CONCEPTS OF PROGRAMMING

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

MODULES

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

MODULES

- Code bases can become big and unwieldy
- Need to work well with other code
  - You've written
  - Third-parties
- Require collaboration with many developers

MODULES

- Strategies to cope
  - Your own internal management of files 🤪
  - Structured organization supported by language 😌

MODULES

- Python source files use the ".py" extension and are called **modules**
- Modules provide for code reuse across applications

```
#
# greeting.py
#
def main():
    print('Hello class')
```

MODULES

- Python source files use the ".py" extension and are called **modules**
- Modules provide for code reuse across applications

```
#
# greeting.py
#
def say_hello():
    print('Hello class')
```

MODULES

- Python source files use the ".py" extension and are called **modules**
- Modules provide for code reuse across applications

SURPRISE!
YOU'VE BEEN USING MODULES THIS WHOLE TIME!

```
#
# greeting.py
#
def say_hello():
    print('Hello class')
```

MODULES

IMPORT ANOTHER MODULE USING IMPORT

CALL USING MODULE.FUNCTION SYNTAX

```
# Import the string module
import string

# Call the function using the module
# name followed by the function name
trans = string.maketrans('abegiloprstz', '463611092572')

s = 'The quick brown fox jumped over the lazy dog.'

print(s.translate(trans))

# >>> Th3 qu1ck 620wn f0x jum93d 0v32 7h3 142y d06.
```

MODULES

- Let's make a module

```
#
# greeting.py
#
def say_hello():
    print('Hello class')
```

MODULES

- Python files run in different ways using modules
- Before executing code, special variables defined
  - __name__

```
#
# greeting.py
#
def say_hello():
    print('Hello class')


print("What is the name of this module?")
print(__name__)
```

MODULES

- If Python interpreter is running the module (source file) as the main program it sets the `__name__` variable to `__main__`

```
#
# greeting.py
#
def say_hello():
    print('Hello class')


say_hello()

# __name__ is set __main__
# when we run the program as
#
# % python ./greeting.py
```

MODULES

- If file is being imported from another module `__name__` is set to the module's name
  - File name

```
#
# greeting.py
#
def say_hello():
    print('Hello class')
```

```
#
# name.py
#
import greeting

greeting.say_hello()
```

MODULES

```
#
# greeting.py
#

def say_hello():
    print("Hello class from %s" % __name__)


say_hello()

# % python greeting.py
# >>> Hello class from __main__
```

MODULES

```
#
# greeting.py
#
def say_hello():
    print("Hello class from %s" % __name__)

#
# name.py
#
import greeting
greeting.say_hello()

# % python name.py
# >>> Hello class from greeting
```

MODULES

- Defining a main() function in your code allows you to use it as a standalone program and or module

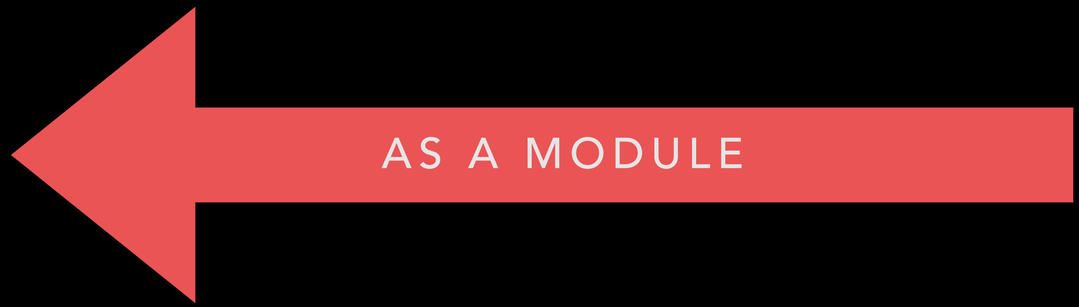
```
def say_hello():
    print("Hello class")

# Define a main function to allows the
# program to run in both modes
def main():
    say_hello()

# Boilerplate pattern
if __name__ == '__main__':
    main()
```

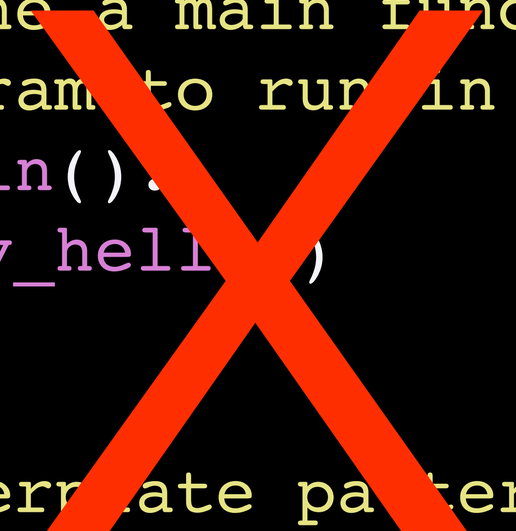
MODULES

```
def say_hello():
    print("Hello class")

# Define a main function to allows the
# program to run in both modes
def main():
    say_hello()


# Boilerplate pattern
if __name__ == '__main__':
    main()
```

MODULES

```
def say_hello():
    print("Hello class")

# Define a main function to allows the
# program to run in both modes
def main():
    say_hello()


# Boilerplate pattern
if __name__ == '__main__':
    main()
```





MODULES

- Different ways to import a module

```
# Import style 1
import greeting
```

```
# Import style 2
from greeting import say_hello
from greeting import *
```

MODULES

- Pros
  - Imports all functions in the module
- Cons
  - Typing module.function is tedious and redundant
  - 😩

```
import greeting


# Call 'say_hello' from
# the `greeting` module

greeting.say_hello()
```

MODULES

- Pros
  - Less typing
  - Control over which functions are loaded
- Cons
  - Loose context
  - Potential name clash

```
from greeting import say_hello


# Call 'say_hello' from the `greeting`
# module. No need to call using
greeting.say_hello()

say_hello()
```

MODULES

```
from greeting import say_hello, func1, func2, func3

# Call the functions imported from `greeting` module
say_hello()
func1()
func2()
func3()
```

