Learn to Code with Python

Chia James Chang
Microsoft Corp.
Volunteer teacher @TSG of ROLF & ROLCA

Agenda

- Introduction
- Get Started
- Python Crash Course
  - Basics
  - Operators and Expressions
  - Control Flow
  - Functions and Modules
  - Data Structures
  - Inputs & Outputs
  - Object Oriented Programming
  - Exceptions
  - Standard Library
- Class Projects
- What Next

Resources

- python.org. (download, document, community, etc.)
- Learn To Code With Python, the website for this course.
- Useful Books
- A Byte of Python, Swaroop C H.
  - Allen B. Downey's "Think Python 2nd Edition by Allen B. Downey" book.
  - The Quick Python Book 3rd Edition by Naomi Ceder.
  - Invent Your Own Computer Games with Python 3rd Edition, Al Sweigart, March, 2015.

- IDE (Integrated development environment)
  - Python's IDLE (part of Python 3 installation).
  - Visual Studio Code (recommended).
  - repl.it - Online Python IDE (quick learning).
- Basic commands
  - Windows: 11 basic commands you should know (cd, dir, mkdir, etc.)
  - How to use the Terminal command line in Mac OS.
  - Basic Linux/Unix Commands with Examples.

Weekly Schedule

- First Week on 11/5/2019
- Second Week on 11/12/2019
- Third Week on 11/19/2019
- Fourth Week on 11/26/2019

First week on 11/5/2019

- Python was up and running on both PCs and Macs.
- Visual Studio Code was up and running on both PCs and Macs.
- Learned how to use the tools, Visual Studio Code and Python console.
- Finished the Introduction, 1. First Step, and touched the 2. Basics.
- Wrote the first program, "Hello, World!", ran it, and extended it.
- Wrote some more tiny programs to get our hands wet.
- etc.

Second week on 11/12/2019

- Python Crash Course
- Write lots of small programs

Third week on 11/19/2019

- Review all the subjects covered in the class.
- Class Hands-On Projects
    - Design and develop one or more projects.

Forth week on 11/26/2019

- Review all the subjects covered in the class.
- Class Hands-On Projects
    - Design and develop one or more projects.

Introduction

Why Learn Computer Programming?

- Programming fosters creativity, reasoning, and problem solving.
- Programming is fun and challenging.
- "As young people create Scratch projects, they are not just learning how to write computer programs. They are learning to think creatively, reason systematically, and work collaboratively— essential skills for success and happiness in today's world." --- Professor Mitchel Resnick, Director, MIT Scratch Team, MIT Media Lab.

Think like a computer scientist

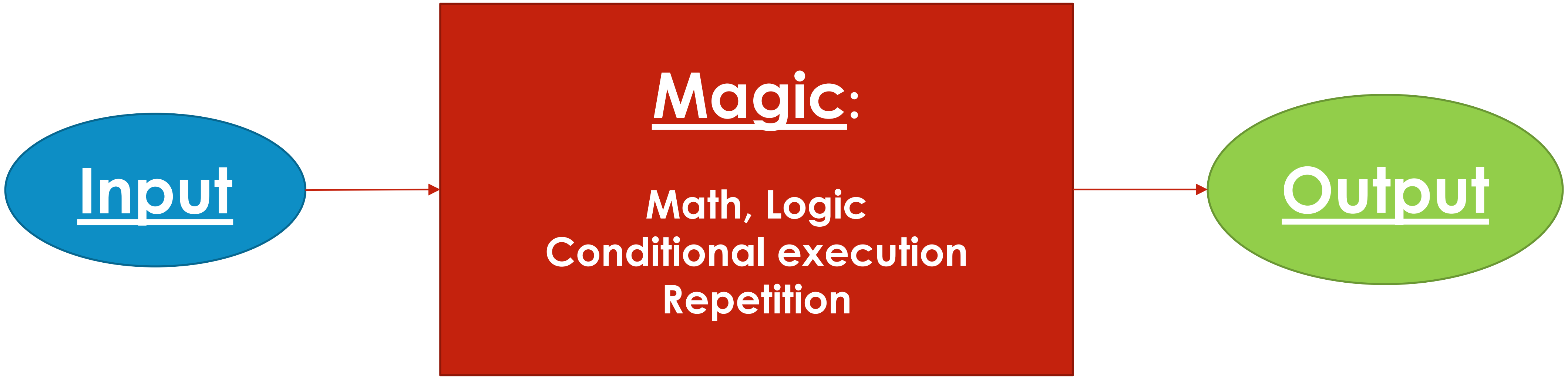
- The way of **thinking** combines some of the best features of:
  - **Mathematicians**: formal languages.
  - **Engineers:** design, assemble and evaluate.
  - **Scientists:** observe, form hypotheses, and test predictions.
  - **Entrepreneur**: marketing, ROI.
- The single most important skill for a computer scientist is **Problem solving**: formulate problems, think creatively about solutions, and express a solution clearly and accurately.

What is programming?

- Programming means to write programs, code, or instructions that cause a computer to perform some kind of action.

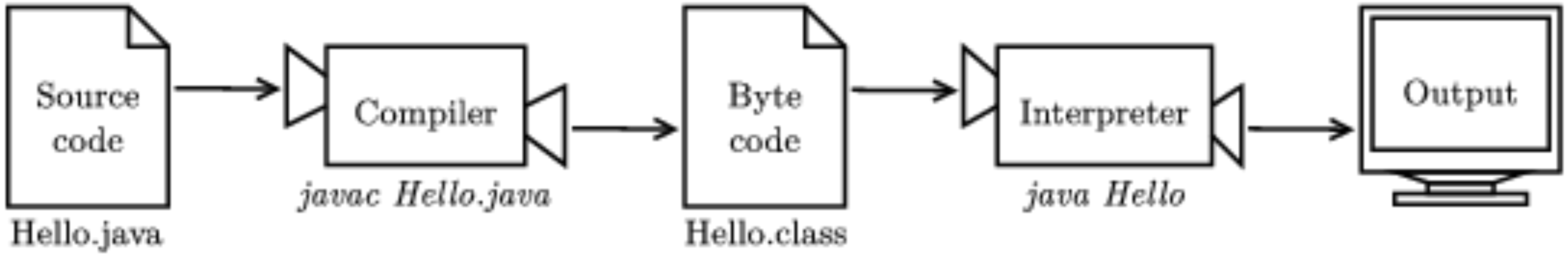
What is a program?

- A **program**, aka **code**, is a sequence of instructions that specifies how to perform a computation.



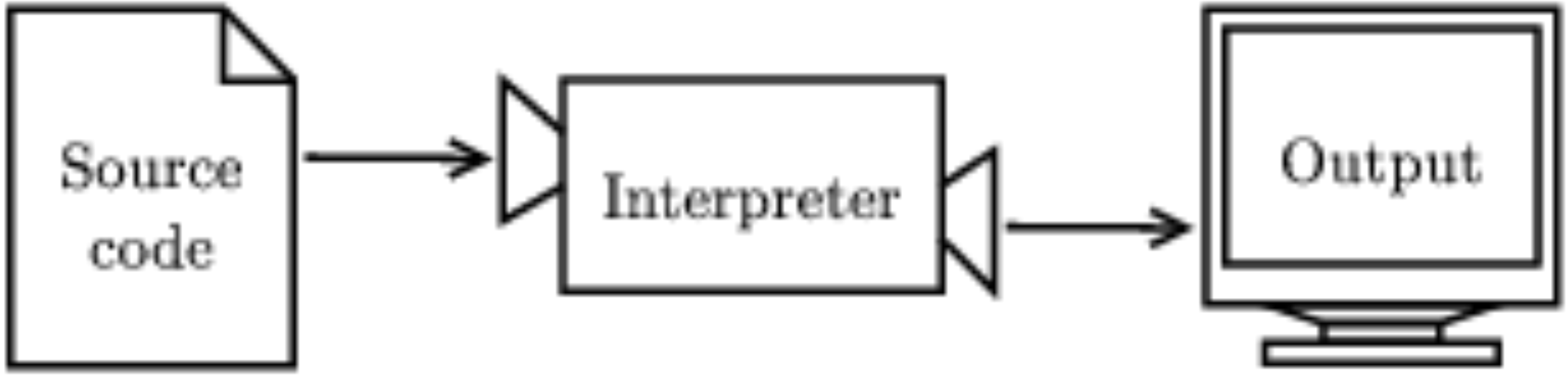
The process of compiling and running a Java program

- Source: http://greenteapress.com/thinkjava6/html/thinkjava6002.html#sec8



How interpreted languages are executed

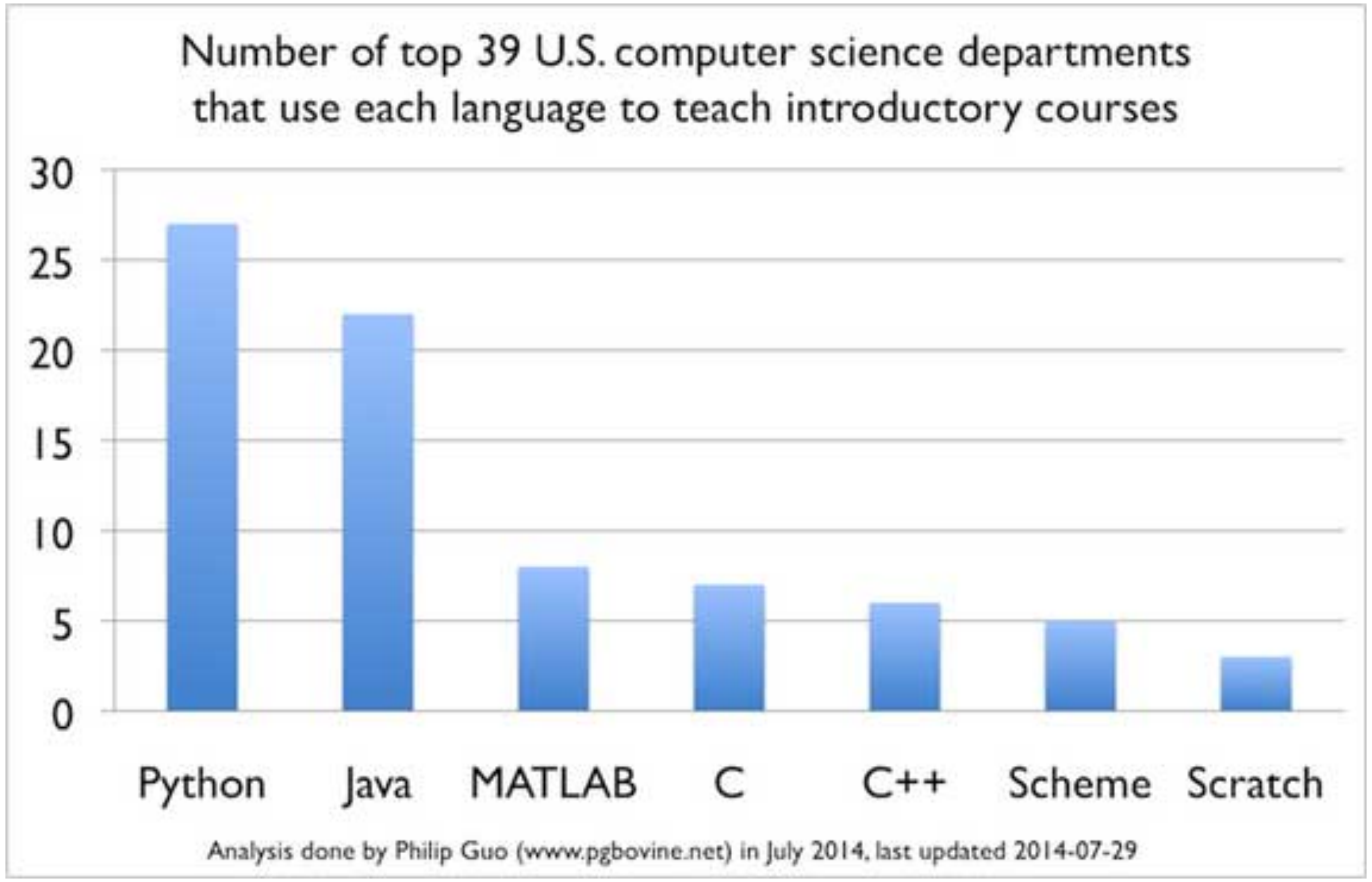
- Source: http://greenteapress.com/thinkjava6/html/thinkjava6002.html#sec8



What is debugging?

