Python Bootcamp

Day 1

Angel, Laura, Laurel, MinhQuan

Running Python Code/Script

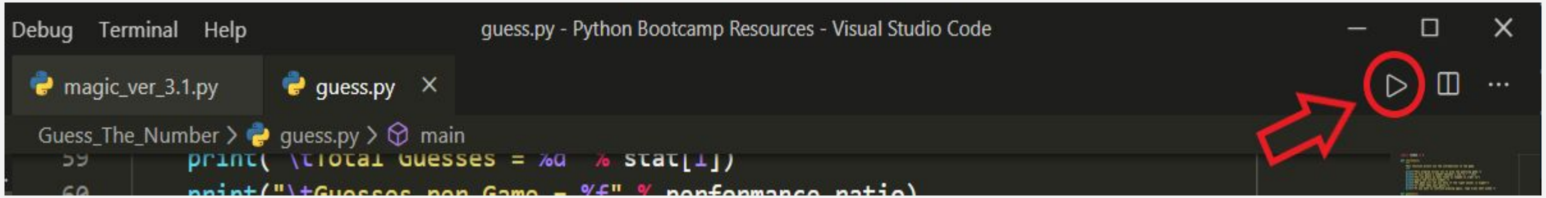
Need to install Python 3 first

There are two ways:

- Using a IDE/Text Editor
- Running in terminal

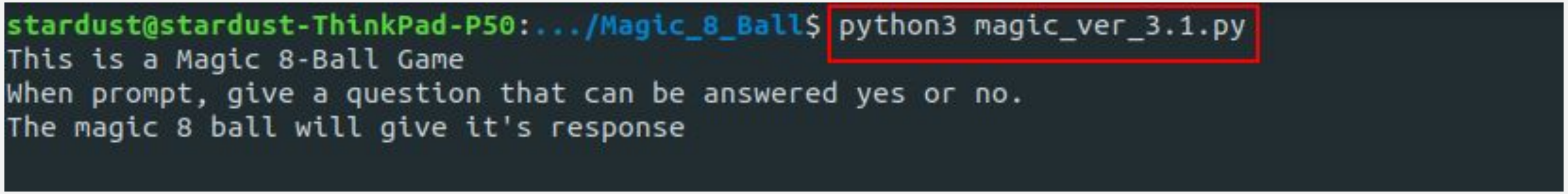
How to Run Python Scripts (in VS Code)

- Needs Python Extension


Debug
Terminal
Help
guess.py - Python Bootcamp Resources - Visual Studio Code
magic_ver_3.1.py
guess.py
Guess_The_Number > guess.py > main

```
print("\tTotal Guesses = %d" % stat[1])
```

How to Run Python Scripts Terminal



```
stardust@stardust-ThinkPad-P50:.../Magic_8_Ball$ python3 magic_ver_3.1.py
This is a Magic 8-Ball Game
When prompt, give a question that can be answered yes or no.
The magic 8 ball will give it's response
```

Comments

In Python, comments start with #

```
# This is a comment
```

No official multiline comment (multiline doc string exists)

PEP8 suggests multiple #

IDE has shortcuts you can use for easy commenting

Indentation

Leading white space is syntax

Any consistent spacing is acceptable (4 spaces or 1 tab)

```
if 7 > 3:
print("Seven is greater than three!")        # Syntax Error
```

```
if 7 > 3:
    print("Seven is greater than three!")    # Correct
```

This space matters!

Conditional Statements

if, elif (else if), else

```
a = 200
b = 33

if b > a:
    print("b is greater than a")
elif a == b:                       # if previous condition(s) not true, try this
    print("a and b are equal")
else:                              # anything not caught by other conditions
    print("a is greater than b")
```

Logical Operators

Used to combine conditional statements:

and - all statements need to be true to return True.

```
if x < 5 and x > 1:          # runs if less than 5 and more than 1
```

or - all statements need to be false to evaluate to False.

```
if x >= 2 or y <= 2:          # runs if one of these is true
```

not - switches the boolean value.

```
if not(x < 5 and x > 1):    # runs if not between 1 and 5
```

Variables

Store the memory address of an object, so it is a reference to that object

Variable names have restrictions:

- Can only contain alphanumeric characters (A-Z, 0-9, underscores)
  - `underscore9_`
- Can only start with a letter or an underscore, not numbers
  - `_underscore9`
- Case sensitive
  - `case_sensitive, CASE_SENSITIVE, Case_Sensitive`

Lists

Collection of ordered, changeable objects

Duplicate members are allowed

Lists are written with square brackets

```
thislist = [0, 1, "orange", "kiwi", "melon", 0]
print(thislist[2:5])      # prints [“orange”, “kiwi”, “melon”]


thislist[0] = "boat"                # change item value
print(thislist)

# prints ["boat", 1, "orange", "kiwi", "melon", 0]
```

Dictionaries

Sequence of *key:value* pairs, separated by commas

Keys must be unique and immutable (str, num, tuple)

Values can be any type of object

Whole dictionary enclosed in curly braces

Empty dictionary written with just two curly braces, like this: `{}`

Dictionary Operations

```
dictio = {“Name”: “Zara”, “Age”: 7, “Class”: “First”}
print(dictio[“Name”])                      # prints Zara
print(dictio[“Agr”])                       # prints KeyError: “Agr”
dictio[“Age”] = 8                          # update existing entry
dictio[“School”] = "DPS School"            # Add new entry
del dictio[“Name”]                         # remove entry with key “Name”
dictio.clear()                             # remove all entries in dict
del dictio                                 # delete entire dictionary
```

Casting

Done using using constructor functions

```
a = int(3.7)                        # 3
b = int(“234”)                      # 234
c = float(2)                        # 2.0
d = float(“43.7”)                   # 43.7
e = str(16)                         # “16”
f = str(4.5)                        # “4.5”
```

Looping

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break      # Out of loop
    if i == 1:
        i += 1
        continue   # Next iteration
    i += 1
```

```
for x in range(1,6): # 1-5
    print(x)
    if x == 1:
        continue     # Next iteration
    if x == 3:
        break        # Out of loop
```

User Input

Ask the user for input using `input()` function

Python stops executing when used

Continues when the user has given some input

```
username = input("Enter username:")    # this prints
print("Username is: " + username)      # prints with what you type
```

Functions

Not dependent on Classes and their instances, unlike methods

May or may not return any data

Created using the def keyword:

```
def my_function():
     print("Hello from a function")
```

Can only be called by its name, as it is defined independently

```
my_function()                    # prints "Hello from a function"
```

Modules/Imports

File containing a set of functions you want to include in an application

Can also contain objects of all types (arrays, dictionaries, objects, etc)

Several modules in standard Python library that you can import

Example: Import and use the platform module