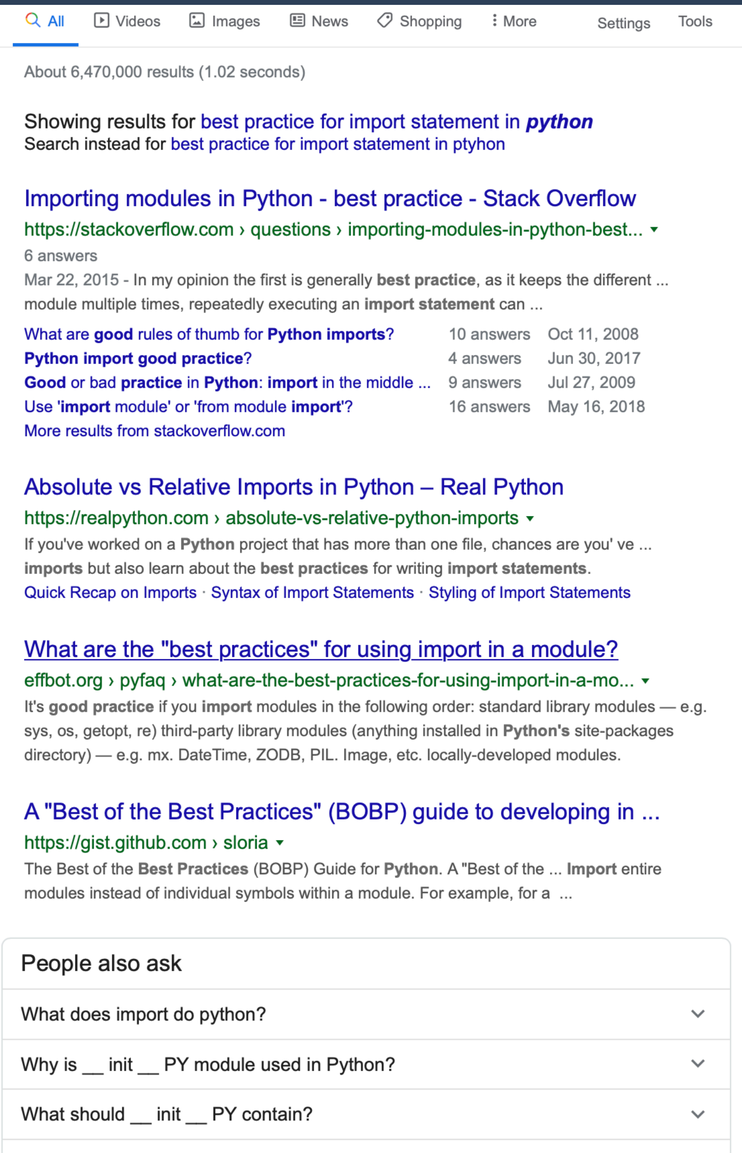
MODULES

```
from greeting import say_hello, func1, func2, func3

# Call the functions imported from `greeting` module
say_hello()
func1()
func2()
func3()


func4() # 🐞This is in the greeting.py file, but not imported
```

MODULES

```
# Import every function from the greeting module
from greeting import *

# Call functions
say_hello()
func1()
func2()
func3()
func4() # Here  it  is!
```

MODULES



BEST PRACTICE?

THOSE ARE FIGHTING WORDS

MODULES

```
# Module fight
from greeting import say_hello
from another_file import say_hello
```

```
# Which 'say_hello' is being called?
say_hello()
say_hello()
```

MODULES

```
# Module fight
from greeting import say_hello
from another_file import say_hello


# Which 'say_hello' is being called?
say_hello()
say_hello()
```

MODULES

DIR() RETURNS LIST OF THE ATTRIBUTES AND METHODS OF ANY OBJECT (SAY FUNCTIONS , MODULES, STRINGS, LISTS, DICTIONARIES ETC.)

```
% python
Python 2.7.12 |Anaconda 4.1.1 (x86_64)| (default, Jul  2
2016, 17:43:17)

>>> import greeting
>>> dir(greeting)
['__builtins__', '__doc__', '__file__', '__name__',
'__package__', 'say_hello']
```

MODULES

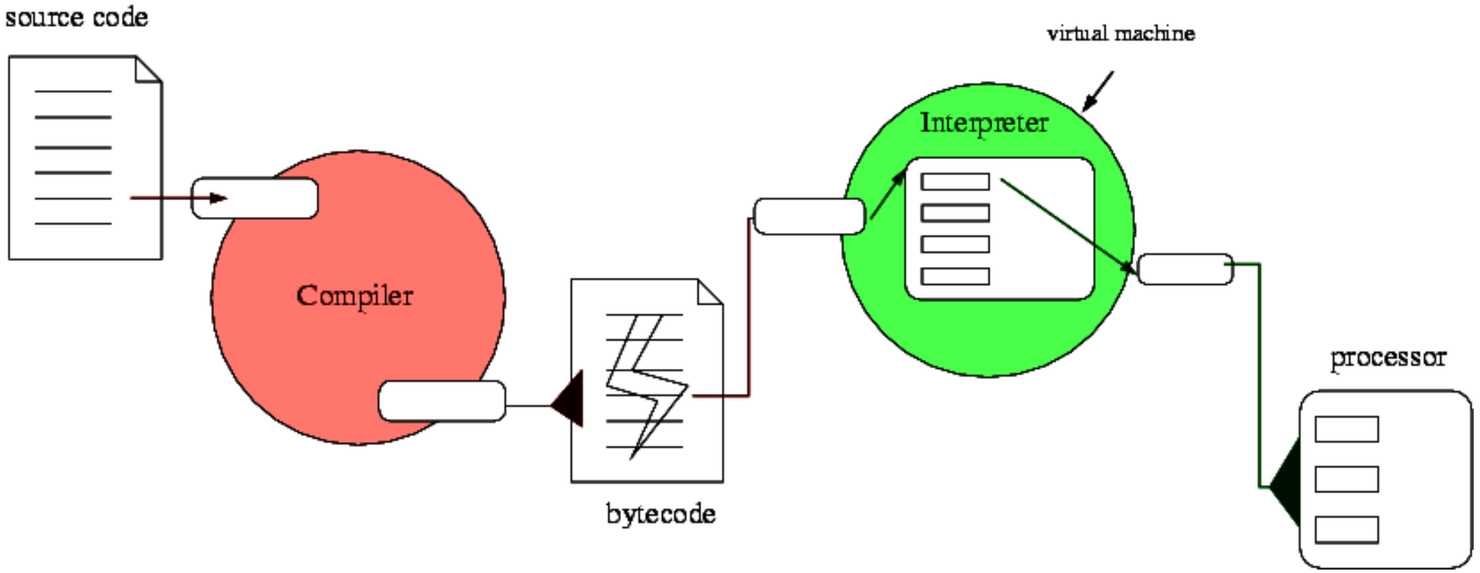
```
>>> import greeting
>>> dir(greeting)
['__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'argv',
'debugging_statement', 'say_goodbye', 'say_hello',
'say_hola']

>>> greeting.__doc__

>>> greeting.__file__
'/Users/2020-session-4/modules/greeting.py'

>>> greeting.__name__
'greeting'
```

MODULES

- .pyc file is created by the Python interpreter when it is imported
- "c"ompiled byte code that can be interpreted by the Python VM