- Programming is error-prone. For unusual reasons, programming errors are called bugs and the process of tracking them down is called debugging.
- Three kinds of errors can occur in a program:
  - **Syntax errors**: syntax refers to the structure of a program and the rules about that structure. For example, parentheses have to come in matching pairs, so (1 + 2) is legal, but 8) is a syntax error.
  - **Runtime errors**: the error does not appear until after the program has started running.
  - **Semantic errors**: If there is a semantic (meaning or logic) error in your program, it will run successfully without any error but it will no do the right thing.

Most Popular Languages Taught in Colleges in USA



Why Python?

Simple/Readable	Interpreted
Easy to Learn & Use	Object Oriented
Free and Open Source	Extensible
High-Level Language	Expressive
Portable/Cross-platform	Complete—"Batteries included"

Why Python?

- What Python doesn’t do as well
  - Python isn’t the fastest language.
  - Python doesn’t have the most libraries.
  - Python doesn’t check variable types at compile time.
  - Python doesn’t have much mobile support.
  - Python doesn’t use multiple processors well.

History of Python

- Python's name was came from Monty Python's Flying Circus of British comedy show first broadcast in the 1970s.
- Python was created in 1989 by Guido van Rossum in the Netherlands.
  - 2005-2012: Google
  - 2012: Dropbox
- Python 2 released on 2000
- Python 3 released on 2008

How to Learn to Code?

- Like anything you try for the first time, it's always best to start with the basics.
- Try each of the examples and the programming exercises, so you can see how they work.
- The better you understand the basics, the easier it will be to understand more complicated ideas later on.
- Break a problem down into smaller pieces.
- If that still doesn't help, sometimes it's best to just leave it alone for a while. Sleep on it, and come back to it another day.

Have Fun!

- Think of programming as a way to create some fun games or applications that you can share with your friends or others.
- Learning to program is a wonderful mental exercise and the results can be very rewarding.
- Most of all, whatever you do, have fun!

Get Started

Download and Install Tools

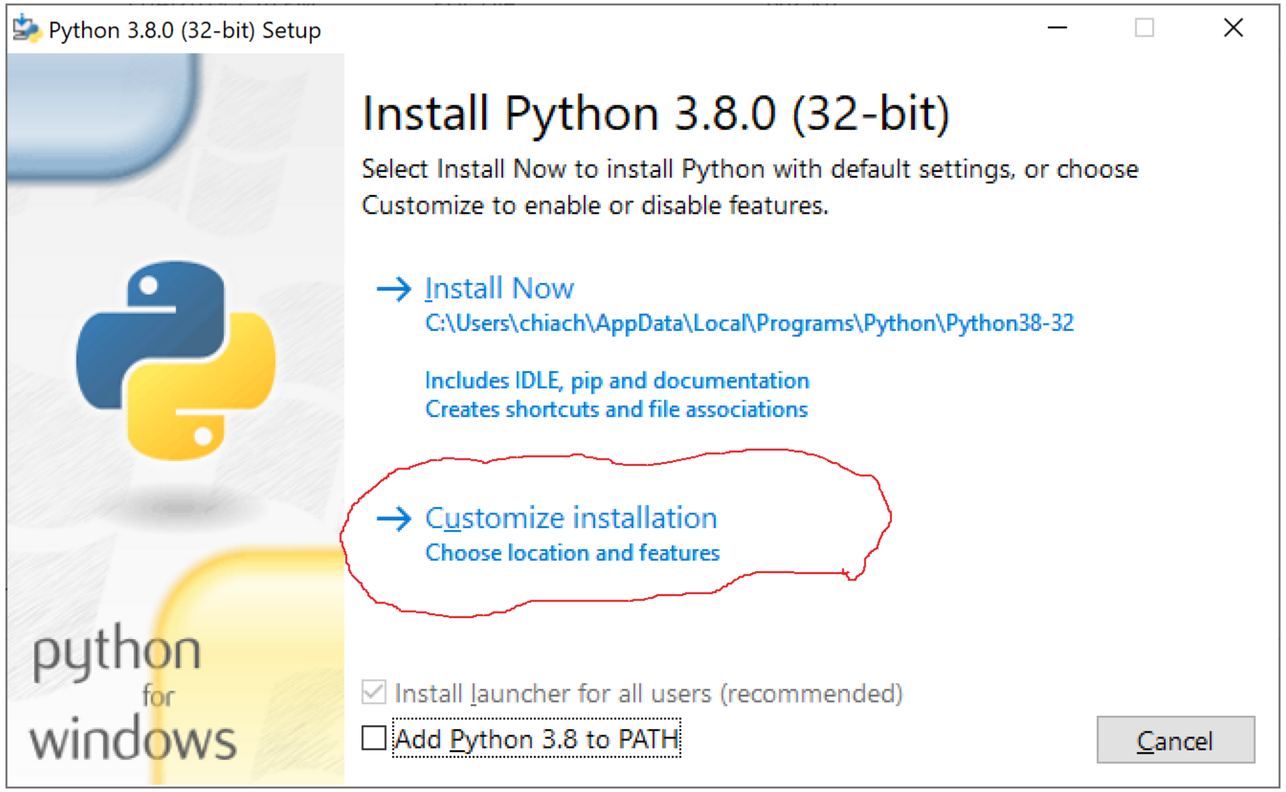
- Download and install Python.
  - https://www.python.org/downloads/
- Download and install one of the following IDE (Integrated Development Environment).
  - Download and install Visual Studio Code.
    - https://code.visualstudio.com/download

Download and Install Python

Python Download URL:
https://www.python.org/downloads/

Install Python #1

- Select **Customize Installation**



Python 3.8.0 (32-bit) Setup

Install Python 3.8.0 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\chiach\AppData\Local\Programs\Python\Python38-32

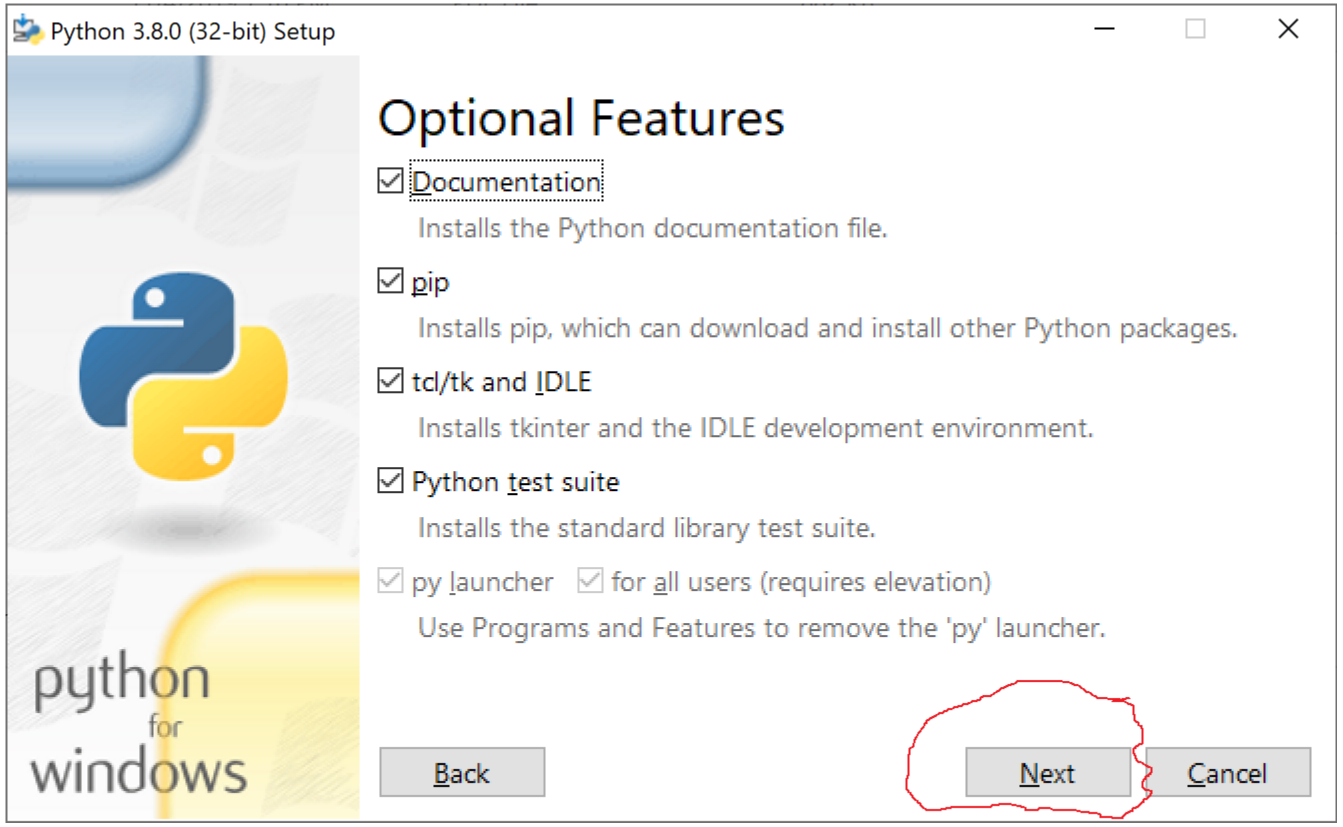
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

[✓] Install launcher for all users (recommended)
[ ] Add Python 3.8 to PATH

Install Python #2

- Optional Features



Python 3.8.0 (32-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation file.

☑ pip
Installs pip, which can download and install other Python packages.

