After saving, go to Run Menu > Run.
 6. Select the name of the program file to run it.
- The next time the program is to be run:
 - Hit the green triangle in the lower-left corner of the screen.
 - Or, right-click the name of the file and choose Run.

Expressions

- **Expression** represent something like a number, string or value
- **'Hello, world!'** is an expression
 - It has a value
 - In this case, it's a **string** (a sequence of characters)
- Numbers are also expressions
 - 5 is an **integer** expression
 - recall that an integer is zero, or a positive or negative whole number with no fractional part
 - 12.36 is a **floating-point** expression
 - **floating-point** is a format that computers use to represent **real** numbers
 - recall that a real number is zero, or a positive or negative number that might have a fractional part

Expressions

- An expression may consists of operators and operands
 - **2 * 9** is an expression and represents a multiplication
- Python also has **Boolean** expressions, which are expressions that can be **True** or **False**
 - Boolean expressions allow programs to change their behavior from one run to the next. (More soon).
- So there are least three kinds of data in Python programming:
 - Strings
 - Numbers
 - true/false (Boolean) values
- In computer programming, there is a wide variety of data because there is a wide variety of problems that computers can help to solve

Arithmetic in Python

- Some of the simplest statements in Python involve arithmetic expressions, which contain numbers (operands) and mathematical operators
- Arithmetic in Python follows the **PEMDAS** rule:
 1. First, evaluate all expressions in parentheses (**P**)
 2. Then, perform exponentiations (**E**)
 3. Next, perform multiplications (**M**) and divisions (**D**) in left-to-right order
 4. Finally, perform additions (**A**) and subtractions (**S**) in left-to-right order

Arithmetic in Python

- The symbols used for operators are commonly used in other languages and applications (e.g., spreadsheets)
 - add: +
 - subtract: -
 - multiplication: *
 - division for real numbers: /
 - division for integers: // (when a remainder is not needed or desired)
 - remainder: % (gives the remainder of an *integer* division)
 - exponentiation: **

Examples of arithmetic in Python

- $11 + 5 \rightarrow 16$
- $11 - 5 \rightarrow 6$
- $11 * 5 \rightarrow 55$
- $11 / 5 \rightarrow 2.2$
- $11 // 5 \rightarrow 2$
 - This example shows **integer division**. Any remainder is discarded.
- $11 \% 5 \rightarrow 1$
 - The computer divides 11 by 5 and returns the remainder (which is 1) instead of the quotient (which is 2).
 - Use the remainder operator only with integers.

Arithmetic in Python

- The `**` operator does exponentiation or raises a number to a power
- For example, `2 ** 5` would be 32 because $2^5 = 32$
- Recall raising a number to the power $\frac{1}{2}$ is the same as taking a square root
 - So `16 ** 0.5` would be the same as $\sqrt{16}$ which is 4

Arithmetic in Python

- The constant π is built into Python
- First the programmer must make it available by **importing** the **math module**:
 - **import math**
- Then the expression **math.pi** can be used in expressions
 - **math.pi * 2 + 1**
- A Python module is a file consisting of Python source code that are all related somehow
 - For example, the **math** module contains code pertaining to mathematical functions and constants

Variables

- A **variable** in computer programming is similar to the concept of a variable in mathematics
 - A name for some value or quantity of interest in a given problem
- In a program, variables can store a person's age, GPA, name, or almost any other kind of information
 - Value is temporarily stored in the main memory (RAM) of the computer while the program is running
 - A variable is a kind of **identifier** because it identifies (names) something in source code
- It is important to choose identifiers (e.g., variable names) that are informative and helpful
 - Example: **first_name** would be a good variable to store a person's first name, whereas **fn** would not be as good because it's less informative
 - Note how the underscore is used to separate words that define the identifier
 - Spaces are not allowed in variable names

Variables

- A Python variable name may contain lowercase letters, uppercase letters, digits and underscores
 - First character must be a letter or underscore
- Lowercase and uppercase letters are treated as completely different characters
 - Because of this we say that Python is a **case-sensitive language**
 - **First_Name**, **first_name** and **FIRST_NAME** would all be treated as different identifiers
- There are a number of **keywords** built into the Python language that have pre-defined meanings
 - Predefined keywords may not be used as variables

Assignment statements

- To give a value to a variable, write an **assignment statement**
- An assignment statement consists of a variable name, the equals sign, and a value or expression
- Examples:
 - count = 3** (“count is 3” or “count becomes 3”)
 - total = 3.85 + 12.9**
 - firstName = ‘Susan’**
- These examples show three different data types: an integer, a real number, and a string

Assignment statements

- After assigning a value to a variable, you can change the value of the variable with another assignment statement:

total = 5 + 8 + 3

... other code here ...

total = 17 + 6

... etc. ...

