

PYTHON LAB BOOK

Python For Programmers

UCSC-Extension CMPR.X416.(37)

June 8- 11, 2020

Online available until June 21

Marilyn Davis, Ph.D.
Your Instructor

Lab 1 – Output

- Executing a Python program
- Writing to `stdout`
- Assigning: labels and objects
- `strings`
- `if, elif, and else`
- `while` and another `else`
- Iterating with a `for` loop
- Counting loop with `range`
- Relational and logical operators

Lab 2 – input

- Input from `stdin`
- Formatted strings
- Factory functions
- Catching an exception:
 - yet another `else`
- `bytes, str, and unicode`

Lab 3 – Functions

- Function protocols
- Default function arguments
- Keyword function arguments

Lab 4 – Star Power

- Unpacking sequences into a call
- Variable number of arguments
- Unpacking sequences into identifiers
- Swapping elements

Lab 5 – Libraries

- Reference/Assignment
- `global` declaration
- importing Libraries
- Modules: `random, math`
- Introspection

Lab 6 – Sequences

- Sequence types: `str, tuple, list`
- Sequence slicing and other manipulations
- `list` facilities

Lab 7 – Sorting

- Sorting by a key

Lab 8 – Important Trick

- Important trick:
 - `__name__` and "`__main__`"

Lab 9 – Functional Programming

- List Comprehensions
- Functional Programming

Lab 10 – Dictionaries

- Dictionaries
- Dictionary Comprehensions

Lab 11 – File IO

- File I/O
- Context handler:with keyword
- Reading unicode
- sys library
- Modules: shutil, tempfile

Lab 12 – Portable Python

- Module: os
- Walking A Directory

Lab 13 – set and from

- Passing mutables/immutables into functions
- Importing with the from keyword
- sets
- nonlocal declaration

Lab 14 – Packages

- copy library
- Python Packages

Lab 15 – Dynamic Code

- Dynamic Code Generation
- Modules:
 - subprocess(Optional)
 - cProfile(Optional)

Lab 16 – Fancy Facilities

- Function protocols: variable number of keyword arguments
- Formatted strs using a dictionary
- Unpacking dictionaries
- locals() dictionary
- Generators
- iter and next
- Decorators (Optional)

Lab 17 – OOP

- Classes
- Container class
- Inheritance
- Class attribute

Lab 18 – Magic

- Multiple Inheritance
- Useful attributes
- Making Magic with *Special Methods*
- Privacy

Lab 19 – Extending Builtins

- Extending Builtin Classes
- Iterators
- Diamond Inheritance
- Encapsulation
- Static Methods (Optional)
- Class Methods (Optional)

Lab 20 – Developer Modules

- Raw File I/O
- `raise` an exception
- `try/except/else/finally`
- Context Manager Class
- Module: `unittest` (Optional)
- Module: `optparse` (Optional)

Lab 21 – Wrap Up

- Exceptions:
 - `traceback` module
 - `raise` exceptions
 - Inventing exceptions
- Namespaces
- Nests
- Pitfalls
- Finding Modules and Help

©Marilyn Davis, 2007-2020

Acknowledgements

These materials are the product of hundreds of programmers in the Silicon Valley. I'm grateful for their feedback: finding errors, making suggestions, and honest reactions.

Thanks are also due to my family, Sebastian, Isabella, Josephine, Charlie, and Clint Davis, for being proud of me, and relentlessly encouraging me to do what I really want to do, in spite of external pressures.

Instructor: Marilyn Davis, Ph.D.

Email Address: marilyn@pythontrainer.com

Phone Number: (650) 814-4435

Book Recommendation

The Quick Python Book Second Edition by Vernon L. Ceder; ISBN # 9781935182207, published by Manning.

Regrettably, it's about Python 3, but the two Python languages are similar enough that you can learn about both Pythons from this excellent book.

Online Resources

1. <http://www.python.org> and in particular: <http://www.python.org/doc> has very helpful documentation, online and free.
2. *The Python Cookbook* by Alex Martelli, Anna Martelli Ravenscroft & David Ascher. This is a very interesting collection of Python code, best read after you have taken the class. ISBN # 0-596-00797-3 and online free.

Python Class Style Guide

One big advantage of Python is its readability. When you write your code, always think of it as literature or art, something to be admired by a reader.

If you don't understand a style listed below, we have probably not yet learned that part of Python.

1. You may break the style guide in certain circumstances – but you must have a good reason for everything you do that does not follow the style guide. Document that reason!

2. Do what is expected. Don't give your reader a reason to think about anything but the intention of the code. No distractions; nothing unnecessary.

3. Be tasteful about documentation:

- Your code should not include documentation about language features, or about anything obvious.
- Do provide complete documentation about your modules, functions, and classes in doc strings.
- Use the # when you want to document details for the reader of the code.
- There should be no external documentation, i.e., no README files.

4. Indentations are 4 spaces. Don't create deep nests of indents. Instead, improve your architecture. Maybe make more functions. Check that you don't have any unnecessary indentations:

```
    continue <-- or break, or return, or sys.exit()
else: <-- This else is not necessary.
      This indentation is not necessary.
```

5. Blocks of code begin on the next line: *after* the line with a colon. When you have a colon in your code, it is the last thing on that line.

6. No duplicate code (after you have learned to make functions).

7. All functionality should be in functions (after you have learned to make functions). Even data should almost always be in functions, not global.

8. Labels are meaningful, long if necessary, and self-documenting. The case of your labels means a lot to the reader:

- `most_object_labels` (except callable functions) are lower case with an underline to separate words.
- `CONSTANT_LABELS` are all upper case.
- `FunctionLabels` have every word capitalized, including the first. They start with verbs or are verbs.
- `ClassName` labels have every word capitalized, including the first. They are nouns.
- `module_labels.py` are lower case with an underline to separate words.