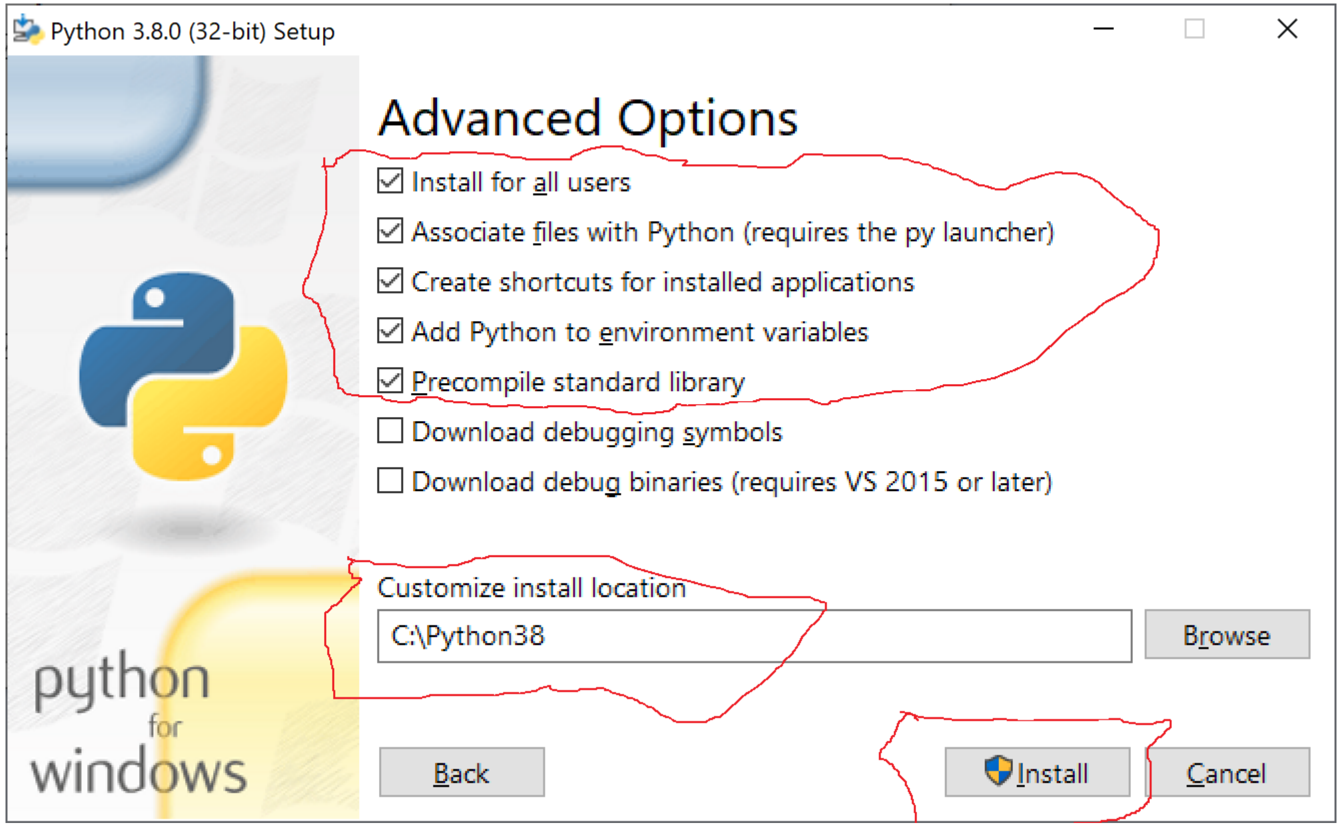
☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)
Use Programs and Features to remove the 'py' launcher.

Install Python #3

- Advanced Options



Python 3.8.0 (32-bit) Setup

Advanced Options

- [✓] Install for all users
- [✓] Associate files with Python (requires the py launcher)
- [✓] Create shortcuts for installed applications
- [✓] Add Python to environment variables
- [✓] Precompile standard library
- [ ] Download debugging symbols
- [ ] Download debug binaries (requires VS 2015 or later)

Customize install location

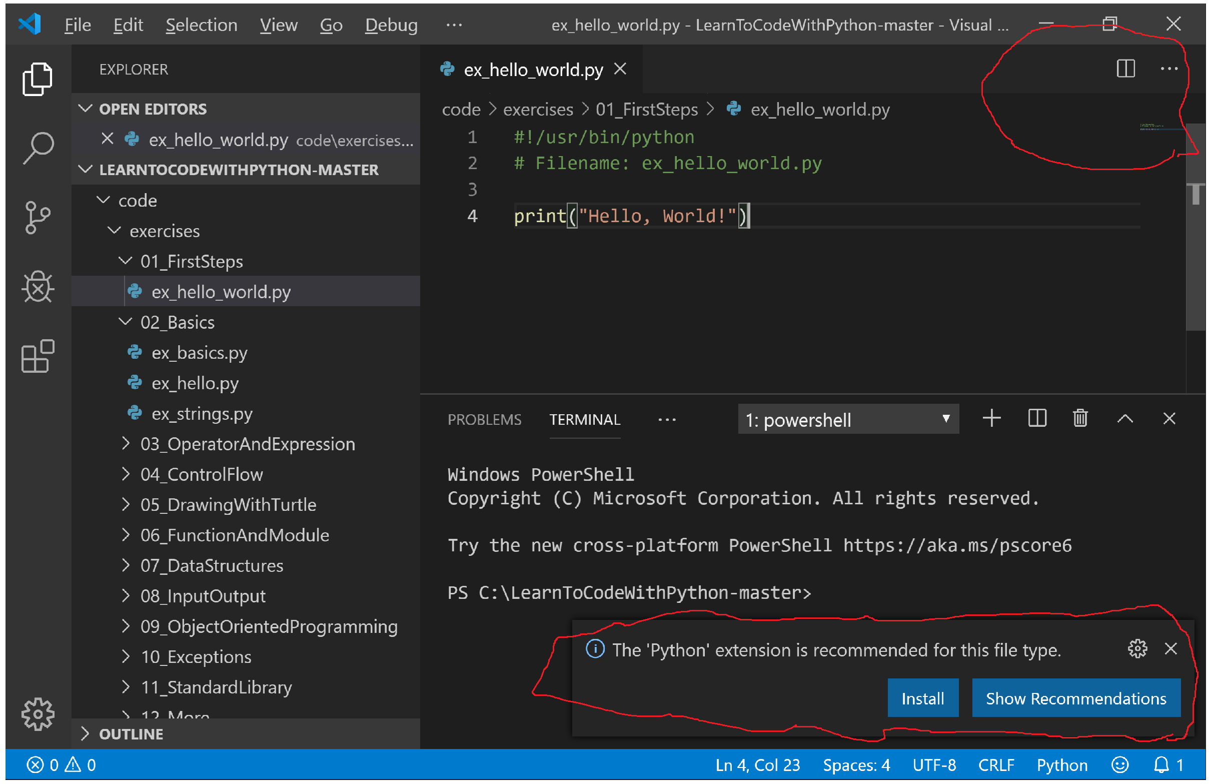
C:\Python38

Download and Install Visual Studio Code

Visual Studio Code Download URL:
https://code.visualstudio.com/download

Install Visual Studio Code #1

- Install Python Extension



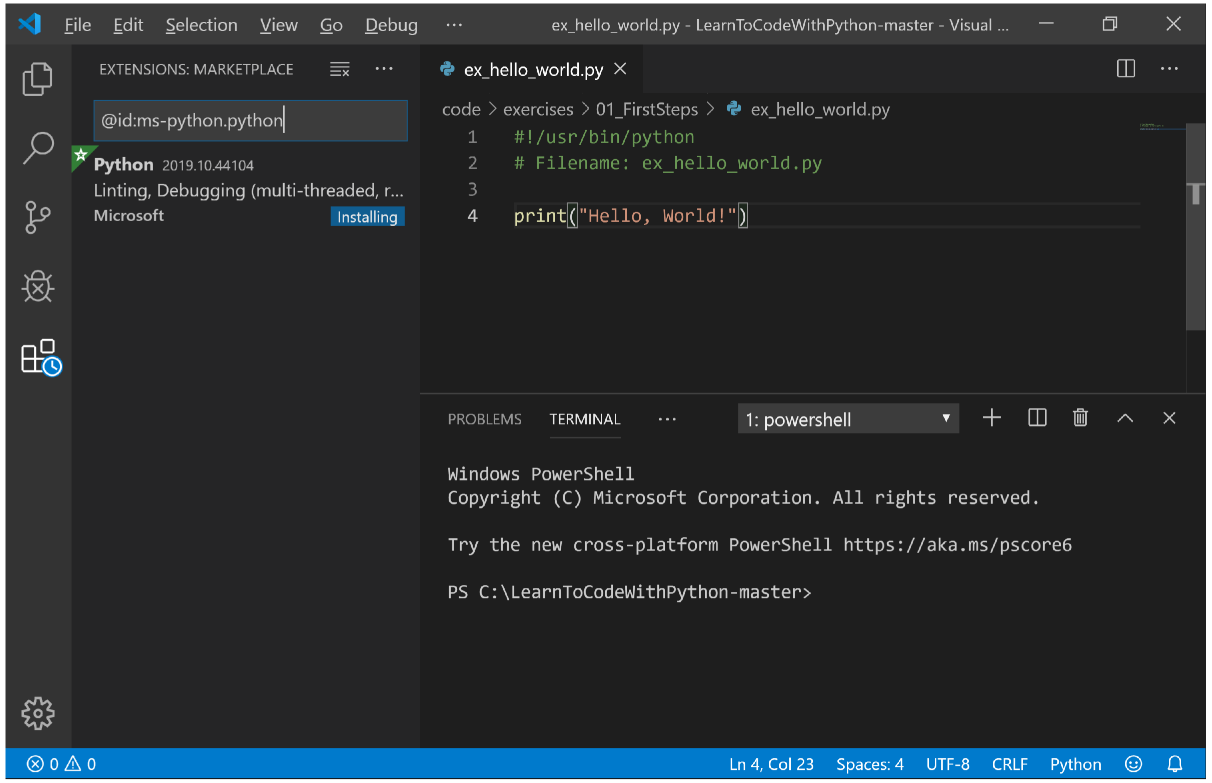
```python
#!/usr/bin/python
# Filename: ex_hello_world.py

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

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\LearnToCodeWithPython-master>
```

The 'Python' extension is recommended for this file type.