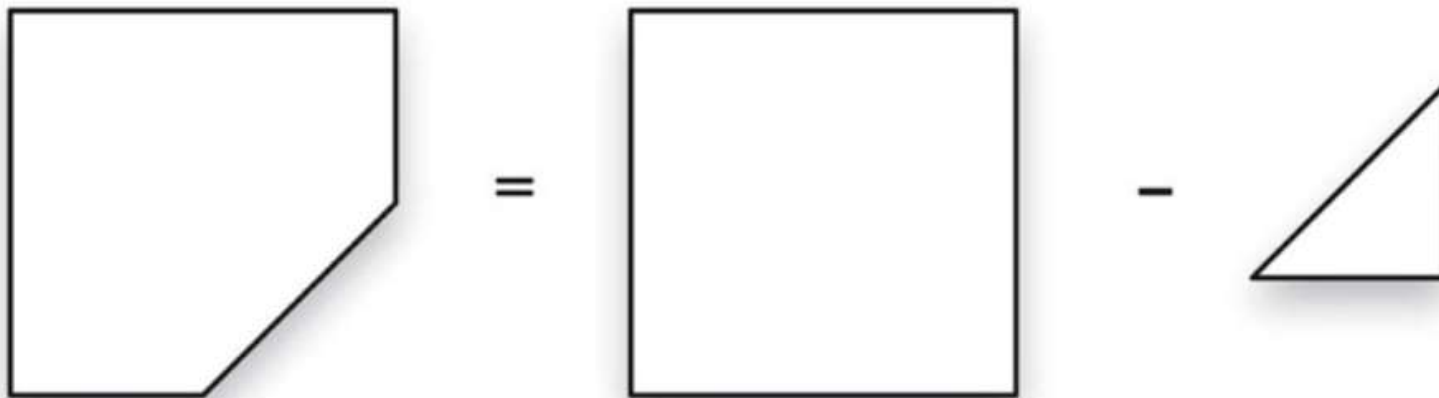
- Variables can also appear on the right-hand side (RHS) of an assignment statement:

next_year = this_year + 1

total_bill = subtotal + tax + tip

Example: Area calculation

Want to compute the area of a square countertop with one corner cut off, as shown here



Example: Area calculation

- Assume that the triangular cut-out begins halfway along each edge
- If the computation is needed only once, say for a 100cm-long countertop, can write a statement like this:

`area = 1002 - 50*50/2`**

- Note that this code has a few issues with it:
 - It's just a formula of sorts with no explanation of what the numbers mean
 - The code works only for countertops exactly 100 cm long. What if we had countertops of other sizes?

Example: Area calculation

- Consider the first issue: lack of clarity
 - # area = area of square - area of triangle**
 - # area of triangle is 1/2 base*height**
 - area = 100**2 - 50*50/2**
- The lines beginning with the # symbol are called **comments**
 - Comments are notes that the programmer writes to explain what the program does
 - Comments do not affect the input or output of the program or anything about how it runs

Example: Area calculation

- Now let's address the other issue: lack of generality

```
side = 100
```

```
square = side**2
```

```
triangle = (side/2)**2 / 2
```

```
area = square - triangle
```

- To compute the area for a countertop of a different size, simply change the first line:

```
side = 100
```

- This code is also more readable; comments aren't needed
 - This is an example of **self-documenting code**
- The spacing in between variables, numbers, and operator is optional, but is included here to make the formulas easier to read