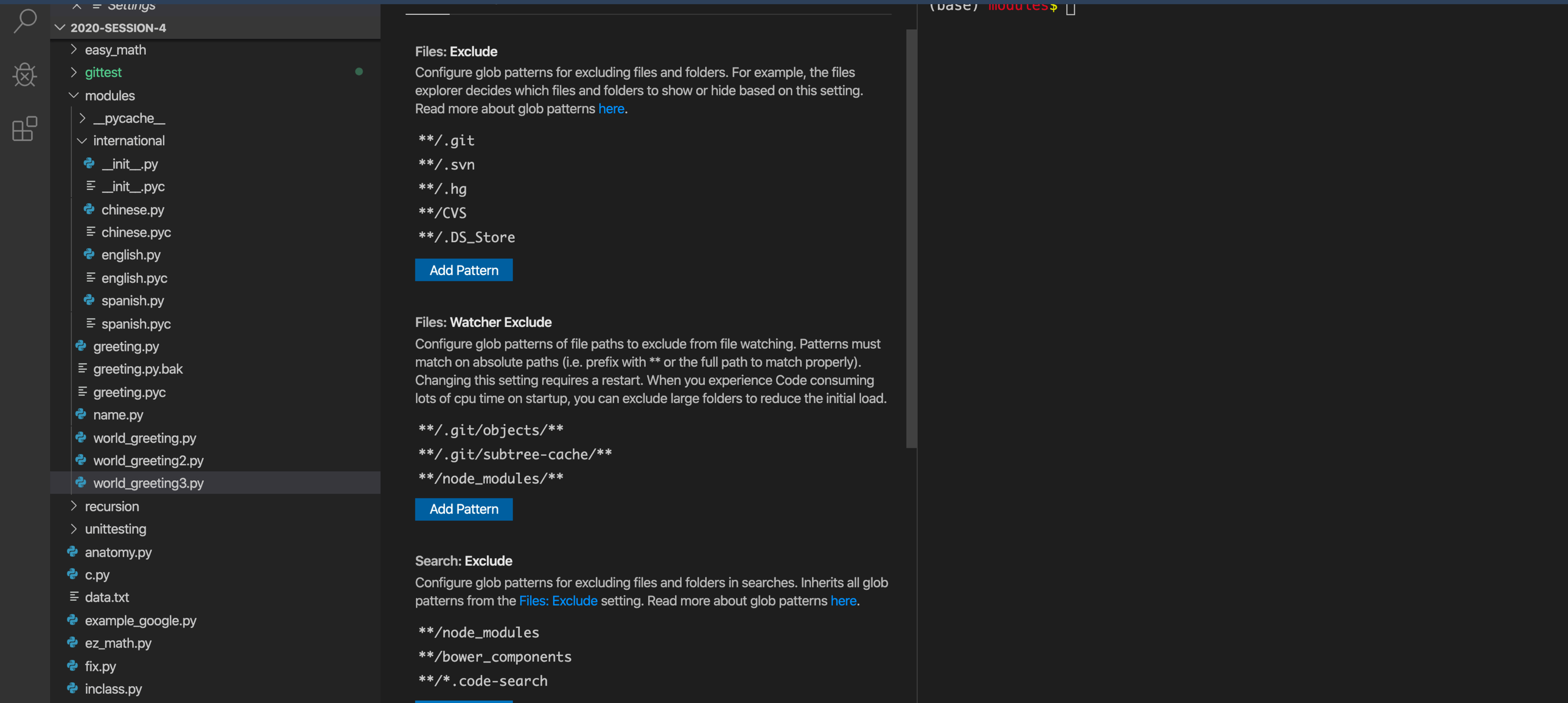
MODULES

- So, is Python is compiled language or interpreted language?