Install Visual Studio Code #2

- Run Arrow

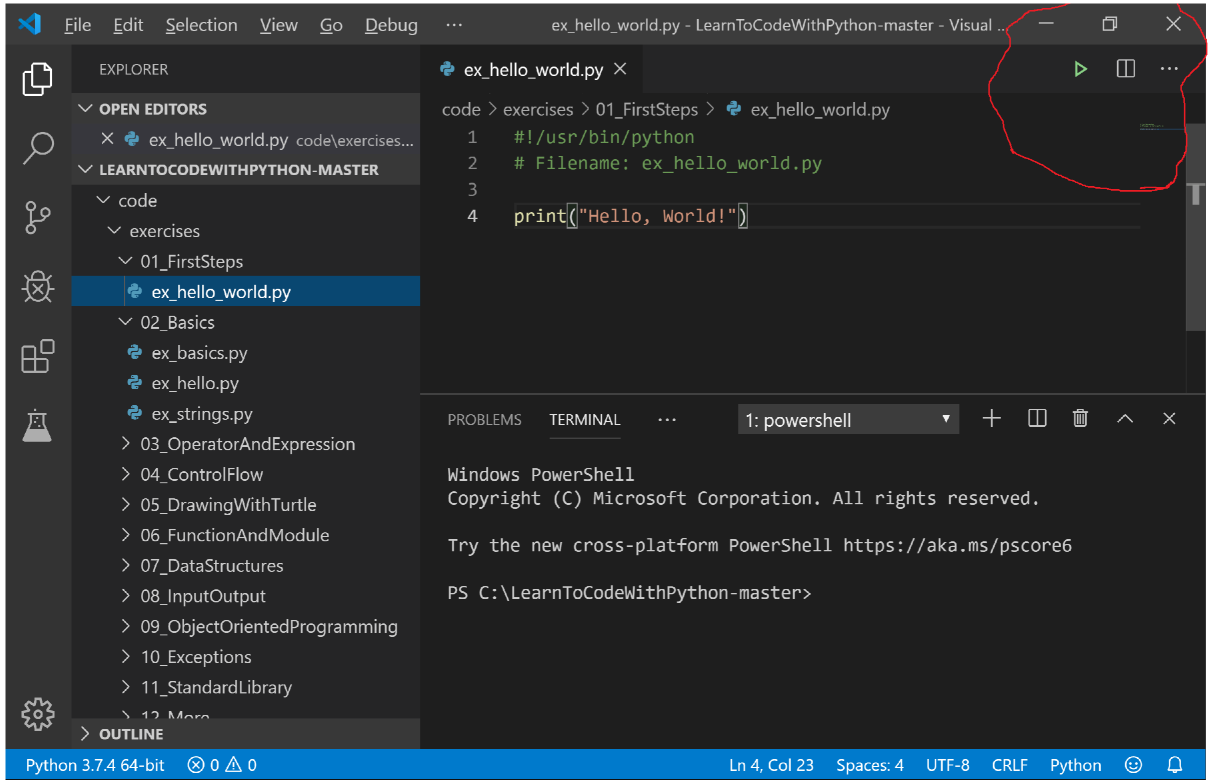


```python
#!/usr/bin/python
# Filename: ex_hello_world.py

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

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\LearnToCodeWithPython-master>
```

Install Visual Studio Code #3

- Install Python Extension

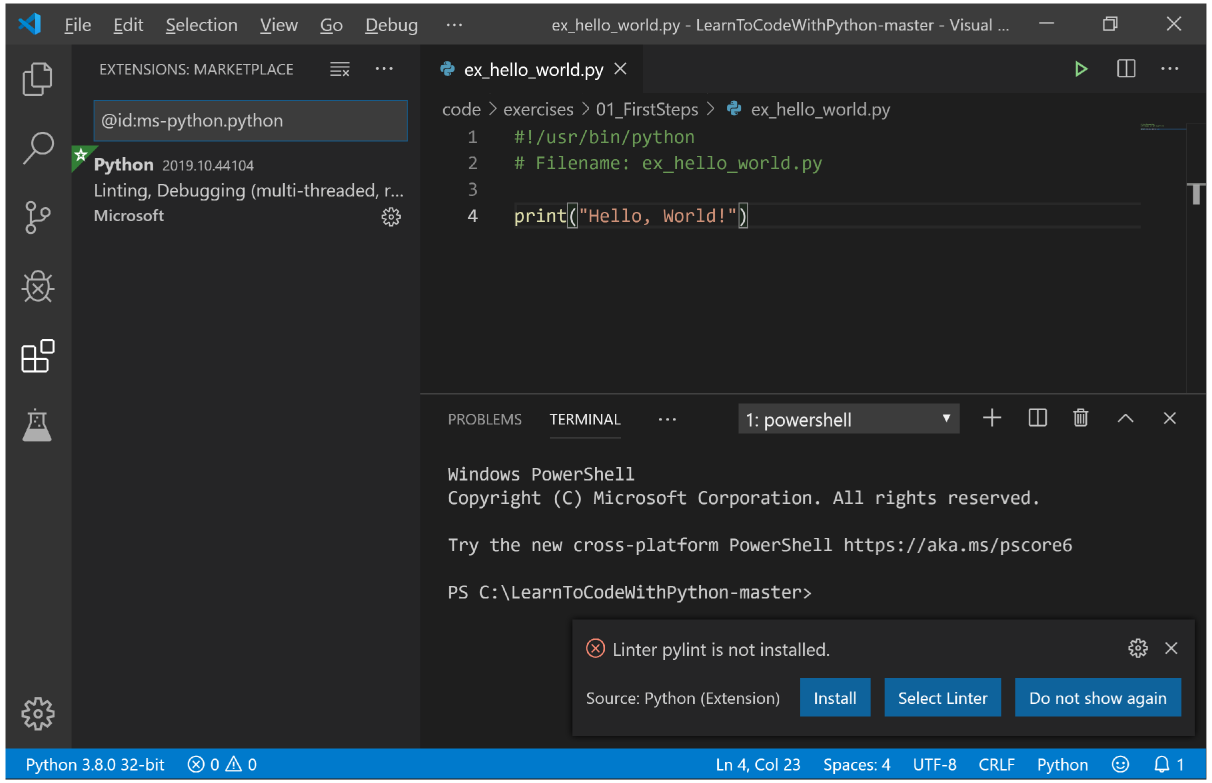


```python
#!/usr/bin/python
# Filename: ex_hello_world.py

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

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\LearnToCodeWithPython-master>
```

Install Visual Studio Code #4

- Install pylint Extension



```python
#!/usr/bin/python
# Filename: ex_hello_world.py

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

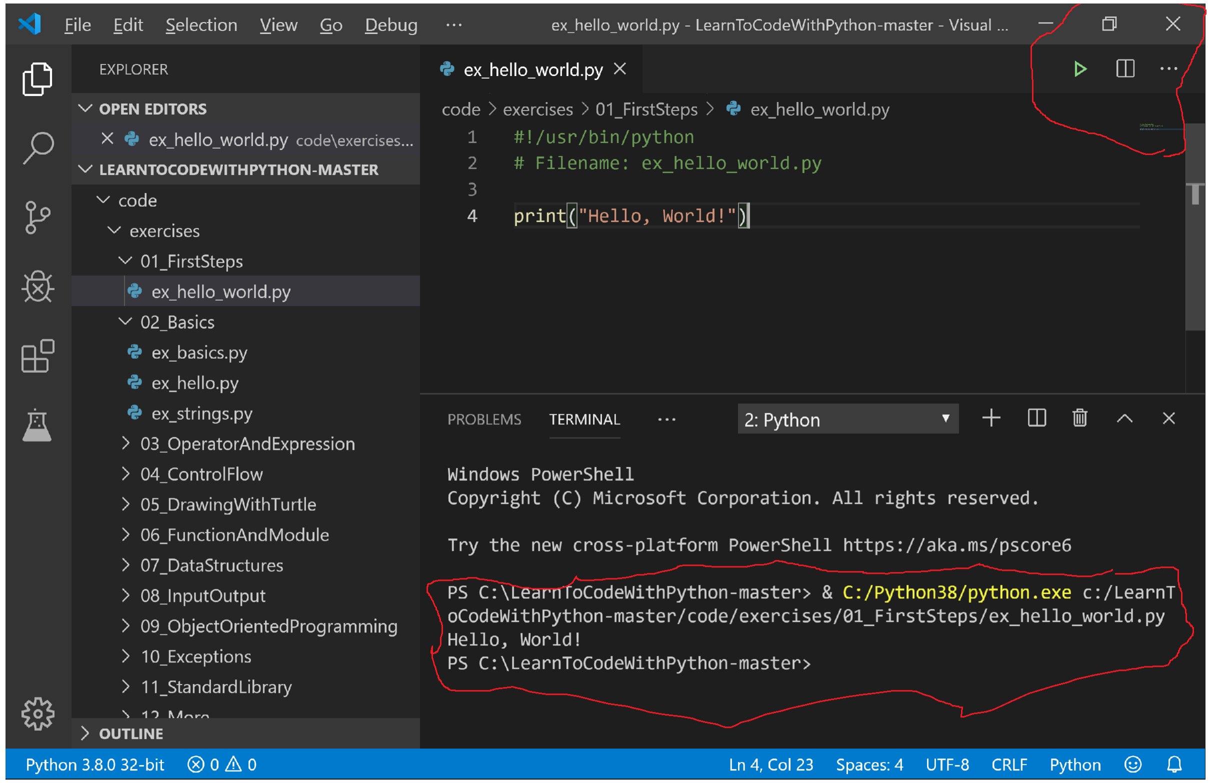
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\LearnToCodeWithPython-master>
```

Linter pylint is not installed.

Source: Python (Extension)

Install Visual Studio Code #5

- Run Python program


File Edit Selection View Go Debug ···
ex_hello_world.py - LearnToCodeWithPython-master - Visual ...
EXPLORER
OPEN EDITORS
ex_hello_world.py code\exercises...
LEARNTOCODEWITHPYTHON-MASTER
code
exercises
01_FirstSteps
ex_hello_world.py
02_Basics
ex_basics.py
ex_hello.py
ex_strings.py
03_OperatorAndExpression
04_ControlFlow
05_DrawingWithTurtle
06_FunctionAndModule
07_DataStructures
08_InputOutput
09_ObjectOrientedProgramming
10_Exceptions
11_StandardLibrary
12_More
OUTLINE
code > exercises > 01_FirstSteps > ex_hello_world.py
PROBLEMS TERMINAL ···
2: Python
Python 3.8.0 32-bit ⊗ 0 ⚠ 0
Ln 4, Col 23 Spaces: 4 UTF-8 CRLF Python

```python
#!/usr/bin/python
# Filename: ex_hello_world.py

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

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\LearnToCodeWithPython-master> & C:/Python38/python.exe c:/LearnT
oCodeWithPython-master/code/exercises/01_FirstSteps/ex_hello_world.py
Hello, World!
PS C:\LearnToCodeWithPython-master>
```

First Steps

- Start python.exe at the command line OR
- Using Python Interactive Shell
  - Windows Key -> type “IDLE (Python …).”
  - Enter Python statements and press ENTER to run.

```
print(‘Hello, World!’)
```

- Congratulations! You’ve just created your first Python program!
- Getting Help
  - help(print)

Using a Program Source File

- Creating a Program Source File via Python IDLE
  - File -> New Window then File -> Save
  - Create a new directory: C:\python\exercises
  - File name: hello_world.py

```
#Filename: hello_world.py
print('Hello, World!')
```

- Running the program
  - Using IDLE
    - Run -> Run Module or keyboard shortcut F5.
  - Command line
    - > python hello_world.py

For Windows Users

- Open Terminal
  - Windows -> Run -> type “cmd”
  - python -V
- Open IDLE
  - Windows Key -> type “IDLE (Python …).”
- Open Visual Studio Code
  - Windows Key -> type “visual studio code”

For Mac OS X Users

- Open Terminal
  - Open the terminal by pressing Command+Space keys (to open Spotlight search), type Terminal and press enter.
  - python –V
- Open IDE