9. A comma has a space after it, and no spaces before it.

10. Usually the assignment operator has one space on each side:

```
bacon_strips = 3
```

But:

- In function definitions, there are no spaces around the `=` in defaulted arguments:

```
def RateIt(name, rating=100):
    pass
```

- In function calls, when providing keyword arguments, there are no spaces around the `=`:

```
RateIt("Alice's", rating=99)
```

11. Parentheses have no spaces around them. Also:

- No extra parentheses, unless they help readability.
- Conditionals don't have parentheses:

```
if chocolate_candy == "dark":
    pass
```

Grading

Your grade will be the average of the scores you earn on 8 *Reviews* that will become available online.

There is one Review that you should attempt after each 2 labs we complete in class. Even if we get time to do the optional labs about Regular Expressions, there will not be a Review to cover this material.

You are welcome to research the answers to the Reviews any way you wish. In particular, ask questions in class. You can only submit each Review one time, so try to get the answers correct before you submit your Review. You can *save* your answers to work on them later, and then *submit* when you are sure of your answers.

Note that the Reviews are learning activities, not *Tests* or *Quizzes*.

The Reviews will be available until 1 week after the last classroom class. And, when the class meetings are finished, there will be an online *Discussions* forum to ask questions about the Reviews, or anything.

However, it is more fun for us if you ask your questions in class. Read through the Review that covers the labs we just completed. Ask about any doubts before you turn in your Review. I am very happy when a question from the Review leads to a discussion in class. And, I want you to get the right answer before you turn in your Review.

Class Meetings

The face-to-face class meetings will be a series of short lectures and lab exercises. We'll look at the solutions to the lab exercises after you have had some time to work on them.

Because the lectures are mostly code review, you will get the most from your studies if you bring the printed pdf, and a pencil, to class. The printed pdf fits nicely into a 1.5", 3-ring binder. When the class is over, you will find it to be a great resource for your continued work in Python.

Note that two pdf's are provided:

- ***Python_For_Programmers_for_printing.pdf*** which has the solutions in the back of the document. The pages are arranged so that you can, as we work through the labs, bring the pages with solutions forward when you are ready to see each solution. This way, you will be less tempted to look at the solutions before you have given the exercise a good try.
- ***Python_For_Programmers_for_viewing.pdf*** which, hopefully, you will not need because you have printed the "for_printing.pdf".

Other Online Resources

Online, you'll also find `labs.zip` and `code.zip`:

- ***labs.zip*** has partially solved exercises, to keep you focused on the new material. This becomes important later in the class.
- ***code.zip*** has all the class code in it.

©Marilyn Davis, 2007-2020

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 1 Output

- Executing a Python program
- Writing to stdout
- Assigning: labels and objects
- strings
- if, elif, and else
- while and another else
- Iterating with a for loop
- Counting loop with range
- Relational and logical operators

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0

primes.py

```

1 #!/usr/bin/env python3
2 """Produces a list of prime numbers > 2.
3
4 Here, we are only checking the "look" of Python code.
5 """
6 MAX = 100                                     # '#' starts a comment.
7
8 print("primes less than", MAX, "are:") # A new line is added
9                                # by default.
10
11 for number in range(3, MAX, 2):
12     div = 2
13     while div * div <= number:
14         if number % div == 0:
15             break
16         div += 1
17     else:           # Overloaded 'else': control goes
18                 # here when the loop finishes and
19                 # doesn't 'break'.
20     print(number, end=' ')
21     # Use the 'end' keyword argument to control what is
22     # printed at the end.
23
24 print() # This call produces only the new line.

```

Notes:

Call on the command line if you are running *NIX.

```
$ primes.py
primes less than 100 are:
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Or, invoke the interpreter:

```
$ python3 primes.py
primes less than 100 are:
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

```
# Or, ask the interpreter to run it and then stay active for
# introspection.
```

```
$ python3 -i primes.py
primes less than 100 are:
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
>>> number
99
>>>
```

The screenshot shows the Python Idle Development Environment. On the left is a code editor window titled "primes.py - /home/marilyn/python/mm/labs/lab_01_Output/primes.py*". The code is a script to find prime numbers up to MAX = 100. It includes comments explaining the code's flow and a section on notes. On the right is a terminal window titled "Python Shell" showing the output of the script, which lists prime numbers from 3 to 73. The terminal also displays system information and a timestamp.

```
#!/usr/bin/python
"""Produces prime numbers.
Here, we are showing the "look" of Python code.

MAX = 100          # Here is a comment.

print 'primes less than', MAX, 'are:' # A new line is added by default.

for number in range(3, MAX, 2):
    div = 2
    while div * div <= number:
        if number % div == 0:
            break
        div += 1
    else:
        print number, # Trailing comma suppresses the new line.
    number += 2
print          # This only produces the new line.

""" Notes:
Call on the command line if you are running *NIX.

0, 15:52:39
primes() for more inform
*****arn about the connecti
is computer's internal
ot visible on any exte
or received from the
*****
=====
47 53 59 61 67 71 73 7
```

Figure 1: Idle Development Environment

Integrated development environments for Python abound. We will be using *idle*, the original Python environment, because it is free and a no-brainer. But, some others are much better:

- <http://wiki.python.org/moin/IntegratedDevelopmentEnvironments>

delimiting_strs.py

```
1 #!/usr/bin/env python3
2 """Demonstrates 4 ways to delimit strings."""
3
4 print('Hello world')
5 print()
6 print('She said "Hello world"')
7 print()
8 print("She said 'Hello world'")
9 print()
10 print('''Little dark woman of my suffering,
11 with eyes of flying paper,
12 you say "Yes" to everyone,
13 but you never say when.
14 ''') # end of string started on line 10.
15
```

\$ delimiting_strs.py

Hello world

She said "Hello world"

She said 'Hello world'

Little dark woman of my suffering,
with eyes of flying paper,
you say "Yes" to everyone,
but you never say when.