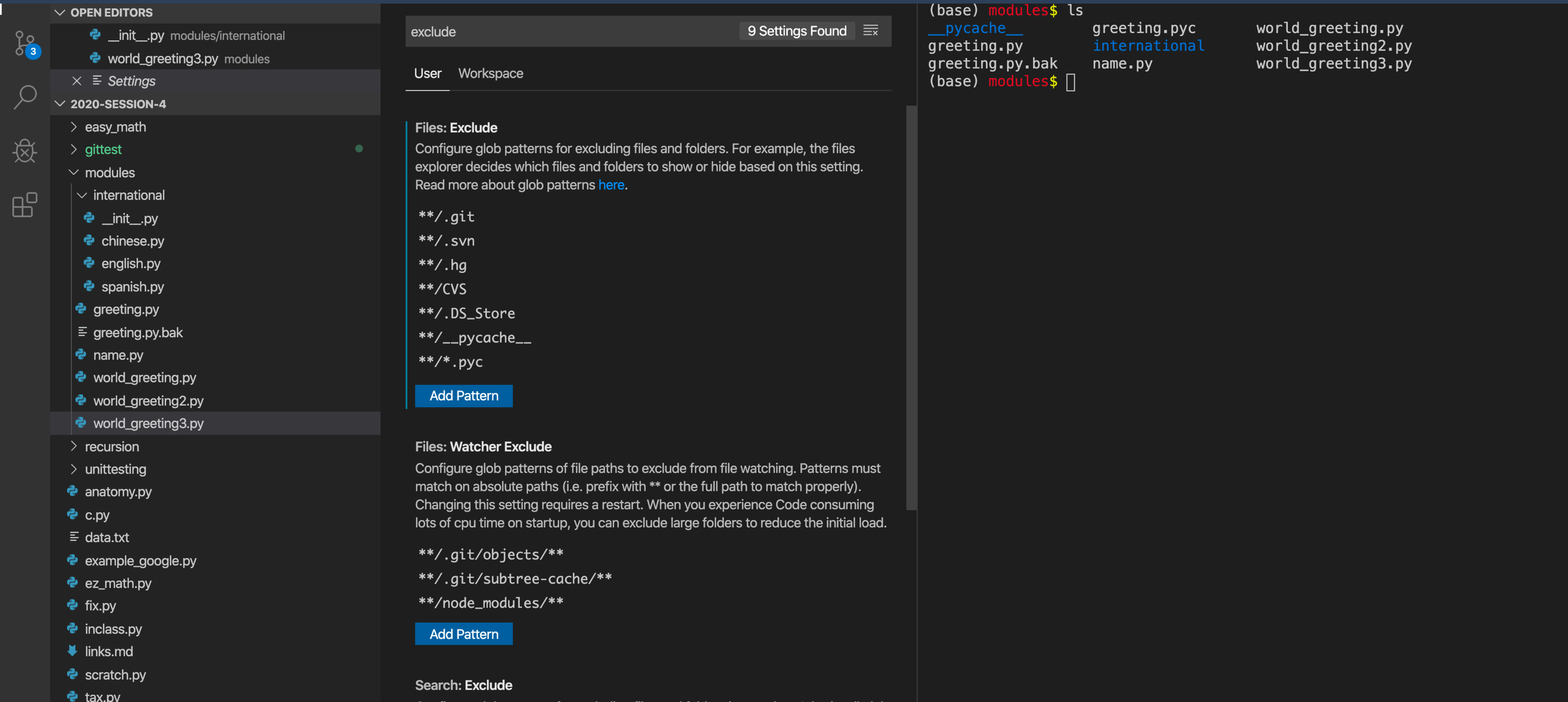


HTTP://WWW.TECHDARTING.COM/2014/04/PYTHON-COMPILED-OR-INTERPRETED-LANGUAGE.HTML

MODULES


Settings
2020-SESSION-4
easy_math
gittest
modules
__pycache__
international
__init__.py
__init__.pyc
chinese.py
chinese.pyc
english.py
english.pyc
spanish.py
spanish.pyc
greeting.py
greeting.py.bak
greeting.pyc
name.py
world_greeting.py
world_greeting2.py
world_greeting3.py
recursion
unittesting
anatomy.py
c.py
data.txt
example_google.py
ez_math.py
fix.py
inclass.py
Files: Exclude
Configure glob patterns for excluding files and folders. For example, the files explorer decides which files and folders to show or hide based on this setting. Read more about glob patterns here.
**/.git
**/.svn
**/.hg
**/CVS
**/.DS_Store
Add Pattern
Files: Watcher Exclude
Configure glob patterns of file paths to exclude from file watching. Patterns must match on absolute paths (i.e. prefix with ** or the full path to match properly). Changing this setting requires a restart. When you experience Code consuming lots of cpu time on startup, you can exclude large folders to reduce the initial load.
/.git/objects/
/.git/subtree-cache/
/node_modules/
Add Pattern
Search: Exclude
Configure glob patterns for excluding files and folders in searches. Inherits all glob patterns from the Files: Exclude setting. Read more about glob patterns here.
**/node_modules
**/bower_components
**/*.code-search
(base) modules$

MODULES


OPEN EDITORS
__init__.py modules/international
world_greeting3.py modules
Settings
2020-SESSION-4
easy_math
gittest
modules
international
__init__.py
chinese.py
english.py
spanish.py
greeting.py
greeting.py.bak
name.py
world_greeting.py
world_greeting2.py
world_greeting3.py
recursion
unittesting
anatomy.py
c.py
data.txt
example_google.py
ez_math.py
fix.py
inclass.py
links.md
scratch.py
tax.py
exclude
9 Settings Found
User
Workspace
Files: Exclude
Configure glob patterns for excluding files and folders. For example, the files explorer decides which files and folders to show or hide based on this setting. Read more about glob patterns here.
**/.git
**/.svn
**/.hg
**/CVS
**/.DS_Store
**/__pycache__
**/*.pyc
Add Pattern
Files: Watcher Exclude
Configure glob patterns of file paths to exclude from file watching. Patterns must match on absolute paths (i.e. prefix with ** or the full path to match properly). Changing this setting requires a restart. When you experience Code consuming lots of cpu time on startup, you can exclude large folders to reduce the initial load.
/.git/objects/
/.git/subtree-cache/
/node_modules/
Add Pattern
Search: Exclude
(base) modules$ ls
__pycache__ greeting.pyc world_greeting.py
greeting.py international world_greeting2.py
greeting.py.bak name.py world_greeting3.py
(base) modules$

Organizing modular code

ORGANIZING MODULES

- Reusing code may require you to organize your code in a less efficient manner
- Think about other benefits before refactoring

```
def compute_tax_and_tip(amount):
    return ((amount + \
                (amount*.0875)) * 1.20)
```

ORGANIZING MODULES

- Reusing code may require you to organize your code in a less efficient manner
- Think about other benefits before refactoring

```
def compute_tax_and_tip(amount):
    return ((amount + \
             (amount*.0875)) * 1.20)


# Refactored for reusability and
# testing
def compute_tax_and_tip(amount):
    tax = compute_tax(amount)
    tip = compute_tip(amount + tax,20)
    return amount + tax + tip

def compute_tax(amount):
    pass

def compute_tip(amount):
    pass
```

ORGANIZING MODULES

- Module search path order
  - Python interpreter looks for a built-in module with that name
  - A file with the module_name.py
  - System path (sys.path, $PYTHON_PATH

```
Python 2.7.13 |Anaconda 4.4.0 (x86_64)| (default,
05:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.
Type "help", "copyright", "credits" or "license" f
ion.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and h
org
>>> import sys
>>> print sys.path
['', '/Applications/anaconda/lib/python27.zip', '/
conda/lib/python2.7', '/Applications/anaconda/lib/
arwin', '/Applications/anaconda/lib/python2.7/plat
tions/anaconda/lib/python2.7/plat-mac/lib-scriptpa
cations/anaconda/lib/python2.7/lib-tk', '/Applicat
b/python2.7/lib-old', '/Applications/anaconda/lib/
nload', '/Applications/anaconda/lib/python2.7/site
plications/anaconda/lib/python2.7/site-packages/Sp
.egg', '/Applications/anaconda/lib/python2.7/site-
 '/Applications/anaconda/lib/python2.7/site-packag
.2.0-py2.7.egg']
```

ORGANIZING MODULES

```
[Type "help", "copyright", "credits" or "license" for more informat]
[ion.                                                              ]
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https:    ..
org
>>> import sys
>>> print sys.path
['', '/Applications/anaconda/lib/python27.zip', '/Applications/ana
conda/lib/python2.7', '/Applications/anaconda/lib/python2.7/plat-d
arwin', '/Applications/anaconda/lib/python2.7/plat-mac', '/Applica
tions/anaconda/lib/python2.7/plat-mac/lib-scriptpackages', '/Appli
cations/anaconda/lib/python2.7/lib-tk', '/Applications/anaconda/li
b/python2.7/lib-old', '/Applications/anaconda/lib/python2.7/lib-dy
nload', '/Applications/anaconda/lib/python2.7/site-packages', '/Ap
plications/anaconda/lib/python2.7/site-packages/Sphinx-1.5.6-py2.7
```

FIRST

ORGANIZING MODULES

```
502 % ls /Applications/anaconda/lib/python2.7/
BaseHTTPServer.py       config                  md                      smtpd.py
BaseHTTPServer.pyc      contextlib.py           mh                      smtpd.pyc
Bastion.py              contextlib.pyc          m                       smtplib.py
Bastion.pyc             cookielib.py            mi                      smtplib.pyc
CGIHTTPServer.py        cookielib.pyc           mi                      sndhdr.py
CGIHTTPServer.pyc       copy.py                 mi                      sndhdr.pyc
ConfigParser.py         copy.pyc                mi                      socket.py
ConfigParser.pyc        copy_reg.py             mimify.py               socket.pyc
Cookie.py               copy_reg.pyc            mimify.pyc              sqlite3
Cookie.pyc              csv.py                                          sre.py
DocXMLRPCServer.py      csv.pyc                 modulefinder.pyc        sre.pyc
DocXMLRPCServer.pyc     ctypes                  multifile.py            sre_compile.py
HTMLParser.py           curses                  multifile.pyc           sre_compile.pyc
HTMLParser.pyc          dbhash.py               multiprocessing         sre_constants.py
LICENSE.txt             dbhash.pyc              mutex.py                sre_constants.pyc
MimeWriter.py           decimal.py              mutex.pyc               sre_parse.py
MimeWriter.pyc          decimal.pyc             netrc.py                sre_parse.pyc
Queue.py                difflib.py              netrc.pyc               ssl.py
Queue.pyc               difflib.pyc             new.py                  ssl.pyc
```