Getting Familiar with Visual Studio Code

- Change Setting: Code->Preferences->Settings
  - Line Numbers: on
  - Tab Size->4
  - Insert Spaces: check
- Create Project: File->New Project...
  - Location: C:\PythonWork
  - Interpreter: C:\Python38

Python Crash Course

Python Crash Course

- Basics
- Operators and Expressions
- Control Flow
- Functions and Modules
- Data Structures
- Inputs & Outputs
- Object Oriented Programming
- Exceptions
- Standard Library

Basics

Basics Data Types

- Built-In types
  - Numbers
    - integers: 2
    - floating point (float): 3.23
    - complex numbers: (-5+4j)
    - Literal Constants: use its value literally, it’s constant
  - number: 5, 1.23, 9.25e-3
  - String: ‘Hello World!’
  - Lists: [1, 2, 3, 4, 5]
  - Tuples: (1, 2, 3, 4, 5)
  - Dictionaries: { 1: “one”, 2: “two” }
  - File Objects: f = open(“myfile”, ”w”)…
- User defined types using classes and objects

Numbers

```
# Python’s four number types:
# Integers: 1, 2
# Floats: 3.7
# Complex numbers: 3 + 2j
# Booleans: True, False

X = 5 + 2 – 3 * 2
5 / 2 # float
5 // 2 # truncation
2 ** 8
(3+2j) * (4+9j)
Round(3.49)
X = True
```

Numbers and Math

- Numbers in Python are of three types - integers, floating point and complex numbers.

```
spam = 10
fp = 10.50
```

- Math

2 + 2

```
import math
degrees = 45
radians = degrees / 360.0 * 2 * math.pi
```

Comments

Comment, # symbol: anything to the right of the # symbol is a comment.

```
# Computer programming is fun!
# Python programming is very fun!

# print(2+2)
```

- Block comments

```
# This is a Block comment
```

- Inline comments

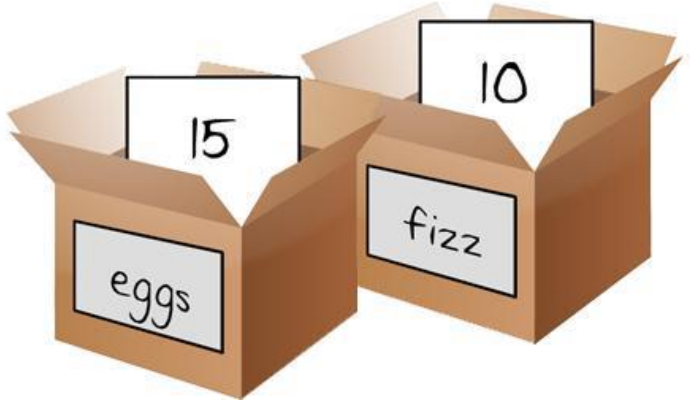
```
x = 5 + 1 # This is an Inline comment
```

Variables and Types

Variables stores some information and their value can vary.

- Variables and Values

```
fizz = 10
eggs = 15 print(fizz)
spam = fizz + eggs
print(spam)
```



- Values Have Types

– use the type() function to find out the type

```
type(spam)
<class 'int'>
```

Naming Variables

- The first character of the identifier must be a letter of the alphabet or an underscore (‘_’).
- The rest of the identifier name can consist of letters, underscores (‘_’) or digits (0-9).
- Identifier names are case-sensitive. myname and myName are not the same.
- Valid identifiers: i, name_23, _my_name

Strings

- A string is a sequence of characters enclosed either single or double quote marks. Strings are Immutable.
- Single Quotes

```
msg = 'Quote me on this'
```

- Double Quotes work exactly as single quotes

```
msg = "What’s your name?"
```

- Triple Quotes (""" or ‘’’)

msg = ‘’’This is the first line. This is the second line.
'''

Strings

```
# String processing is one of Python’s strengths.
# operators (in, +, *), built-in functions (len, max)
“A string”
‘A string’
“””A string”””
‘’’A string```
x = “live and  let \t.  \t live”
len(x)
x.split()
x.replace(“  let \t.  \t live”, “enjoy life”)
print(x)
```

Escape Sequences

```
'What\'s your name?’
“Newlines are indicated by \n”
```

- Raw Strings

```
r”Newlines are indicated by \n”
R”Newlines are indicated by \n”
```

Strings Concatenation

Concatenate Strings

```
'Hello, ' + 'World!’
```

Two string literals side by side, they are automatically concatenated by Python

```
'What\'s ' 'your name?’
```

Multiplying Strings

```
print(10 * 'a')
```

Strings Format

```
The format Method: construct strings from other information
age = 25 name = 'John’
print(name + " is " + age + " years old.")
print(name + " is " + str(age) + " years old.")
print("%s is %d years old." % (name, age))
print("{0} is {1} years old.".format(name, age))
```

Indentation!!!

- Indentation: leading whitespace (spaces or tabs) at the beginning of the statement.
  - to determine the grouping of statements.
- Statements which go together must have the **same indentation**. Each such set of statements is called a **block**.
- Do not use a mixture of tabs and spaces for the indentation.
- Use four spaces or one tab for each indentation level.

```
while True:
    •print('Hello, World!')
```

More About Strings

- Strings are also objects and have methods which do everything from checking part of a string to stripping spaces!
- The strings that you use in program are all objects of the class str.
- For a complete list of such methods,
  - see help(str).
- Code: ex_str_methods.py

```
name = 'Python’
if name.startswith('Py’):
    print('Yes, the string starts with “Py"')
```

Operators and Expressions

Operators and Expressions

- Expression can be broken down into operators and operands.
- Operators:
  - Numerical: +, -, *, /,**, //, %
  - Bitwise: <<, >>, &, |, ^, ~
  - Comparison: <, >, <=, >=, ==, !=
  - Boolean: not, and, or

3 + 5

2 * 3

Order of operations

- Order of operations (**PEMDAS**)
  - **P**arentheses: 2 * (3 – 1)
  - **E**xponentiation: 3*1**3
  - **M**ultiplication and **D**ivision: 2 * 3 -1; 6 + 4 / 2 – **A**ddition and **S**ubtraction: 2 + 2 - 1
- Operators with the same precedence are evaluated from left to right (except exponentiation).
- Changing the Order of Evaluation: use parentheses

```
(2 + 3) * 4
```

What Will You Get?

```
2 + 3 * 4 – 6 / 2
(2 + 3) * 4 – 6 / 2
2 + (3 * 4) – (6 / 2)
2 + 3 * (4 – 6) / 2
```

Control Flow

Booleans

- Boolean: True or False

```
1 == 1
1 == 2
```

- Converting Between Data Types
  - int(), float(), str()

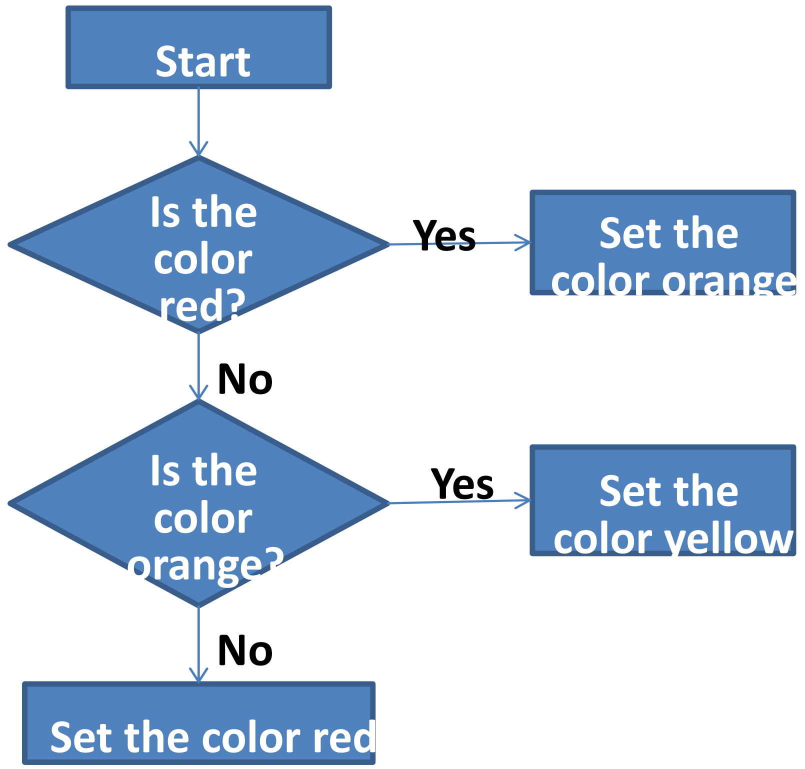
```
print(int(3.9))
print(float("Three point two"))
```

Conditions

Symbol	Definition
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The if Statement

```
color = 'white’

if color == 'red': color = 'orange'
elif color == 'orange': color = 'yellow'
else:
    color = 'red'
```



The break Statement

- Breaking out of a loop statement

```
while True:
    s = (input('Enter something: ‘))
    if s == 'quit’:
        break
print('Length of the string is ', len(s))
```

The continue Statement

- Skip the rest of statements in the current loop block and continue to the next iteration of the loop.

```
while True:
    s = (input('Enter something: ‘))
    if s == 'quit’:
        break
    if len(s) < 3:
        print('Too short: ', len(s))
        continue
    print('Input is too long: ', len(s))
```

The while Statement

```
running = True
while running:
    print(“Python is fun!”)
step = 0 while step < 10:
    print("%s: Python is awesome!" % step)
    step += 1
```

For Loop 1

Print multiple times:

```
print('hello’)
print('hello’)
print('hello’)
print('hello’)
print('hello')
```

For Loop 2

```
Using for Loops:

for x in range(0, 5):
    print('hello')
```

Functions and Modules

Functions

Functions are reusable pieces of programs.

- ex_function.py

Function Parameters

- ex_func_param.py

```
def functionName(params):
    block of statements
def printMax(a, b):
    if a > b:
        print(a, ' is maximum’)
    elif a == b:
        print(a, ' is equal to', b)
    else:
        print(b, ' is maximum’)
printMax(3, 4)
```

Default Argument Values

Parameter name with assignment operator (=) followed by the default value.

```
def say(message, times = 1):
    print(message * times)


#
say('Hello’)
say('World', 5)
```

Keyword Arguments

```
def func(a, b=5, c=10):
    print('a is ', a, ' and b is ', b, ' and c is ', c)

#
func(1,2,3)
func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

The return Statement

return from a function, can optionally return a value.

```
def maximum(x, y):
    if x > y: return x
    else:
        return y
#
print(maximum(2, 3))
```

The pass statement: an empty block

```
def someFunction():
    pass
```

Data Structures

Data Structures

- Python has built-in, dynamic data structures or collections
  - You basically never need to define your own
- Lists (approx. homogenous arrays)
- Tuples (approx. heterogeneous arrays)
- Dictionaries (key-value mappings)
- Sets (unique values)

Lists are Variable Length, Same Types

- List holds an ordered collection of items enclosed in square brackets.
- Lists are **mutable**, and their elements are usually **homogeneous** and are accessed via iteration.