\$

©Marilyn Davis, 2007-2020

else.py

```
1 #!/usr/bin/env python3
2 """Demonstrates if/elif/else and while/else."""
3
4 number = 25
5
6 print(number, "is", end=' ')
7 if number < 10:
8     print("small")
9 elif number < 1000:
10    print("medium")
11 else:
12    print("large")
13
14 if 10 < number < 50:    # "and" is assumed
15    print("number is in")
16 # Alternate syntax since 2.5 -- all one line but less readable.
17 print(number, "is", end=' ')
18 print("small" if number < 10 \
19       else "medium" if number < 1000 \
20       else "large")
21 # else can also occur in a loop
22 div = 2
23 while div * div <= number:
24     if number % div == 0:
25         print(number, "is divisible by", div)
26         break
27     div += 1
28 else:
29     print(number, "is prime")
```

```
$ else.py
25 is medium
number is in
25 is medium
25 is divisible by 5
```

range.py

```
1 #!/usr/bin/env python3
2 """ Studying range.
3 """
4 MAX = 10
5 print('range is used with a "for/in" combination:')
6 for number in range(3, MAX, 2):
7     print(number, end=' ')
8 print()
9 print('range outside a "for/in" loop gives:', range(3, MAX, 2))
```

```
$ range.py
range is used with a "for/in" combination:
3 5 7 9
range outside a "for/in" loop gives: range(3, 10, 2)
$
```

range – Built-in Type

If you want to get the results of range without a loop:

```
>>> tuple(range(3, 10, 2))
(3, 5, 7, 9)
>>> list(range(3, 10, 2))
list(range(3, 10, 2))
[3, 5, 7, 9]
>>>
```

```
range(almost_end)
range(start, almost_end[, increment=1])
```

All the same:

```
range(10)
range(0, 10)
range(0, 10, 1)
```

counting_loop.py

```
1 #!/usr/bin/env python3
2 """Always use a for/in loop when you know how times you want
3 the loop to go around. It is, by far, the most efficient.
4 """
5
6 for num in range(5):
7     print(num, "* 2 =", num * 2)
8
```

OUTPUT:

```
$ counting_loop.py
0 * 2 = 0
1 * 2 = 2
2 * 2 = 4
3 * 2 = 6
4 * 2 = 8
```

Relational Operators in Python:

< means less than

> means greater than

<= means less than or equal

>= means greater than or equal

== means equal

!= means not equal

Logical Operators:

and means and

or means or

not means not

Lab 01 – Exercises:



1. Bring up **Idle** on your computer.

The interpreter works as a calculator. Type this at the prompt:

```
>>> 1 + 2 + 4
```

Then try:

```
>>> _ + 8
```

Type:

- On Windows and Linux:
 - **Alt-p** repeatedly to cycle through previous lines.
 - **Alt-n** repeatedly to cycle back.
- On OS X:
 - **Ctrl-p** repeatedly to cycle through previous lines.
 - **Ctrl-n** repeatedly to cycle back.
- If that didn't work, find Idle's **Help** button.

Now, make a program *module* (script) with the `1 + 2 + 4` line in it, i.e., `1 + 2 + 4`. Run it. Does it work? No? Fix it.

Note that the interpreter displays its evaluation of the line but that a program will not display unless you ask it to **print**.

2. Although Python tries to remove the back-slash from your code. Sometimes you need it. Try this in the interpreter:

```
>>> greeting = "Hello \nworld."  
>>> greeting
```

Now try:

```
>>> print(greeting)
```

Notice that **printing** interprets the backslash-n to create a new line, while evaluating just spits out the raw string.

And try:

```
>>> food = "popcorn"  
>>> print(food * 3)
```

and

```
>>> food + food
```

+ means **concatenation** to strings.

* means **repetition** to strings.

3. Write a script to produce this output — EXACTLY:

```
He said "Hello World".  
She said 'Hello Sky'.  
She said "He said 'Hello World'".
```

(Hint: This is very easy if you remember to use triple-quotes, and print appears only once in your solution. In fact, try to always avoid using a backslash in your code to keep it *Pythonic*, i.e., readable.)

4. How would you produce the following using the range operator?

```
[3, 6, 9, 12]  
[-10, 100, 210]  
[-1, -3, -5, -7]
```

5. Write a script to produce this output using range and for:

```
10 9 8 7 6 5 4 3 2 1 BLASTOFF!!!
```

6. Try this in the interpreter:

```
>>> for ch in "Howdy":  
...     print(ch)  
...     <-- Here, hit the return key to  
...         finish the indented block  
  
>>> for num in (2, 4, 16):  
...     print(num)  
...
```

Strings and comma-separated objects, as well as many other Python objects, can be iterated with the for and in.

If the comma-separated objects are not wrapped with [] or {}, but may be wrapped with (), they are called *tuples*.

If the comma-separated objects are wrapped with [] or {}, they are other collection objects with terrific facilities, and we'll study them soon. And try this:

```
for thing in (2, "hat", (0, 1)):  
    print(thing)
```

A tuple can contain any sort of object, even nested tuples.

7. Use a for loop and a tuple of strings to produce:

```
Hi ya Manny!  
Hi ya Moe!  
Hi ya Jack!
```

Do it without duplicating any code or data to maximize robustness.

8. (Optional) Write a script that produces this pattern:

A 10x10 grid of black asterisks (*) arranged in a diamond shape. The pattern is centered at the top of the grid. It consists of four layers of asterisks, with the innermost layer being a 2x2 square of asterisks at the center.

Can you find an easier way? Hint: Remember exercise 2 where you learned to multiply and concatenate strings.

9. (Optional) Print the decimal equivalent of a binary string. Test with "1011".

Binary string: 1011
Decimal equivalent: 11

You will find the `int` builtin function very useful. At the interpreter's prompt, type:

```
>>> help(int)
```

Only read a few lines until you discover the Pythonic way to do this exercise.