THEN

ORGANIZING MODULES

```
CGIHTTPServer.py
CGIHTTPServer.pyc
ConfigParser.py
ConfigParser.pyc
Cookie.py
Cookie.pyc
DocXMLRPCServer.py
DocXMLRPCServer.pyc
HTMLParser.py
HTMLParser.pyc
LICENSE.txt
MimeWriter.py
MimeWriter.pyc
Queue.py
Queue.pyc
SimpleHTTPServer.py
SimpleHTTPServer.pyc
SimpleXMLRPCServer.py
SimpleXMLRPCServer.pyc
cookielib.pyc
copy.py
copy.pyc
copy_reg.py
copy_reg.pyc
csv.py
csv.pyc
ctypes
curses
dbhash.py
dbhash.pyc
decimal.py
decimal.pyc
difflib.py
difflib.pyc
dircache.py
dircache.pyc
dis.py
dis.pyc
mimetools.pyc
mimetypes.pyc
mimify.py
mimify.pyc
modulefinder.py
modulefinder.pyc
multifile.py
multifile.pyc
nntplib.pyc
ntpath.py
sndhdr.py
sndhdr.pyc
socket.py
socket.pyc
sqlite3
sre.py
sre.pyc
sre_compile.py
sre_compile.pyc
sre_constants.py
sre_constants.pyc
sre_parse.py
sre_parse.pyc
ssl.py
ssl.pyc
stat.py
stat.pyc
statvfs.py
statvfs.pyc
```

FINALLY

ANATOMY OF A FILE (MODULE)

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

ANATOMY OF A FILE

- Best practices for Python files and modules

```python
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

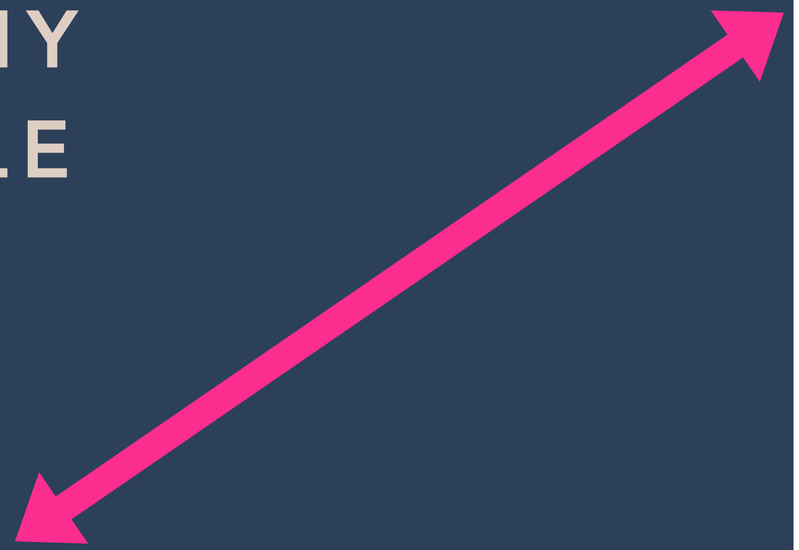
- Shebang #!
- Specify $PATH for Python interpreter
- Used in other scripting languages

#!/USR/BIN/PERL

```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.arg
    # sys.argv[0] is the script name itself and can
```

ANATOMY OF A FILE

- Shebang
- Docstring
- Author information/metada
- Imports
- Constants - in CAPITALS
- Code



```
#!/usr/bin/env python
"""Docstring goes here."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

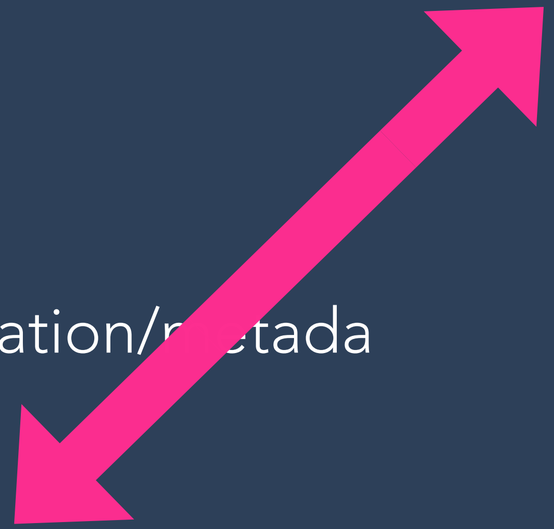
ANATOMY OF A FILE

- Shebang
- Docstring
- Author information/metada
- Imports
- Constants - in CAPITALS
- Code

```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# Import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

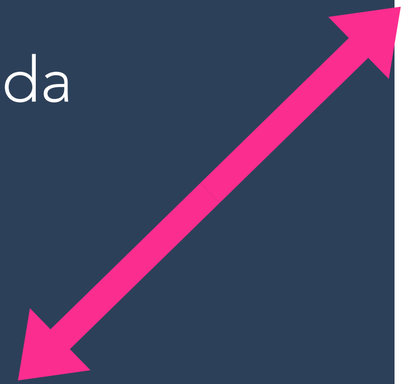
- Shebang
- Docstring
- Author information/metada
- Imports
- Constants - in CAPITALS
- Code



```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

- Shebang
- Docstring
- Author information/metada
- Imports
- Constants - in CAPITALS
- Code



```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

- main() function
- Functions
- Biolerplate for main() function

```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

- main() function
- Functions
- Biolerplate for main() function

```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in c
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

DOCSTRING AND COMMENTS

ANATOMY OF A FILE

- main() function
- Functions
- Biolerplate for main() function

```
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

- main() function
- Functions
- Biolerplate for main() function

```python
#!/usr/bin/env python
"""Anatomy.py: Description of what this file does."""

__author__      = "Jane Smith"
__copyright__   = "Copyright 2017, Company"
__license__     = "GPL"
__version__     = "1.0.1"
__email__       = "j.smith@co.com"

# import modules used here
import sys
import greeting

# Declare global constants (in caps)
PI = 3.1415
TAX_RATE = 8.75

# A main function if the module will be used as a
# standalone script
def main():
    """Print a greeting to the console using input"""
    print 'Hello there', sys.argv[1]
    # Command line args are in sys.argv[1], sys.argv[2] ...
    # sys.argv[0] is the script name itself and can be ignored

def func1():
    pass

def func2():
    pass

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
    main()
```

ANATOMY OF A FILE

- Demo

do_math.py | world_greeting2.py | add_subt

1-modules > do_math.py > ...

```
1  from ez_math import add_subtract
2
3
4  answer = add_subtract.add(3,3)
5  add_math = add_subtract.subtract(7,3)
6
7  print(answer)
8  print(add_math)
9
```