```
import platform as pl              # renaming module/aliasing

x = pl.system()
print(x)                            # on Windows, prints “Windows”
```

PIP

For when you need something else for your project

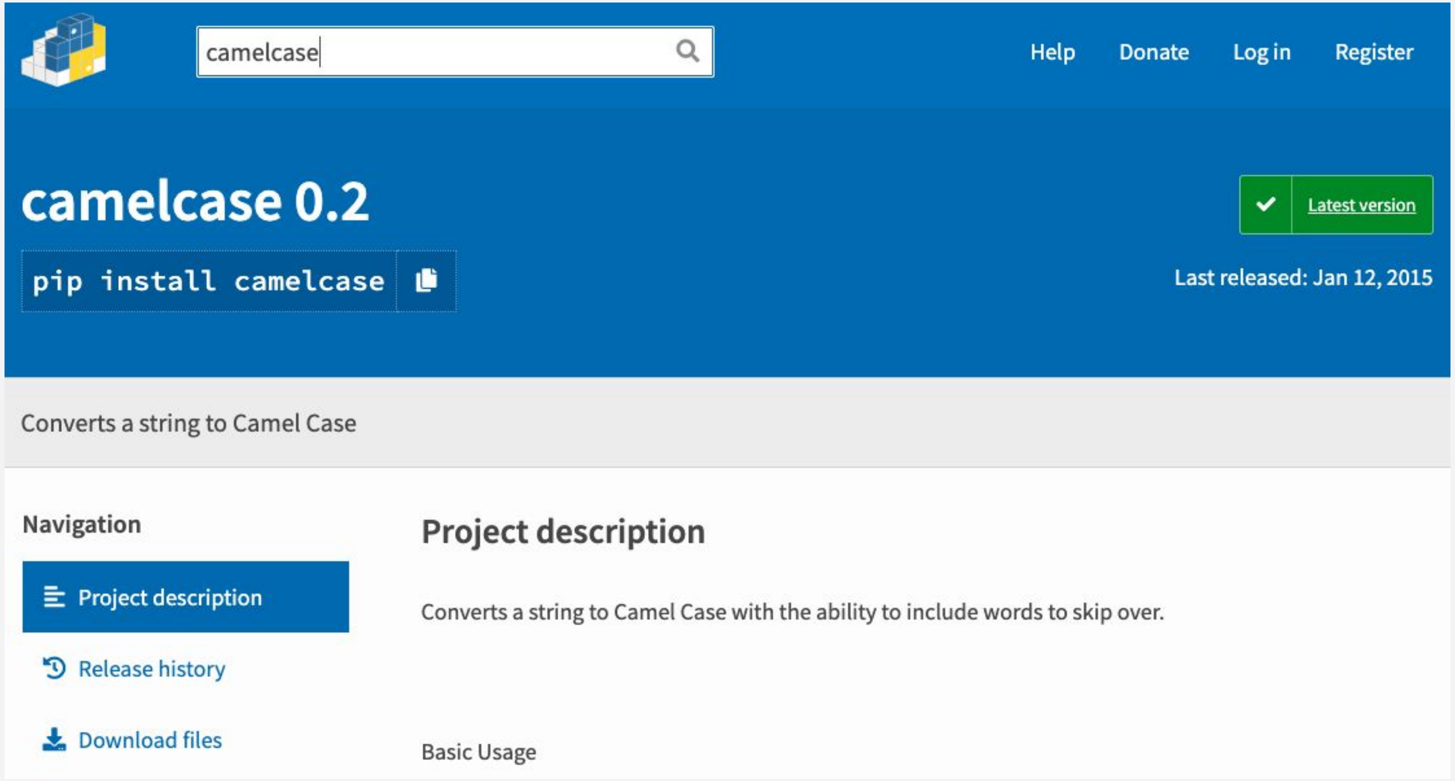
Package manager for Python, included in Python 3.4 and newer

(Distribution) packages can include:

1. Development frameworks
2. Tools
3. Modules
4. Libraries

Python Package Index is repository where pip finds and installs from

pypi.org



camelcase

Help Donate Log in Register

camelcase 0.2

```
pip install camelcase
```

Latest version

Last released: Jan 12, 2015

Converts a string to Camel Case

Navigation

- Project description
- Release history
- Download files

Project description

Converts a string to Camel Case with the ability to include words to skip over.

Basic Usage

Using a Package

Import the "camelcase" package into your project.

```
from camelcase import CamelCase

c = CamelCase()

txt = "hello world"

print(c.hump(txt))                    # prints “Hello World”

# hump method capitalizes the first letter of each word.
```

Sources

Specific Topics

https://realpython.com/python-comments-guide/

https://pythonspot.com/encapsulation/

https://dzone.com/articles/python-class-attributes-vs-instance-attributes

https://www.geeksforgeeks.org/encapsulation-in-python/

General

https://www.geeksforgeeks.org/

https://www.tutorialspoint.com/

https://www.w3schools.com/python/default.asp

Link For Project Stuff

http://bit.ly/acmw-python-dayone

Extras

Identity Operators

Check for same object, with same memory location:

is - Returns true if both variables are the same object

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x == y)     # True because content of x is equal to content of y
print(x is y)     # False because x is not the same object as y
print(x is z)     # True because z points to same object as x
```

is not - Returns true if both variables are not the same object

```
print(x is not z)     # False
```

(Default) Parameters

Specified after function name, inside the parentheses

Multiple parameters are separated by commas

If you call the function without parameter, it uses the default value:

```
def printinfo(name, age=35):       # Default parameters go last
    print "Name: ", name
    print "Age ", age
    return


printinfo(age=50, name="miki")     # print Name: miki Age 50
printinfo(name="miki")             # Name: miki Age 35
```

Passing List as a Parameter

Can send any data type as parameter for a function

Will be treated as the same data type inside the function

Send List as a parameter; it will be a List when it reaches the function:

```
def my_function(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)          # prints list in order
```