And/or try it using a for-loop and a while-loop.

lab01_3.py

```
1 #!/usr/bin/env python3
2 """Produces strings with embedded strings."""
3
4 print("""He said "Hello World".
5 She said 'Hello Sky'.
6 She said "He said 'Hello World'".
7 """)    # End of string to print
8
```

Output:

```
$ lab01_3.py
He said "Hello World".
She said 'Hello Sky'.
She said "He said 'Hello World'".
```

\$

lab01_4.py

```
1 #!/usr/bin/env python3
2 """Produces the following sequences using
3 the range operator.
4
5     [3, 6, 9, 12]
6     [-10, 100, 210]
7     [-1, -3, -5, -7]
8 """
9 print(list(range(3, 13, 3)))
10 print(list(range(-10, 211, 110)))
11 print(list(range(-1, -8, -2)))
```

```
$ lab01_4.py
[3, 6, 9, 12]
[-10, 100, 210]
[-1, -3, -5, -7]
$
```

lab01_5.py

```
1 #!/usr/bin/env python3
2 """Produce a countdown and BLASTOFF!!!
3 """
4 for count in range(10, 0, -1):
5     print(count, end=' ')
6 print("BLASTOFF!!!")
7
```

```
$ lab01_5.py
10 9 8 7 6 5 4 3 2 1 BLASTOFF!!!
$
```

lab01_7.py

```
1 #!/usr/bin/env python3
2 """ Greets the Pep Boys.
3 """
4
5 for name in "Manny", "Moe", "Jack":
6     print("Hi ya", name + '!')
7
```

```
$ lab01_7.py
Hi ya Manny!
Hi ya Moe!
Hi ya Jack!
$
```

©Marilyn Davis, 2001/2020

lab01_8.py

```
1 #!/usr/bin/env python3
2 """Produces a christmas tree pattern of stars."""
3
4 blanks = 18
5 stars = 1
6 for line in range(10):
7     print(blanks * ' ', stars * '* ')
8     blanks -= 2
9     stars += 2
10
```

```
$ lab01_8.py
```

```
*
 * *
 * * *
 * * * *
 * * * * *
 * * * * * *
 * * * * * * *
 * * * * * * * *
 * * * * * * * * *
 * * * * * * * * * *
```

©Marilyn Davis, 2007-2020

lab01_9.py

```
1 #!/usr/bin/env python3
2 """Prints the decimal equivalent of a binary string."""
3 string = "1011"
4 print(string, end=' ')
5 for ch in string:
6     if ch not in "01":
7         print("is not a binary string.")
8         break
9 else:
10     answer = 0
11     for ch in string:
12         answer = 2*answer + int(ch)
13     print("in binary is", str(answer) + '.')
14     # Python method
15     print("Easy way:", int(string, 2))
```

```
$ lab01_9.py
1011 in binary is 11.
Easy way: 11
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 2 input

- Input from `stdin`
- Formatted strings
- Factory functions
- Catching an exception:
 - yet another `else`
- `bytes`, `str`, and `unicode`

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0

get_input.py

```
1 #!/usr/bin/env python3
2 """Demonstrates input."""
3
4 answer = input("What is your favorite number? ")
5 print("You like", answer, '?')
6 print("You really like " + answer + '?')
7 print(answer, "is", end=' ')
8 number = int(answer)
9 if number < 10:
10     print("small.")
11 elif number < 1000:
12     print("medium.")
13 else:
14     print("large.")
15
16 print("But you like " + str(number) + '!')
```

```
$ get_input.py
What is your favorite number? 8
You like 8 ?                      # space before the '?!'
You really like 8?
8 is small.
But you like 8!
$  
Notes: int(x)    -> converts to integer
      str(x)   -> converts to string
      float(x) -> converts to float

$ get_input.py
What is your favorite number? x
You like x ?
You really like x?
x is Traceback (most recent call last):
  File "./get_input.py", line 8, in <module>
    number = int(answer)
ValueError: invalid literal for int() with base 10: 'x'  
$
```

©Marilyn Davis, 2007-2020

try_input.py

```
1 #!/usr/bin/env python3
2 """Demonstrates catching an exception on error."""
3
4 answer = input("What is your favorite number? ")
5 print("You like", answer, '?')
6 print("You really like " + answer + '?')
7 print(answer, "is", end=' ')
8 try:
9     number = int(answer)
10 except ValueError:
11     print("not a number!")
12 else:
13     if number < 10:
14         print("small.")
15     elif number < 1000:
16         print("medium.")
17     else:
18         print("large.")
19
20     print("But you like " + str(number) + '!')
```

```
$ try_input.py
What is your favorite number? 44
You like 44 ?
You really like 44?
44 is medium.
But you like 44!
$ try_input.py
What is your favorite number? xx
You like xx ?
You really like xx?
xx is not a number!
$
```

input_pattern.py

```
1 #!/usr/bin/env python3
2 """Demonstrates input in a while loop."""
3
4 while True: # True/False are Python's boolians.
5     answer = input("What is your favorite number? ")
6     try:
7         number = int(answer)
8     except ValueError:
9         print(answer, "is not a number! Please try again.")
10    else:
11        break
12 print("I got your %s!" % (number))      # % replacement
13 print("I got your {}!".format(number))  # str.format method
14 print(f"I got your {number}!")          # f-string
```