A FEW USEFUL MODULES

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

USEFUL MODULES

- Text
  - string — Text Constants and Templates
  - textwrap — Formatting Text Paragraphs
  - re — Regular Expressions
  - difflib — Compare Sequences
- Data Structures
  - enum – Enumeration Type
  - collections — Container Data Types
  - array — Sequence of Fixed-type Data
  - heapq – Heap Sort Algorithm
  - bisect — Maintain Lists in Sorted Order
  - queue — Thread-safe FIFO Implementation
  - struct — Binary Data Structures
  - weakref — Impermanent References to Objects
  - copy — Duplicate Objects
  - pprint — Pretty-print Data Structures
- Algorithms
  - functools — Tools for Manipulating Functions
  - itertools — Iterator Functions

Get the book

USEFUL MODULES

- sys provides access to system features

```
from sys import argv

# argv reads the command line and
# separates the values by space
script, first, second, third = argv

print("Argv: ", argv)
print("First: ", first)
print("Second: ", second)



# % python workspace.py 1 2
# ('Argv:', ['workspace.py', '1', '2'])
# ('First: ', '1')
# ('Second: ', '2')
```

USEFUL MODULES

- Alternative way of importing

```
import sys

script, first, second, third = sys.argv

print("Argv: ", sys.argv)
print("First: ", first)
print("Second: ", second)
```

```
# % python workspace.py 1 2
# ('Argv:', ['workspace.py', '1', '2'])
# ('First: ', '1')
# ('Second: ', '2')
```

USEFUL MODULES

- datetime for date and time manipulation

datetime — Date and Time Value Manipulation

Purpose: The datetime module includes functions and classes for doing date and time parsing, formatting, and arithmetic.

`datetime` contains functions and classes for working with dates and times, separately and together.

Times

Time values are represented with the `time` class. A `time` instance has attributes for `hour`, `minute`, `second`, and `microsecond` and can also include time zone information.

```
# datetime_time.py

import datetime

t = datetime.time(1, 2, 3)
print(t)
print('hour        :', t.hour)
print('minute      :', t.minute)
print('second      :', t.second)
print('microsecond:', t.microsecond)
print('tzinfo      :', t.tzinfo)
```

The arguments to initialize a `time` instance are optional, but the default of `0` is unlikely to be correct.

```
$ python3 datetime_time.py

01:02:03
hour        : 1
minute      : 2
second      : 3
microsecond: 0
tzinfo      : None
```


This page was last updated 2016-12-28.

Navigation
time — Clock Time
calendar — Work with Dates

Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.5.2, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

USEFUL MODULES

```
import datetime

today = datetime.date.today()
print('Today    :', today)

one_day = datetime.timedelta(days=1)
print('One day  :', one_day)

yesterday = today - one_day
print('Yesterday:', yesterday)

# Today    : 2017-01-22
# One day  : 1 day, 0:00:00
# Yesterday: 2017-01-21
```

USEFUL MODULES

- random for pseudorandom number generation
- Deterministic - same inputs will give same outputs every time

random — Pseudorandom Number Generators

Purpose: Implements several types of pseudorandom number generators.

The `random` module provides a fast pseudorandom number generator based on the *Mersenne Twister* algorithm. Originally developed to produce inputs for Monte Carlo simulations, Mersenne Twister generates numbers with nearly uniform distribution and a large period, making it suited for a wide range of applications.

Generating Random Numbers

The `random()` function returns the next random floating point value from the generated sequence. All of the return values fall within the range `0 <= n < 1.0`.

```
# random_random.py

import random

for i in range(5):
    print('%04.3f' % random.random(), end=' ')
print()
```

Running the program repeatedly produces different sequences of numbers.

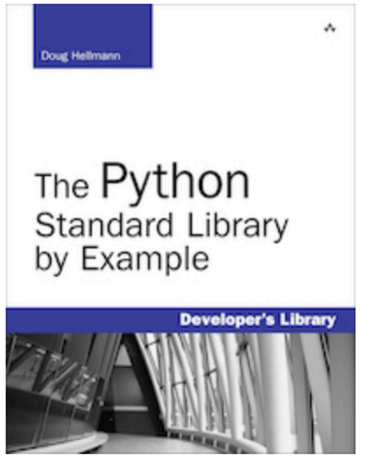
```
$ python3 random_random.py

0.859 0.297 0.554 0.985 0.452

$ python3 random_random.py

0.797 0.658 0.170 0.297 0.593
```

To generate numbers in a specific numerical range, use `uniform()` instead.

```
# random_uniform.py
```


This page was last updated 2016-12-28.

Navigation
fractions — Rational Numbers
math — Mathematical Functions



Get the book

USEFUL MODULES

- Since someone wrote an algorithm for `random` it isn't truly random
- pseudorandom

```
import random

for x in range(10):
    x = random.random()
    print x

#  0.852928792625
#  0.724121900095
#  0.571978696988
#  0.156635427665
#  0.529135673911
```

USEFUL MODULES

- Many ways to get and format a random value

```
>>> random.randint(0,100)
94
>>> random.randint(0,50)
49

>>> t = [1,2,3,4,5]
>>> random.choice(t)
3
```

ITTERTOOLS

MODULE 5
MPCS 50101

THE UNIVERSITY OF CHICAGO

ITERTOOLS

- Module to perform efficient looping

```
from itertools import count

for i in count(10):
    if i > 20:
        break
    else:
        print(i)

# 10
# 11
# 12 and so on
```

ITERTOOLS

- Perform combination and permutations of items

```
items = ['a', 'b', 'c']
from itertools import permutations

for p in permutations(items):
    print

# ('a', 'b', 'c')
# ('a', 'c', 'b')
# ('b', 'a', 'c')
# ('b', 'c', 'a')
# ('c', 'a', 'b')
# ('c', 'b', 'a')
```

ITERTOOLS

```
from itertools import combinations

print list(combinations('WXYZ', 1))
#[('W',), ('X',), ('Y',), ('Z',)]

print list(combinations('WXYZ', 2))
#[('W', 'X'), ('W', 'Y'), ('W', 'Z'), ('X', 'Y'), ('X', 'Z'),
('Y', 'Z')]

print list(combinations('WXYZ', 3))
# [('W', 'X', 'Y'), ('W', 'X', 'Z'), ('W', 'Y', 'Z'), ('X', 'Y',
'Z')]

print list(combinations('WXYZ', 4))
#[('W', 'X', 'Y', 'Z')]
```

WITH

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

WITH

- `with` statement is used in exception handling to make the code cleaner and readable

```
# With statement
with open('file_path', 'w') as file:
  file.write('hello world !')
```

WITH

```
# File handling

file = open('file_path', 'w')
file.write('hello world !')
file.close()
```