```
shoplist = ['apple', 'mango', 'carrot', 'banana'] for item in
shoplist:
    print(item, end=' ‘)


print('My shopping list is: ', shoplist)
shoplist[0] = 'strawberry'
```

Lists are Variable Length, Same Types

```
names = ["Matthew", "Mark", "Luke"]
names.append("John")

# Zero-indexed (but something better is coming up)
names[0]

# Convert other sequences to a list
x = list(y)
x = list(names)
```

List Arithmetic

```
list1 = [1, 2, 3, 4]
list2 = ['I ate chocolate', 'and want more’]
list3 = list1 + list2
print(list3)
```

Tuples are Fixed-Length, Different Types

Tuples are **immutable**, and their elements are usually **heterogeneous** and are accessed via unpacking or indexing.

```
shoptuple = ('apple', 'mango', 'carrot', 'banana') for item in shoptuple:
    print(item, end=' ‘)
    print('My shopping list is: ', shoptuple)


#shoptuple[0] = 'strawberry'
```

Tuples are Fixed-Length, Different Types

```
person = ("Simon", 180.0, 35)
person[0] person[1]

# Convert a sequence to a tuple
x = tuple(y)
x = tuple(person)
```

Dictionaries Map Keys to Values

```
Dictionary is like an address-book and pair of key and value.
Key must be unique.

ab = { 'John' : 'jdoe@acme.com', 'Tony' : 'tsmith@acme.com'}
ab['John']
ab['John'] = 'jdoe@live.com'
```

Dictionaries Map Keys to Values

```
x = { "Rank": 7, "Score": 93.4 }
"Rank" in x
x["Rank"]

# Try x.keys(), x.values(), and x.items()
```

Sequences

```
strings
Lists
Tuples

shoplist = [‘apple’, ‘mango’, ‘carrot’, ‘banana’]
print(shoplist[0])
print(shoplist[-1])
print(shoplist[1:3])
print(shoplist[1:-1])
print(shoplist[2:])
print(shoplist[:])
```

Sets Contain Unique Values

Sets are **unordered** collections of simple objects.

```
x = {1, 3, 3, 5, 7}
# or set([1, 3, 3, 5, 7])
3 in x
x

len(x)
```

Inputs & Outputs

Input from User

- Input from User

```
something = input('Enter text: ‘)
print(something)
```

Files

Reading from a file

code: ex_using_file.py

```
f = open('myinput.py’)
while True:
    line = f.readline()
    if len(line) == 0: # EOF
        break
    print(line, end=’’)

f.close()
```

Writing to a File

Writing to a file

```
f = open('myoutput.txt', 'w')
f.write('Hello, World!')
f.close()
```

Object Oriented Programming (OOP)

The Characteristics of OOP

- Objects
  - We live in an object-oriented world.
  - An object is a structure for incorporating data and methods/functions/procedures for working with that data.
- Abstraction
  - high level of an object
- Encapsulation
  - no direct access to the data is granted
- Polymorphism
  - two different objects have the same methods, e.g., print()
- Inheritance
  - parent and child
- Aggregation
  - an object consists of a composite of other objects

Object Oriented Programming

- Object Oriented Programming paradigm: combines data and functionality and wrap it inside something called an object.
- Python is strongly object-oriented in the sense that everything is an object including numbers, strings and functions.
- **Classes** and **objects** are the two main aspects of object oriented programming.

Classes and Objects

- A class creates a new type where objects are instances of the class.
- Python refers to anything used in a program as an object.
- Objects can
  - store data using variables or **fields**
- **instance/object variables**: variables belong to each instance/object of the class
- **class variables**: variables belong to the class itself
  - have functionality by using functions or **methods**

Classes and Functions

• Objects with Classes

```
class Person():
    def __init__(self, name, age):
        self.name = name self.age = age

    def birthday(self):
        self.age = self.age + 1
#
ben = Person("Ben", 10)
print(ben.name, ben.age)
```

Self

- Class methods have only one specific difference from ordinary functions - they must have an extra first name that has to be added to the beginning of the parameter list, but you **do not** give a value for this parameter when you call the method, Python will provide it.
- This particular variable refers to the object *itself*, and by convention, it is given the name **self**.

The __init__ Method

- The __init__ method is run as soon as an object of a class is instantiated and it does any initialization.
- code: ex_person.py

Object Methods

- classes/objects can have methods just like functions except that we have an extra self variable.
- code: ex_method.py

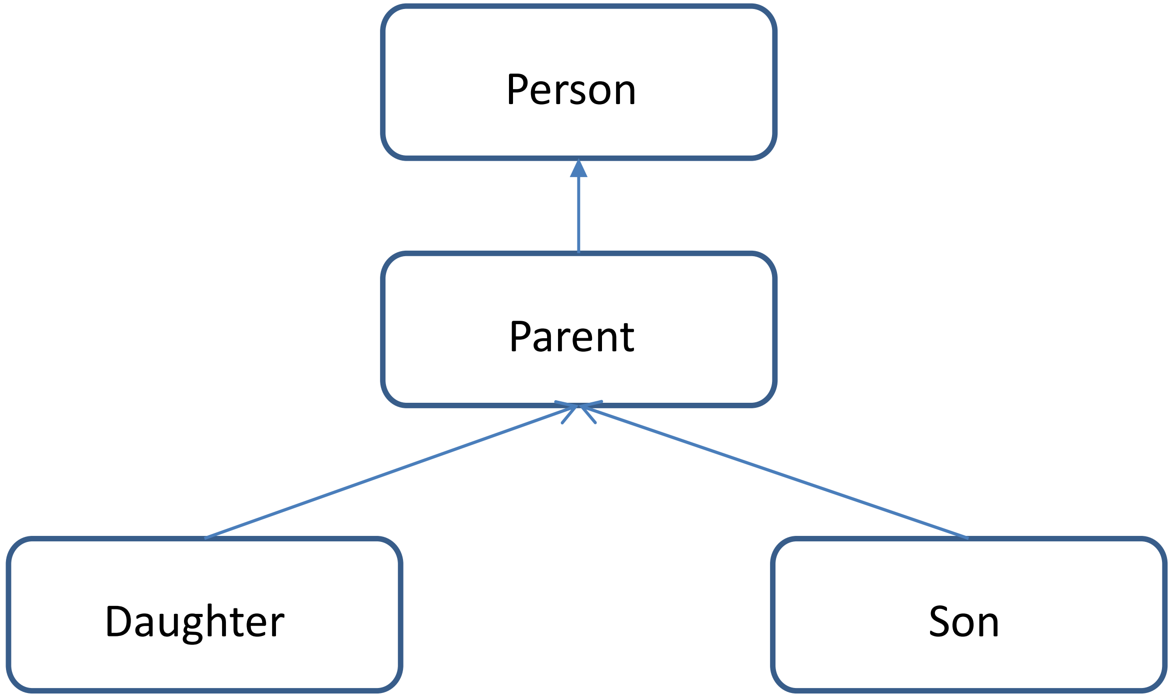
Class and Object Variables

- There are two types of fields (variables):
  - Class variables: variables belong to the class itself. They are shared and can be accessed by all instances of that class. There is only one copy of the class variable.
  - Object/Instance variables: variables belong to each instance/object of the class. Each object has its own copy of the field. They are not shared.
- code: ex_objvar.py

Inheritance

- One of the major benefits of object oriented programming is **reuse** of code and one of the ways this is achieved is through the *inheritance* mechanism. Inheritance can be best imagined as implementing a *type and subtype* relationship between classes.
- polymorphism: where a sub-type can be substituted in any situation where a parent type is expected i.e. the object can be treated as an instance of the parent class.
- code: ex_inherit.py

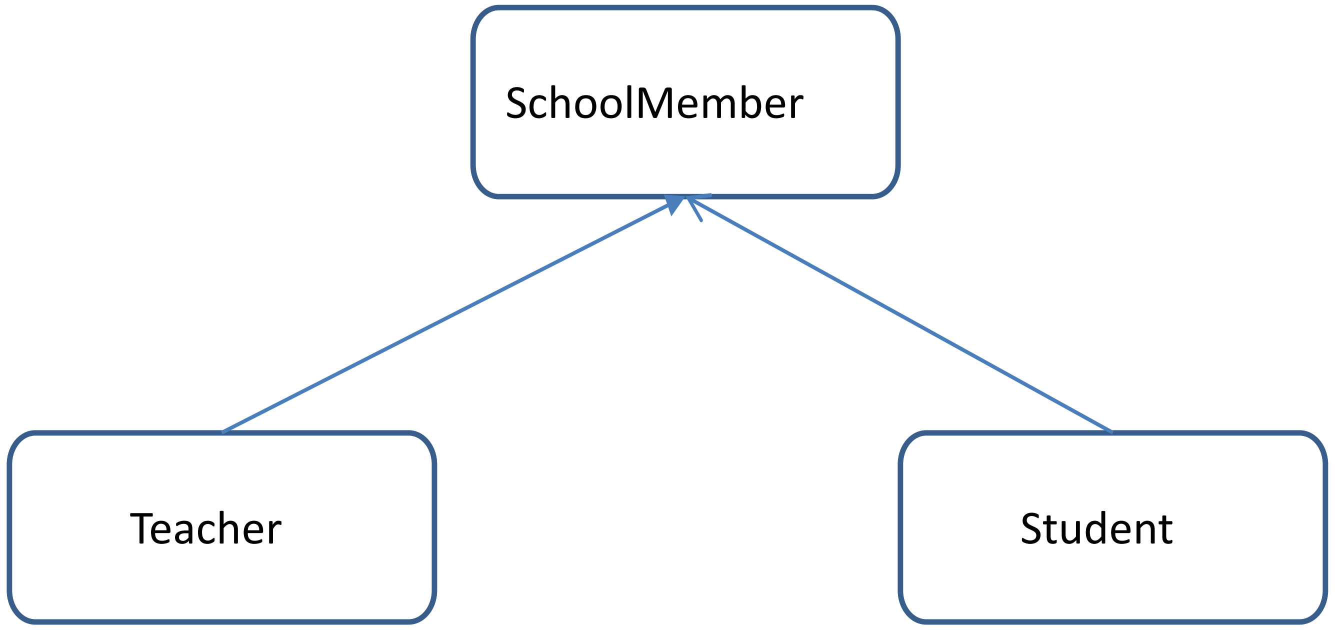
Derived Classes Example



Derived Classes Example

```
Derived Classes
code: ex_parent_child.py

class Parent(Person):
    def __init__(self, name, age):
        Person.__init__(self,name,age)
        self.children = []
    def add_child(self,child):
        self.children.append(child)
    def print_children(self):
        print(“The children of “, self.name, “ are:”)
        for child in self.children:
            print(child.name)
```

Inheritance Example



Exceptions

Exceptions

- Exceptions occur when certain *exceptional* situations occur in your program.

Errors

```
Print('Hello World')
print('Hello World')
```

Handling Exceptions

```
Handling Exceptions
– code: ex_try_except.py

try:
    text = input(‘Enter something: ‘)
except EOFError:
    print(‘It was an EOF’)
else:
    print(‘You entered {0}’.format(text))
```

Raising Exceptions

- Raising Exceptions
- The error or exception that you can raise should be class which directly or indirectly must be a derived class of the Exception class.
- code: ex_raising.py

raise EOFError()

Try ..Finally

Try ..Finally

- code: ex_finally.py