```
$ input_pattern.py
What is your favorite number? nine
nine is not a number! Please try again.
What is your favorite number? 9
I got your 9!
I got your 9!
I got your 9!
$
```

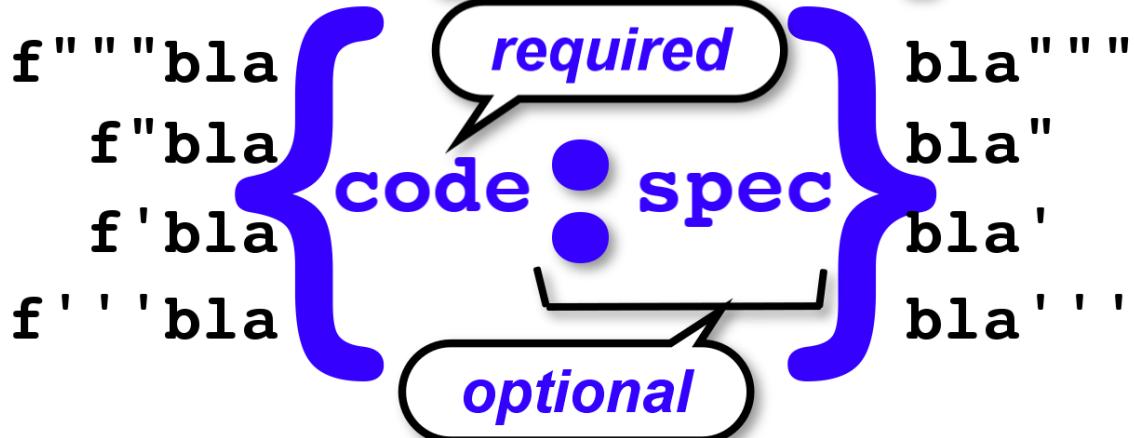
©Marilyn Davis, 2007-2020

paint.py

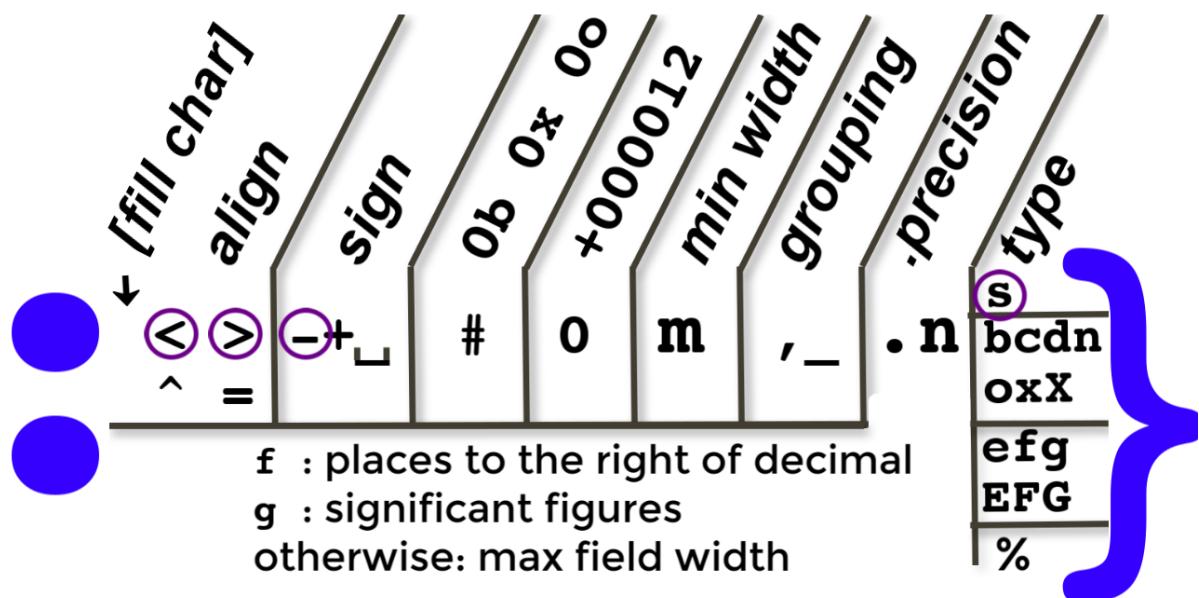
```
1 #!/usr/bin/env python3
2 """
3 Another input pattern.
4 """
5 PER_GALLON = 200          # A can of paint covers 200 square feet
6 sq_ft = 0
7 while sq_ft == 0:
8     said = input("Number of square feet to paint: ")
9     if not said:    # or if said == '':
10         print("Thank you anyway.")
11         break
12     try:
13         sq_ft = int(said)
14     except ValueError:
15         print("Please give a whole number.")
16 else:
17     no_cans = sq_ft//PER_GALLON      # // is integer division
18     if sq_ft % PER_GALLON > 0:      # Leftover after division
19         no_cans += 1
20         print(f"You need {sq_ft/PER_GALLON:.1f} cans"
21               " so you'd better buy", end=' ')
22     else:
23         print("You need exactly", end=' ')
24     print(f"{no_cans} {'can' if no_cans==1 else 'cans'}.")
```

```
$ paint.py
Number of square feet to paint: 250
You need 1.2 cans so you'd better buy 2 cans.
$ paint.py
Number of square feet to paint: 400
You need exactly 2 cans.
$ paint.py
Number of square feet to paint:
Thank you anyway.
```

f-string formatting



spec



<https://docs.python.org/3/library/string.html#grammar-token-align>

```
$ examples.py
float: f"{the_float}"
float: f"{the_float:.2}"
money: f"${the_float:.2f}"
float: f"{the_float:+10.2f}"
float: f"{the_float:<10.2f}"

int: f"{the_int}"
hex: f"{the_int:#X}"

str: f"{the_string}"
str: f"{the_string:<20}"
str: f"{the_string:^20}"
str: f"{the_string:*>20}"
str: f"{the_string:2}"
str: f"{the_string:.2}"
unicode: f"{'\u00a3'c}""

str: f"{short_str * 2:^{3*len(short_str)}}"
$
```

To get '{' or } into your output, you double the braces:

```
>>> f"{{O}"
>>> f"{{O"
>>> x = 3
>>> f"{{{0}{0}{0}}}"
>>>
```

Use the 'r' prefix to get the literal back-slash:

```
>>> message = "f-strings"
>>> print(rf"\index={message}")
\index=f-strings
>>>
```

©Marilyn Davis, 2007-2020

examples.py