IDIOMATIC PYTHON

EXCEPTION BEFORE F.CLOSE()
COULD LEAVE YOUR PROGRAM
IN AN UNKNOWN STATE

WITH

```
# File handling
# Using a try/finally block

file = open('file_path', 'w')

try:
    file.write('hello world')
finally:
    file.close()
```

LOOKS A LITTLE BETTER

CATCHES EXCEPTION DURING FILE WRITE

WITH

```
# File handling
# Using with statement

with open('file_path', 'w') as file:
     file.write('hello world !')
```

NOW WE'RE TALKING 🤯

CAPTURES AND RELEASES THE FILE HANDLE WHEN THE CODE USING IT IS DONE

NO F.CLOSE() NEEDED

WITH

- Pattern used in methods that open/close resources
  - locks, sockets, subprocesses, telnets, etc.

```
# File handling
# Using with statement

with open('file_path', 'w') as file:
    file.write('hello world !')
```

PACKAGES

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

PACKAGES

- Packages are namespaces which contain
  - Modules
  - Other packages

```
[502 % ls /Applications/anaconda/lib/python2.7/
BaseHTTPServer.py       config                  md5.py
BaseHTTPServer.pyc      contextlib.py           mhlib
Bastion.py              contextlib.pyc          mhlib
Bastion.pyc             cookielib.py            mimet
CGIHTTPServer.py        cookielib.pyc           mimet
CGIHTTPServer.pyc       copy.py                 mimet
ConfigParser.py         copy.pyc                mimet
ConfigParser.pyc        copy_reg.py             mimif
Cookie.py               copy_reg.pyc            mimif
Cookie.pyc              csv.py                  modul
DocXMLRPCServer.py      csv.pyc                 modul
DocXMLRPCServer.pyc     ctypes                  multi
HTMLParser.py           curses                  multi
HTMLParser.pyc          dbhash.py               multi
LICENSE.txt             dbhash.pyc              mutex
MimeWriter.py           decimal.py              mutex
MimeWriter.pyc          decimal.pyc             netrc
Queue.py                difflib.py              netrc
Queue.pyc               difflib.pyc             new.p
SimpleHTTPServer.py     dircache.py             new.p
SimpleHTTPServer.pyc    dircache.pyc            nntpl
SimpleXMLRPCServer.py   dis.py                  nntpl
SimpleXMLRPCServer.pyc  dis.pyc                 ntpat
SocketServer.py         distutils               ntpat
SocketServer.pyc        doctest.py              nturl
StringIO.py             doctest.pyc             nturl
StringIO.pyc            dumbdbm.py              numbe
UserDict.py             dumbdbm.pyc             numbe
```

PACKAGES

- Namespaces is a system to clearly define scope in a program
  - Which functions and variables belong to which module
  - Prevents clash
- Python modules namespace by "Dotted module names"

```
import sys
inputs = sys.argv
```

```
import greeting
greeting.say_hello()
```

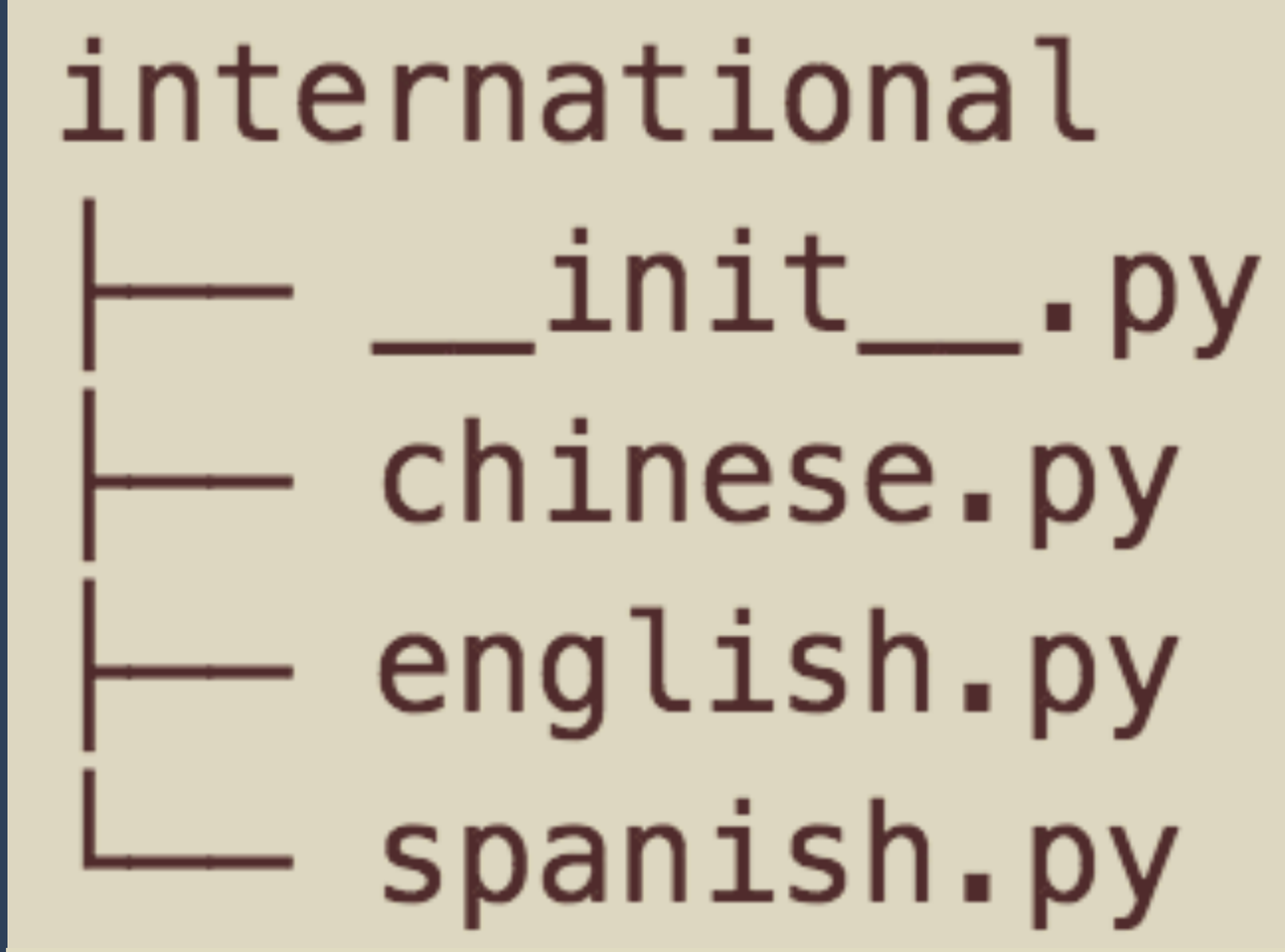
PACKAGES

- Package is a directory that contains
  - Module files (*.py)
- The package `international` has 3 modules
  - English.py
  - Chinese.py
  - Spanish.py

```
.
├── greeting.py
├── greeting.pyc
├── international
│   ├── __init__.py
│   ├── chinese.py
│   ├── english.py
│   └── spanish.py
├── name.py
└── world_greeting.py
```