Aside: input statements

- To improve the code further, make it interactive so that the user can provide the value for **side**
- Do this by writing an **input statement**
- An input statement reads a string from the keyboard
- As part of an input statement, the programmer must give a **prompt** message that tells the user what they should enter
- Example: **name = input('What is your name? ')**
- The person's name will be *assigned to* the **name** variable
 - You could also say that we are *saving* the person's name in the **name** variable

Example: Area calculation

- In the case of the area calculation, the user should enter a number, not a string
- Use the following:
`side = int(input('Enter side length:'))`
- To collect a floating-point number, use:
`side = float(input('Enter side length:'))`
- The type chosen – **int** vs. **float** – depends on the application
- For this program, read in a float so the user could enter a fraction of a centimeter if desired
- The last piece of the puzzle is how to display the final result on the computer screen

Aside: print statements

- **print** is a Python command
- It tells Python to display some text on the screen
 - All Python commands are lowercase
- The syntax to print a basic message is just this:
print('Hello, world!')
- Any text printed with additional print commands will appear on a new line
- For Python to print the next output on the same line, do this instead:
print('Hello, world!', end='') # for python3
- This means *print this message, but do not automatically go to the next line*

Aside: print statements

- To print a number, it must first be converted into a string, like so:
`print('The area is ' + str(area))`
 - The assumption here is that **area** is a variable that contains the value we want to print
- When used in this fashion, the + symbol performs **string concatenation**
 - This simply means Python will join the two strings together into one

Example: countertop.py

```
# This program prints the area of a  
# countertop formed by cutting the  
# corner off a square piece of material  
# (e.g., granite).
```

```
side = float(input('Enter side length: '))  
square = side**2  
triangle = (side/2)**2 / 2  
area = square - triangle  
print('The area is ' + str(area))
```

Example: coins.py

- Here is an example of the remainder operator in integer division
- Given a total number of cents, the computer should print how many dimes, nickels, and pennies are needed to make that change while minimizing the number of coins
 - The code will make good use of variables
 - It will use the **str** command to print variables containing numbers to the screen
 - Recall that **str** converts a number to a string so that it can be concatenated with other strings

Example: coins.py

```
cents = int(input("Enter the number of cents: "))
```

```
dimes = cents // 10
```

```
cents = cents % 10
```

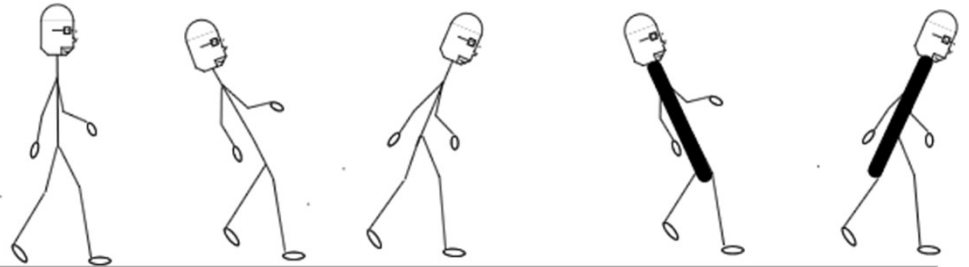
```
nickels = cents // 5
```

```
cents = cents % 5
```

```
pennies = cents
```

```
print("That number of cents is equal to " +  
      str(dimes) + " dimes, " + str(nickels) + "  
      nickels and " + str(pennies) + " pennies.")
```

Escape sequences



- Escape sequences in programming languages like Python allow printing characters (symbols) on the screen that perform special functions
- In Python, some of the escape sequences are:
 - `\t` shifts the text to the right by one tab stop
 - `\n` prints a newline
 - `\"` prints a double quotation mark
 - `'` prints a single quotation mark
- A lone backslash character is called the **line-continuation character** (it's not really an escape sequence, though)
 - This symbol is a signal to Python that the current statement spans two or more lines of a file

Example: limerick.py

Source code:

```
print('There was an old man with a beard\n\  
Who said, \"It's just how I feared!\n\n\  
\tTwo owls and a hen\n\  
\tFour larks and a wren\n\  
Have all built their nests in my beard.')
```

Output:

```
There was an old man with a beard  
Who said, "It's just how I feared!"  
Two owls and a hen  
Four larks and a wren  
Have all built their nests in my beard.
```



Questions?