Variable-Length Arguments

When you don’t know exactly how many arguments will be given

To create a function with VLA use * before parameter name

Arguments are stored in a tuple; iterate to access values

```
def printinfo(arg1, *vartuple):
    print(arg1)
    for var in vartuple:
        print var

printinfo(10)                   # print 10
printinfo(70, 60, 50)           # print 70, 60, 50
```

Modules/Imports

File containing a set of functions you want to include in an application

Can also contain objects of all types (arrays, dictionaries, objects, etc)

Save this code to file with .py file extension, like *mymodule.py*

```
def greeting(name):
     print("Hello, " + name)

person1 = {
     "age": 36,
     "country": "Norway"
}
```

Use Module

Type the import statement and call the greeting function

```
import mymodule as mx                          # renaming module/aliasing

mx.greeting("Jonathan")                        # prints “Hello, Jonathan”

a = mx.person1["age"]
print(a)                                       # prints 36
```

When using a function from a module, use the syntax:
`module_name.function_name`

Note: Importing happens once and will override previous imports

Import From Module

To import only the person1 dictionary from the module

```
from mymodule import person1

print (person1["age"])                          # prints 36
```

Now, don't use module name when referring to elements in module

Example: person1["age"], **not** mymodule.person1["age"]

Exception Handling

When error/exception occurs, Python generates an error message

This statement will raise an error, because x is not defined:

```
print(x)
```

```
Traceback (most recent call last):
  File "demo_try_except_error.py", line 3, in <module>
    print(x)
NameError: name 'x' is not defined
```

Try and Exceptions

These exceptions can be handled using the try statement to test code

Define as many exception blocks as you want

Example: Test block of code for a special kind of error

```
try:
    print(x)
except NameError:
    print("x is not defined")                   # Gets printed
except:
    print("Something else went wrong")
else:
    print("Nothing went wrong")
```

Raise an exception

You can choose to throw an exception if a condition occurs

To throw (or raise) an exception, use the raise keyword

```
x = -1
if x < 0:
    raise Exception("Sorry, no numbers below zero")


# in terminal:
Traceback (most recent call last):
  File "demo_ref_keyword_raise.py", line 4, in <module>
    raise Exception("Sorry, no numbers below zero")
Exception: Sorry, no numbers below zero
```

Classes and Objects

Classes define an object

Classes bundle data and data structures:

1. Methods
2. Attributes

Initializing an object defined by a class called an “instance” of that class

Methods

Defined within a class, so they are dependent on that class

A method is implicitly passed to the object on which it is invoked

May or may not return any data

Can operate on data (instance variables) contained by its class

Called by its name, but is linked to an object (dependent)

Defining/Using a Class

```
class SimpleFunctionality:

    def functionality(self, num):
        print("%d" % num)                  # placeholder for number

    def __init__(self):
        # Special method, the constructor of the class.
        self.attribute = "Instance attribute"


cat = SimpleFunctionality()
mouse = SimpleFunctionality()

cat.functionality(10)      # print 10
mouse.functionality(21)    # print 21
print(cat.attribute)       # print “Instance attribute”
```

Attributes

Class attributes belong to the class

A single copy is shared by all instances

Usually defined at the top of the class

Instance attributes are not shared by all instances

Every object has its own copy of an instance attribute

Defined in class constructor

```
class ExampleClass(object):
    class_attr = 0

    def __init__(self, instance_attr):
        self.instance_attr = instance_attr


foo = ExampleClass(1)




# try to print instance attribute as a class property
print(ExampleClass.instance_attr)
# AttributeError: type object 'ExampleClass' has no attribute
'instance_attr'
```

Encapsulation

Fundamental Concept: *An object encapsulates state and behavior*

- State = The object’s attributes/variables
- Behavior = The object’s functionality/methods

Objects can restrict direct access to their internal components

A design paradigm that is part of the concept of abstraction

```
class Base:
    def __init__(self):
        self._a = 2            # Protected attribute
        print("Calling protected member: ")
        print(self._a)


obj1 = Base()
# Calling protected attribute from inside class/subclass will work
# prints both lines

print(obj1.a)
# Calling protected attribute outside class results in AttributeError
```