```
1 #!/usr/bin/env python3
2 """
3 Some example f-strings.
4 """
5 the_float = 12.345
6 print(f'float: f"{{the_float}}" \t\t\t\t= |{the_float}|')
7 print(f'float: f"{{the_float:.2}}" \t\t\t\t= |{the_float:.2}|')
8 print(f'money: f"${{{the_float:.2f}}}" \t\t\t\t= ${the_float:.2f}')
9 print(f'float: f"{{the_float:+10.2f}}" '
10      f' \t\t\t\t= |{the_float:+10.2f}|')
11 print(f'float: f"{{the_float:<10.2f}}" '
12      f' \t\t\t\t= |{the_float:<10.2f}|')
13 print()
14 the_int = 12
15 print(f'int: f"{{the_int}}" \t\t\t\t= |{the_int}|')
16 print(f'hex: f"{{the_int:#X}}" \t\t\t\t= |{the_int:#X}|')
17 print()
18 the_string = "Croque Monsieur"
19 print(f'str: f"{{the_string}}" \t\t\t\t= |{the_string}|')
20 print(f'str: f"{{the_string:<20}}" \t\t\t\t= |{the_string:<20}|')
21 print(f'str: f"{{the_string:^20}}" \t\t\t\t= |{the_string:^20}|')
22 print(f'str: f"{{the_string:*>20}}" \t\t\t\t= |{the_string:*>20}|')
23 print(f'str: f"{{the_string:2}}" \t\t\t\t= |{the_string:2}|')
24 print(f'str: f"{{the_string:.2}}" \t\t\t\t= |{the_string:.2}|')
25 print(f'unicode: f"{{165:c}}" \t\t\t\t= |{165:c}|')
26 print()
27 short_str = "XO"
28 print(f'str: f"{{short_str * 2:^\{3*len(short_str)\}}}" '
29      f' \t\t\t\t= |{short_str * 2:^\{3*len(short_str)\}}|')
```

uni.py

```
1 #!/usr/bin/env python3
2 """Two string types in Python 3:
3     str and bytes
4 """
5 yen_symbol = chr(165) # unicode int
6 print("yen_symbol = chr(165)")
7 print("yen_symbol =", yen_symbol)
8 print("ord(yen_symbol) =", ord(yen_symbol))
9 print("type(yen_symbol) =", type(yen_symbol))
10 print()
11
12 yen_bytes = yen_symbol.encode("utf-8")
13 print('yen_bytes = yen_symbol.encode("utf-8")')
14 print("type(yen_bytes) =", type(yen_bytes))
15 print("yen_bytes =", yen_bytes)
16 print("""bytes(yen_symbol, encoding="utf-8") = """,
17       bytes(yen_symbol, encoding="utf-8"))
18 print()
19
20 greeting = "Hello"
21 print('greeting = "Hello"')
22 print("type(greeting) =", type(greeting))
23 print("greeting =", greeting)
24 print("""bytes(greeting, encoding="utf-8") = """,
25       bytes(greeting, encoding="utf-8"))
26 print()
27
28 greeting_bytes = b"Hello"
29 print('greeting_bytes = b"Hello"')
30 print("type(greeting_bytes) =", type(greeting_bytes))
31 print("greeting_bytes =", greeting_bytes)
32 print('str(greeting_bytes, encoding="utf-8") = ',
33       str(greeting_bytes, encoding="utf-8"))
34 print()
```

```
$ uni.py
yen_symbol = chr(165)
yen_symbol = ¥
ord(yen_symbol) = 165
type(yen_symbol) = <class 'str'>

yen_bytes = yen_symbol.encode("utf-8")
type(yen_bytes) = <class 'bytes'>
yen_bytes = b'\xc2\xa5'
bytes(yen_symbol, encoding="utf-8") = b'\xc2\xa5'

greeting = "Hello"
type(greeting) = <class 'str'>
greeting = Hello
bytes(greeting, encoding="utf-8") = b'Hello'

greeting_bytes = b"Hello"
type(greeting_bytes) = <class 'bytes'>
greeting_bytes = b'Hello'
str(greeting_bytes, encoding="utf-8") = Hello
$
# print("b'¥' =", b'¥')
# SyntaxError:
# bytes can only contain ASCII literal characters.
```

©Marilyn Davis, 2007-2020

Formatting Strings

"string with % language" % (argument for every % and *)

%	OPTIONAL				?
	flag	m	.	n	
		minimum width all conversions		precision	conversion character
		reads the width * from the arguments		reads the * precision from the arguments	
	-	left justify		for integers - forces n digits	d, i o, x, X
	+	sign on positive values		for strings - maximum characters	s, r
	_	blank instead of plus		for floats - digits to right of decimal	f, e, E
	0	pad with zeros		general floats - significant figures	g, G
	#	prepends: 0 for %#o 0x or 0X for %#x or %#X		%% makes one %	%

FUTURE TOPIC	
Pythonic Pre-Flag	
(key)	dictionary replacement of value

We'll learn this after we learn about
dictionaries.

Notes:

- %s and %r always work.
- r"\n" is a raw string and makes \n.
Backslashes are not escaped if you prepend a r.

```
Printing 12.3456
    "%d" % (12.3456) -> |12|
    "%o" % (12.3456) -> |14|
    "%X" % (12.3456) -> |C|
    "%#o" % (12.3456) -> |014|
    "%-#10o" % (12.3456) -> |014           |
    "%#x" % (12.3456) -> |0xc|
    "%#X" % (12.3456) -> |0XC|
    "%+d" % (12.3456) -> |+12|
    "%6d" % (12.3456) -> |     12|
    "%-+6d" % (12.3456) -> |+12   |
    "%06d" % (12.3456) -> |000012|
    "%-6d" % (12.3456) -> |12   |
    "%1d" % (12.3456) -> |12|
    "%f" % (12.3456) -> |12.345600|
    "%g" % (12.3456) -> |12.3456|
    "%.1f" % (12.3456) -> |12.3|
    "%.2f" % (12.3456) -> |12.35|
    "%10.1f" % (12.3456) -> |      12.3|
    "%-10.1f" % (12.3456) -> |12.3      |
    "%.*f" % (5, 2, 12.3456) -> |12.35|
Printing "Croque Monsieur"
    "%s" % ("Croque Monsieur") -> |Croque Monsieur|
    "%20s" % ("Croque Monsieur") -> |      Croque Monsieur|
    "%-20s" % ("Croque Monsieur") -> |Croque Monsieur      |
    "%5s" % ("Croque Monsieur") -> |Croque Monsieur|
    "%.5s" % ("Croque Monsieur") -> |Croqu|
-----
```

Division Issue Python 2