```
try:
    # do something
except EOFError:
    # process the exception
finally:
    # clean up
```

With Statement

- Acquiring a resource in the try block and subsequently releasing the resource in the finally block is a common pattern. Hence, there is also a with statement that enables this to be done in a clean manner.
- code: ex_using_with.py

```
with open(“poem.txt”) as f:
    for line in f:
        print(line, end=‘’)
```

Standard Library

Standard Library

- The Python Standard Library contains a huge number of useful modules and is part of every standard Python installation.
  - https://docs.python.org/3/library/index.html
- Some of the commonly used modules in the
- Python Standard Library
  - sys
  - logging
  - urlib
  - json

sys module

- The sys module contains system-specific functionality.
- We have already seen that the sys.argv list contains the command-line arguments.
- code: ex_versioncheck.py

logging module

- logging module displays debugging messages.
- code: ex_use_logging.py

urlib and json modules

- Use urllib and json modules to access the web.
- code: ex_yahoo_search.py

Special Methods

- Special Methods
  - __init__
  - __del__
  - __str__
  - etc.
- Single Statement Blocks

```
if flag:
    print(‘Yes’)
```

Class Hands-On Projects

Team Members

- Three members per team
- Team Discussion
  - Discuss the strategy of solving the problem
  - Design the solution
  - Pick the member roles
- Member Roles
  - Developer 1: tell the coder what to write
  - Developer 2: type in the code and check the code
  - Tester 1: test the code to ensure the quality

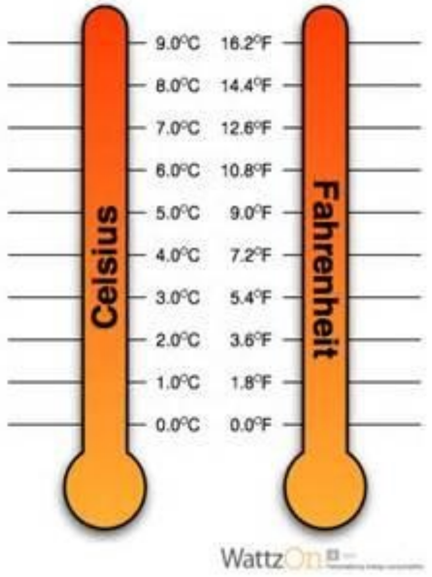
Guess the Number

- Guess the Number: http://inventwithpython.com/chapter4.html
- You're going to make a "Guess the Number" game. The computer will think of a random number from 1 to 20, and ask you to guess it. The computer will tell you if each guess is too high or too low. You win if you can guess the number within six tries.
- Source code: guess.py

Temperature Converter

- Convert Celsius to Fahrenheit
  - fahrenheit = (celsius * 9 / 5) + 32
- Convert Fahrenheit to Celsius
  - celsius = (fahrenheit - 32) * 5 / 9
- Temperature Converter
  - Enter celsius
    - output celsius, fahrenheit
  - Enter celsius
    - output fahrenheit, celsius

- Generic Temperature Converter
  - Enter “c2f” & celsius, output fahrenheit
  - nter “f2c” & fahrenheit, output celsius



Drawing the Turtles

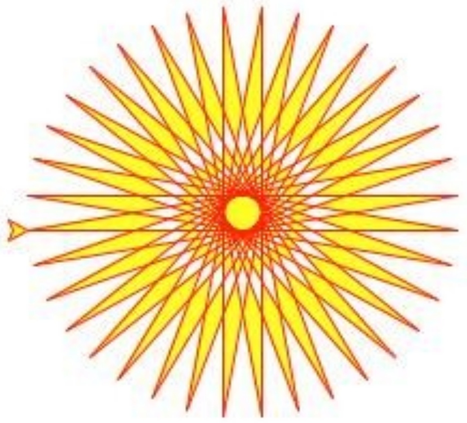
Drawing with Turtles

- A turtle in Python is sort of like a turtle in the real world.
- In the world of Python, a turtle is a small, black arrow that moves slowly around the screen.
- The turtle is a nice way to learn some of the basics of computer graphics.
- Python has a special module called turtle that we can use to learn how computers draw pictures on a screen.

Using Python's turtle Module

- A module in Python is a way of providing useful code to be used by another program.
- Turtle can draw intricate shapes using programs that repeat simple moves.
- Turtle graphics for Tk
  - https://docs.python.org/3/library/turtle.html
- tkinter package

```
import turtle
```



Using Python's turtle Module

- Pixel is a tiny, square dot on the computer monitor.
- t.forward(50) # advance 50 pixels (or backward)
- t.left(90) # turn left 90 degrees (or right)
- t.reset() # clears the canvas and put the turtle back at the starting position
- t.clear() # clears the screen and leaves the turtle where it is
- t.up() # pick up the pen and stop drawing
- t.down() # put the pen back down and start drawing again

Turtle Program

1. Show the Turtle
2. Can you draw a petal?
3. Can you draw a circle?
4. Can you draw 4 circles?
5. Can you draw all 24 petals?

1. Show a Turtle

```
import turtle
window = turtle.Screen()
t = turtle.Turtle()
window.exitonclick()
```

2. Can you draw a petal?

```
#draw a petal
import turtle
window = turtle.Screen()
t = turtle.Turtle()
t.left(15)
t.forward(50)
t.left(157)
t.forward(50)
window.exitonclick()
```

3. Can you draw a circle?

```
import turtle
window = turtle.Screen()
t = turtle.Turtle()
t.left(90)
t.forward(100)
t.right(90)
t.circle(10)
window.exitonclick()
```

4. Can You draw 4 circles?

```
# Hint: for x in range(0, 3):
import turtle
window = turtle.Screen()
t = turtle.Turtle()
t.left(90)
t.forward(100)
t.right(90)
t.circle(10)
window.exitonclick()
```

4. Can You draw 4 circles? (with loop)

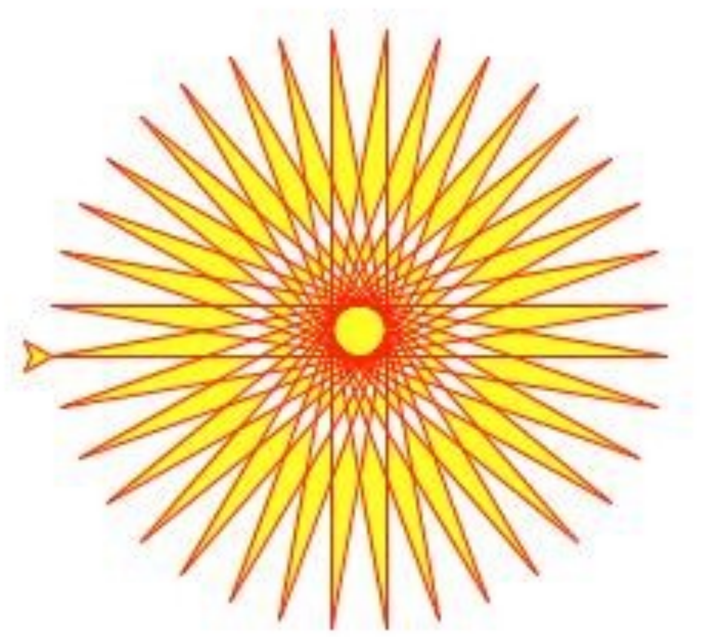
```
# Hint: for x in range(0, 3):
import turtle
window = turtle.Screen()
t = turtle.Turtle()
for x in range(0, 4):
    t.left(90)
    t.forward(100)
    t.right(90)
    t.circle(10)
    t.right(90)
window.exitonclick()
```

5. Can you draw all 24 petals?

```
#draw all petals
import turtle
window = turtle.Screen()
t = turtle.Turtle()
for i in range(1,24):
    t.left(15)
    t.forward(50)
    t.left(157)
    t.forward(50)

window.exitonclick()
```

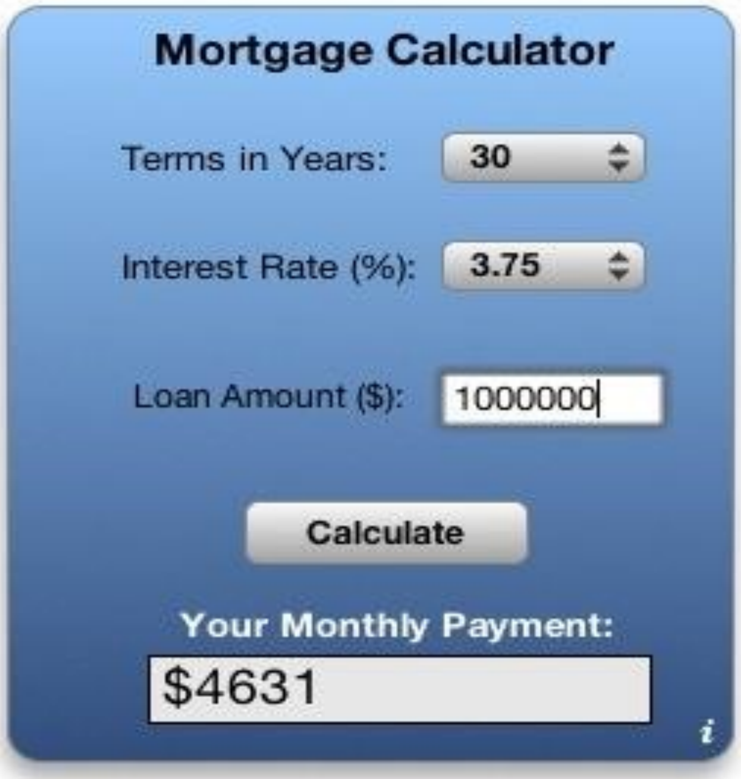
Drawing with Turtle

```
from turtle import *
color('red', 'yellow’)
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```



Mortgage Loan Calculator

- Mortgage Loan Calculator
- M = P [i (1 + i)^n] / [(1 + i)^n – 1]
    - M: monthly payment
    - P: principal balance
    - i: monthly interest rate
    - n: number of monthly payments
    - ^: power