PACKAGES

- Package require an `__init__.py` file
- File that indicates that the directory is a package
  - Optionally define modules to import



PACKAGES

```
├── international
│   ├── __init__.py
│   ├── chinese.py
│   ├── english.py
│   └── spanish.py
```

```
import international.chinese
import international.english
import international.spanish

international.chinese.say_hi()
international.english.say_hi()
international.spanish.say_hi()
```

PACKAGES

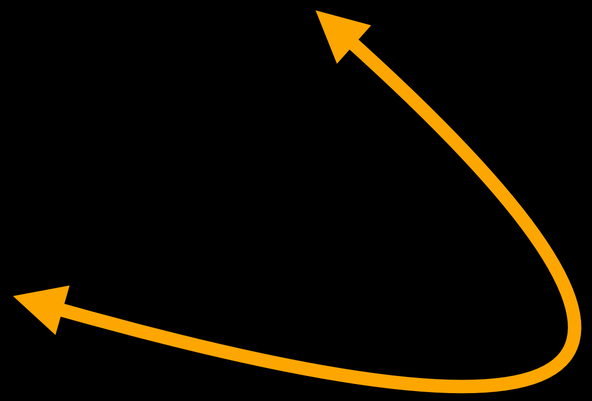
```
├── international
│   ├── __init__.py
│   ├── chinese.py
│   ├── english.py
│   └── spanish.py
```

```python
from international import chinese
from international import english
from international import spanish

chinese.say_hi()
english.say_hi()
spanish.say_hi()
```

PACKAGES

```
# __init__.py
__all__ = ["chinese","english","spanish"]
```

```
├── international
│   ├── __init__.py
│   ├── chinese.py
│   ├── english.py
│   └── spanish.py
```



```
# greeting.py
from international import *

chinese.say_hi()
english.say_hi()
spanish.say_hi()
```

RECURSION

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO

RECURSION


WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

RECURSION

- Recursion is the process when a function calls itself
- Recursive function

```
def countdown(n):
    # Test the current value of `n`
    if n < 0:
        print('Blastoff!')
    else:
        print("n = %s" % n)
        # Call countdown again
        countdown(n-1)


countdown(5)


# >>> 5 4 3 2 1 0 Blastoff!
```

RECURSION

```
def countdown(n):
    # Test the current value of `n`
    if n < 0:
        print('Blastoff!')
    else:
        print("n = %s" % n)
        # Call countdown again
        countdown(n-1)


countdown(5)

# >>> 5 4 3 2 1 0 Blastoff!
```

RECURSION

BASE CONDITION

- Define a "base condition" that escapes the recursion

```
def countdown(n):
    # Test the current value of `n`
    if n < 0:
        print('Blastoff!')
    else:
        print("n = %s" % n)
        # Call countdown again
        countdown(n-1)


countdown(5)


# >>> 5 4 3 2 1 0 Blastoff!
```

RECURSION

```
# Factorial
#     0! = 1
#     5! = 5 * 4 * 3 * 2 * 1
#     n! = n(n-1)!

def fact(n):
    if n == 0:
        return 1
    return n * fact(n -1)

print(fact(0))  # 1
print(fact(5)) # 120
print(fact(10)) # 3628800
```

RECURSION

- If the base condition is not met the memory will become exhausted
- "Stack overflow" error

```
# Stack overflow

def fact(n):
    if n == 1000:
        return 1
    return n * fact(n-1)
```

RECURSION

- If the bas
  the mem
  exhauste
- "Stack ov

```
(base) 1-modules$ /Users/tabinkowski/anaconda3/bin/python "/Users/tabin
kowski/Google Drive/g-Teaching/uchicago.codes/uchicago.codes-courses/mp
cs50101/mpcs50101-2020-winter/mpcs50101-2020-winter-sessions/2020-sessi
on-4/2-recursion/scratch.py"
Traceback (most recent call last):
  File "/Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchic
ago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-2020-winter
-sessions/2020-session-4/2-recursion/scratch.py", line 8, in <module>
    print(fact(10))  # 1
  File "/Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchic
ago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-2020-winter
-sessions/2020-session-4/2-recursion/scratch.py", line 6, in fact
    return n * fact(n-1)
  File "/Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchic
ago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-2020-winter
-sessions/2020-session-4/2-recursion/scratch.py", line 6, in fact
    return n * fact(n-1)
  File "/Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchic
ago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-2020-winter
-sessions/2020-session-4/2-recursion/scratch.py", line 6, in fact
    return n * fact(n-1)
  [Previous line repeated 995 more times]
  File "/Users/tabinkowski/Google Drive/g-Teaching/uchicago.codes/uchic
ago.codes-courses/mpcs50101/mpcs50101-2020-winter/mpcs50101-2020-winter
-sessions/2020-session-4/2-recursion/scratch.py", line 4, in fact
    if n == 1000:
RecursionError: maximum recursion depth exceeded in comparison
```

```
:
act(n-1)
```

RECURSION

- Advantages
  - Don't repeat yourself
  - Simple way to write code
  - Some problems lend themselves to recursive solutions (traversing a tree)

```
def countdown(n):
    # Test the current value of `n`
    if n < 0:
        print('Blastoff!')
    else:
        print("n = %s" % n)
        # Call countdown again
        countdown(n-1)


countdown(5)


# >>> 5 4 3 2 1 0 Blastoff!
```

RECURSION

- Disadvantages
  - Takes more memory (everything is held until base condition is met)
  - Takes longer (calling and returning from functions)

```
def countdown(n):
    # Test the current value of `n`
    if n < 0:
        print('Blastoff!')
    else:
        print("n = %s" % n)
        # Call countdown again
        countdown(n-1)


countdown(5)


# >>> 5 4 3 2 1 0 Blastoff!
```

RECURSION

- All recursive programs can be written iteratively and vice versa

```
def countdown(n):
    # Test the current value of `n`
    if n < 0:
        print('Blastoff!')
    else:
        print("n = %s" % n)
        # Call countdown again
        countdown(n-1)


countdown(5)


# >>> 5 4 3 2 1 0 Blastoff!
```

RECURSION

```
def draw_triangle(dot, star):
    if dot == 0:
        return 0

    else:
        left_space = dot + 1
        print('.' * left_space, end=' ')
        asterixs = star * 2 + 1
        print('*' * asterixs)

        draw_triangle( dot - 1, star + 1 )
```

```
...*
..***
.*****
```

RECURSION

```
def draw_triangle(i, t=0):
    if i == 0:
        return 0
    else:
        left_space = i + 1
        asterixs = t * 2 + 1
        print ' ' * left_space + '*' * asterixs
        return draw_triangle( i - 1, t + 1 )

draw_triangle(10)
```

THE END

MODULE 4
MPCS 50101

THE UNIVERSITY OF CHICAGO