```
>>> 3/10
0
>>> from __future__ import division
>>> 3/10
0.2999999999999999
>>> 3//10
0      <--- //  2 slashes will always mean integer division.
```

Lab 02 – Exercises:



Be sure that your programs behave nicely, even when your user doesn't cooperate.

1. Write a program that asks the user for her name, then asks for the amount of money she has. Then ask her to give you half.

There is nothing to check about the name given.

A sample run:

```
$ lab02_1.py
Name please: Nancy
How much money do you have? some
Please try again.
How much money do you have? 22.17
Nancy, give me $11.09.
$
```

2. Write a program that requests a floating point number from the user, asks for the number of digits to the right of the decimal for presenting the result, and then presents the result of multiplying the original number by itself (squaring the number) with the number of digits to the right of the decimal that was asked for. Here is a sample run, note the error handling:

```
$ lab02_2.py
Number to square please: pi
Please try again.
Number to square please: 3.14
How many digits to the right of the decimal place would
you like to have displayed? many
Please try again.
How many digits to the right of the decimal place would
you like to have displayed? 2
3.14 squared is 9.86.
$
```

3. (Optional) Write a program that asks the user to think of a number between 1 and 10. The program then tries to guess the number and asks the user if the computer's guess is right. If the guess is not right, ask the user if the guess is high or low and then take another guess. When the guess is correct, report how many guesses it took to get the right number. My output looks like:

```
$ lab02_3.py
Think of a number between 1 and 10 and I'll try to guess it.
Is your number 5?
Please press:
'y' for yes
'n' for no
n
No? Then please press:
'h' if 5 is higher than your number
'l' if 5 is lower than your number
h
Is your number 2?
Please press:
'y' for yes
'n' for no
y
Hurray! Only 2 guesses.
```

Realize that you need no libraries for this. The user is making up the number and your program is discovering it. It's a common confusion to think that the computer needs to make up a number. That'll come later.

If you are running in Idle, and there is some mysterious problem, it could be that you have multiple interpreters running, maybe even in the background by now. When your program waits for the user's input, be sure to satisfy its requests, or press Ctrl-d or Ctrl-c Ctrl-c to stop it.

©Marilyn Davis, 2007-2020

lab02_1.py

```

1 #!/usr/bin/env python3
2 """
3 Asks the user for her name, then asks for the amount of
4 money she has. It then asks for half the money.
5 """
6 name = input("Name please: ")
7 while True:
8     try:
9         money = float(input("How much money do you have? "))
10        break
11    except ValueError:
12        print("Please try again.")
13
14 print(f"{name}, give me ${money/2:.2f}.")

```

Sample output is in the specification.

lab02_2.py

```

1 #!/usr/bin/env python3
2 """
3 lab02_2.py Asks the user for a number to multiply by itself,
4 and asks the user for the number of digits beyond the decimal
5 point to display, and gives the answer.
6 """
7 while True:
8     number_str = input("Number to square please: ")
9     try:
10         number = float(number_str)
11     except ValueError:
12         print("Please try again.")
13     else:
14         break
15
16 while True:
17     try:
18         right_of_decimal = int(input(
19             "How many digits to the right of the decimal place"
20             " would you like to have displayed? " ))
21     except ValueError:
22         print("Please try again.")
23     else:
24         break
25
26 print(f"{number_str} squared is "
27       f"{number*number:.{right_of_decimal}f}.")

```

Sample output is in the specification.

lab02_3.py

```
1 #!/usr/bin/env python3
2 """I'm thinking-of-a-number game."""
3
4 print("Think of a number between 1 and 10",
5       "and I'll try to guess it.")
6 high = 10
7 low = 1
8 guesses = 0
9 while high >= low:
10     guesses += 1
11     guess = (high + low)//2
12     print(f"Is your number {guess}?")
13     while True:
14         answer = input("""Please press:
15             'y' for yes
16             'n' for no
17             """)
18         answer = answer[0].lower() # details coming
19         if answer in "yn":
20             break
21         print("Please follow directions.")
22     if answer == 'y':
23         print(f"Hurray! Only {guesses} guesses!")
24         break
25
26     while True:
27         answer = input(f"""No? Then please press:
28             'h' if {guess} is higher than your number
29             'l' if {guess} is lower than your number
30             """)
31         if answer[0].lower() in "lh":
32             break
33         print("Please follow directions")
34
35     if answer == 'l':
36         low = guess + 1
37     else:
38         high = guess - 1
39
```

```
$ lab02_3.py
```

Think of a number between 1 and 10 and I'll try to guess it.

Is your number 5?

Please press:

'y' for yes

'n' for no

n

No? Then please press:

'h' if 5 is higher than your number

'l' if 5 is lower than your number

h

Is your number 2?

Please press:

'y' for yes

'n' for no

y

Hurray! Only 2 guesses.

\$

©Marilyn Davis, 2007-2020

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

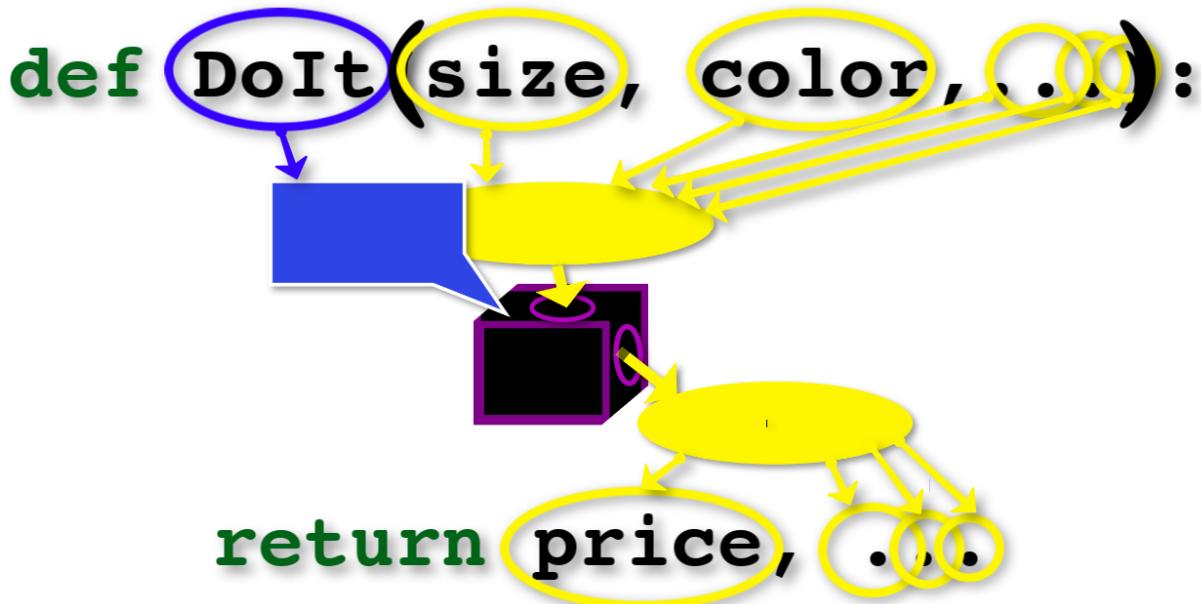
UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 3 Functions

- Function protocols
- Default function arguments
- Keyword function arguments

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.



double_it.py