Mortgage Calculator

Terms in Years: 30

Interest Rate (%): 3.75

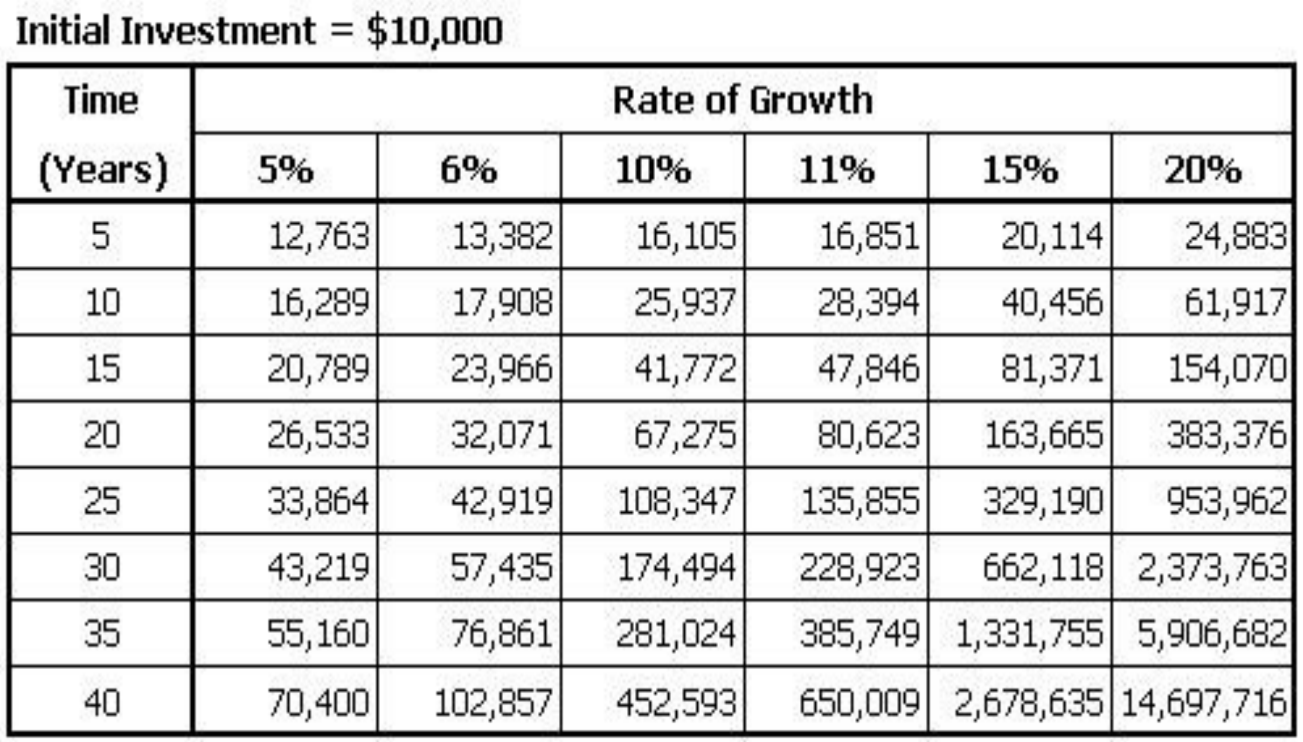
Loan Amount ($): 1000000

Your Monthly Payment:

$4631

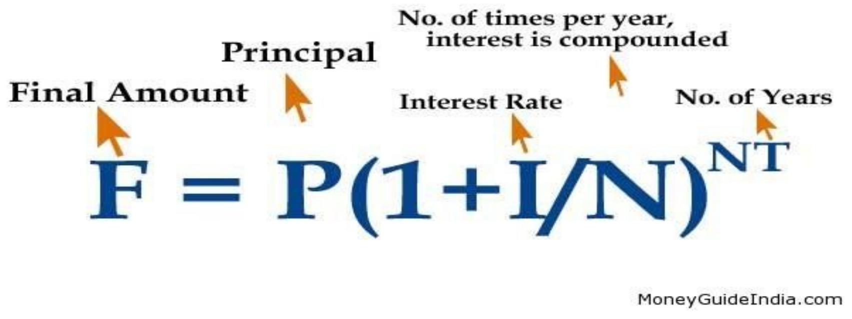
Compound Interest

- Compound Interest



Initial Investment = $10,000

Time (Years)	Rate of Growth					
	5%	6%	10%	11%	15%	20%
5	12,763	13,382	16,105	16,851	20,114	24,883
10	16,289	17,908	25,937	28,394	40,456	61,917
15	20,789	23,966	41,772	47,846	81,371	154,070
20	26,533	32,071	67,275	80,623	163,665	383,376
25	33,864	42,919	108,347	135,855	329,190	953,962
30	43,219	57,435	174,494	228,923	662,118	2,373,763
35	55,160	76,861	281,024	385,749	1,331,755	5,906,682
40	70,400	102,857	452,593	650,009	2,678,635	14,697,716



$$F = P(1+I/N)^{NT}$$

Currency Converter

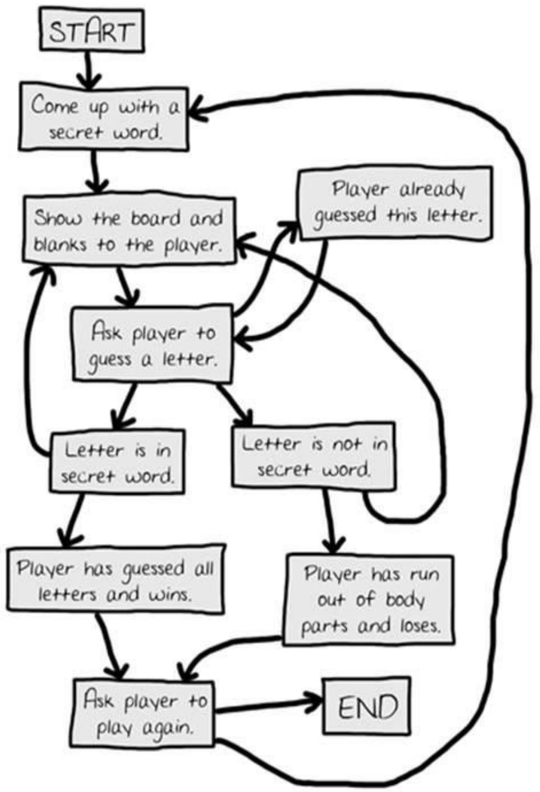
- Currency Converter

Hangman

- Hangman:
  - http://inventwithpython.com/chapter9.html
  - Flow Chart: http://inventwithpython.com/chapter8.html
  - Source code: hangman.py
- Hangman is a game for two people usually played with paper and pencil. One player thinks of a word, and then draws a blank on the page for each letter in the word. Then the second player tries to guess letters that might be in the word.
- If they guess correctly, the first player writes the letter in the proper blank. If they guess incorrectly, the first player draws a single body part of the hanging man. If the second player can guess all the letters in the word before the hangman is completely drawn, they win. But if they can't figure it out in time, they lose.

Flow Chart for Hangman

- Flow Chart for Hangman


START
Come up with a secret word.
Show the board and blanks to the player.
Player already guessed this letter.
Ask player to guess a letter.
Letter is in secret word.
Letter is not in secret word.
Player has guessed all letters and wins.
Player has run out of body parts and loses.
Ask player to play again.
END

Thank You!

- Thanks You!

What Next?

What Next?

- PyGame
- Raspberry Pi
- Minecraft Mods with Python
- Web Development with Django
- Data analysis, machine learning, etc.
- System admin, engineering, finance, etc.
- **Example Code**
- The best way to learn a programming language is to write a lot of code and read a lot of code:
- The PLEAC project
- Python Cookbook is an extremely valuable collection of recipes or tips on how to solve certain kinds of problems using Python. This is a must-read for every Python user.
- Microsoft YouthSpark
- Find Your Passion
- Girls Who Code
- AP Computer Science (College Board)
  - AP Computer Science A
  - AP Computer Science Principles (coming fall 2016)

Supplemental Materials

Variables and Values

- Python has strong, dynamic typing
- Values never change type unless you make them
  - This is the "strong" part
- Variables change type whenever you assign to them
  - This is the "dynamic" part

Keeping Text in Strings

```
"abcde"[0]
"abcde"[1]
"abcde"[10]
"abcde"[1:3]
"abcde"[:3] "abcde"[3:]
len("abcde") "abc" + "def"
```

Local Variables

Scope of the variable

- ex_func_local.py

```
x = 50
def func(x):
    print('x is ', x)
    x = 2
    print('Changed local x to ', x)

func(x)
print('Value of x is ', x)
```

Global Variables

```
global statement
```

- ex_func_global.py

```
x = 50
def func():
    global x
    print('x is ', x)
    x = 2
    print('Changed local x to ', x)

func(x)
print('Value of x is ', x)
```

nonlocal Variables

nonlocal scopes are observed when you define functions inside functions.

- ex_func_nonlocal.py

```
def func_outer():
    x = 2
    print('x is ', x)
    def func_inner():
        nonlocal x
        x = 5

    func_inner()
    print('Changed local x to ', x)
func_outer()
```

DocStrings

- Python has a nifty feature called documentation strings, usually referred to by its shorter name docstrings.

Modules

- How to create modules that contains functions and variables?
  - create a file with a .py extension
  - write the modules in the native language
- A module can be imported by another program.

```
import sys
```

Create a Modules

- Create a module named **first_module.py**

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

- Create another file named **first_hello.py**

```
import first_module
```

```
Hello World!
```

Modules

- Create a second_module.py

```
def hello():
    print("Hello World!")
```

- Calling the function from hello2.py

```
import second_module
second_module.hello()

from second_module
import hello
hello()
```

A module’s __name__

- A module’s __name__
  - Each module has a name and statement in a module can find out the name of its module.

```
if __name__ = '__main__’:
    print('This program is being run by itself')
else:
    print('I am being imported from another module')
```

Lists

```
x = []
x = [1, 2, 3, 4, 5]
y = [“first”, “second”, “third”]
z = [1, ”two”, 3, [“a”, “b”]] # different types
x[0] # index from the front
x[-1] # index from the back
x[0:3] # slice [m:n]
x[1] = 10 # change
len(x)
x.reverse()
```

Tuples

```
# Tuples are similar to lists but are immutable (can’t be changed).
x = ()

# practice with the same examples of lists

x = [1, 2, 3, 4]
tuple(x)
y = (1, 2, 3)
list(y)
```

Dictionaries

```
# Dictionaries, maps, or associative array implemented by using hash tables.
# key must be of an immutable type: numbers, strings, and tuples.
x = {1: “one”, 2: “two”} # key, value
x[“first”] = “one”
list(x.keys())
x[1]
x.get(1, “not available”)
x.get(4, “not available”)
```

Sets

```
# sets is an unordered (unique) collection of objects.
z = [1, 2, 3, 1, 3, 5]
#x = set([1, 2, 3, 1, 3, 5])
x = set(z)
1 in x
4 in x
```

References

- When you create an object and assign it to a variable, the variable only refers to the object and does not represent the object itself!

```
shoplist = [‘apple’, ‘mango’, ‘carrot’, ‘banana’]
mylist = shoplist del shoplist[0]
print(‘shoplist is: ‘, shoplist)
print(‘mylist is: ‘, mylist)
#
mylist = shoplist[:] del mylist[0]
print(‘shoplist is: ‘, shoplist)
print(‘mylist is: ‘, mylist)
```

Pickle

- Python provides a standard module called **pickle** using which you can store **any** Python object in a file and then get it back later. This is called storing the object ***persistently***.
- code: ex_pickling.py

More

- Passing tuples Around

```
def get_error_details():
    return(2, ‘second error details’)

errnum, errstr = get_error_details()
```

Lambda Forms

- Lambda Forms: to create new function objects and return them at runtime

```
def make_repeater(n):
    return lambda s: s * n

twice = make_repeater(2)
print(twice(‘word’))
print(twice(5))
```