```
1 #!/usr/bin/env python3
2 """Function with one argument and one return value."""
3
4 def DoubleIt(x):
5
6     """Function doc comes first, if the name (which
7     is a verb according to the style guide) isn't
8     sufficient documentation.
9     """
10    return 2 * x
11
12 print(DoubleIt(2))
13 print(DoubleIt("Hi"))
14 print(DoubleIt(2.2))
```

```
$ double_it.py
4
HiHi
4.4
$
```

order_counts.py

```

1 #!/usr/bin/env python3
2 """Order matters.
3 """
4 def StrumGuitar(measures):
5     for measure in range(measures):
6         print("strum. strum.")
7     BeatDrum(measures)
8
9 StrumGuitar(2)
10
11 def BeatDrum(measures):
12     for measure in range(measures):
13         print("boom! boom!")

```

```

$ order_counts.py
strum. strum.
strum. strum.
Traceback (most recent call last):
  File "./order_counts.py", line 9, in <module>
    StrumGuitar(2)
  File "./order_counts.py", line 7, in StrumGuitar
    BeatDrum(measures)
NameError: name 'BeatDrum' is not defined

```

right_order.py

```

1 #!/usr/bin/env python3
2 """Function definitions come before any calls in the 0th
3 column.
4 """
5 def StrumGuitar(measures):
6     for measure in range(measures):
7         print("strum. strum.")
8     BeatDrum(measures)
9
10 def BeatDrum(measures):
11     for measure in range(measures):
12         print("boom! boom!")
13
14 StrumGuitar(2)

```

```

$ right_order.py
strum. strum.
strum. strum.
boom! boom!
boom! boom!

```

default.py

```
1 #!/usr/bin/env python3
2 """Demonstrates defaulted arguments."""
3
4 def GoToThePark(name, best_friend="Jose'"):
5     print(f"""{name} and {best_friend}
6 go to the park
7 play hide-and-seek
8 till long after dark
9 """)
10
11 GoToThePark("Charlie", "Josephine")
12 GoToThePark("Judi")
```

```
$ default.py
Charlie and Josephine
go to the park
play hide-and-seek
till long after dark

Judi and Jose'
go to the park
play hide-and-seek
till long after dark
```

```
$
```

greet.py

```
1 #!/usr/bin/env python3
2 """Demonstrates keyword arguments"""
3
4 def Greet(first, last):
5     print(f"Hello {first} {last}.")
6
7 Greet("Rocky", "The Squirrel")
8 Greet(last="Moose", first="Bullwinkle")
9
```

```
$ greet.py
Hello Rocky The Squirrel.
Hello Bullwinkle Moose.
$
```

Lab 03 – Exercises:



From now on, put all your functionality into functions.

1. Pay close attention to the specification and our style guide as you code these little functions. Use string formatting where appropriate. Test your functions and show the output.

- (a) Write a function that receives a number and returns the number formatted to one decimal place as in the sentence "Good number 32.2.", where the number that was received is placed in the sentence instead of the 32.2.

I named my function JudgeNumber because this label adheres to our style guide. This line:

```
print(JudgeNumber(32))
```

results in:

```
'Good number 32.0.'
```

written to stdout. While:

```
print(JudgeNumber(1.4))
```

shows this on stdout:

```
'Good number 1.4.'
```

The function returns the string because the spec says it should. It does not print. If the spec said it should print the string, only then would it print the string.

Optional, and not part of the specification: what happens when you:

```
print(JudgeNumber("1"))
```

- (b) Write a function that receives a noun and a verb on the argument and prints "All of our <the noun> wanted to <the verb>." The noun received is printed in place of <the noun>, and the verb received is printed instead of <the verb>.

My output:

```
$ lab03_1.py
Testing JudgeNumber:
Good number 2.0.
Good number 8.3.
JudgeNumber did not like x.
Testing PrintSillySentence:
All of our cherry wanted to run.
All of our 3 wanted to 2.
```

2. Write a function that returns a total cost from the sales price and sales tax rate. Both price and tax rate should be in the argument list. Provide a default value for the tax rate = 8.25%. Test your function. Be sure to use the default value, and a different value for the tax rate, demonstrating that you understand the point.

3. Write a DoBreakfast function that takes five arguments: meat, eggs, potatos, toast, and beverage. The default meat is bacon, eggs are over easy, potatos is hash browns, toast is white, and beverage is coffee. Using the defaults, the function just says: Here is your bacon and over easy eggs with hash browns and white toast. Can I bring you more coffee? When the arguments are changed, the sentence changes.

Call it at least 3 different times with different orders, showing me that you understand all the flexibility possible.

©Marilyn Davis, 2007-2020

lab03_1.py

```
1 #!/usr/bin/env python3
2 """
3 Pay close attention the specification and our style guide
4 as you code these little functions:
5 """
6 def JudgeNumber(number):
7     """Returns the number formatted to one decimal place
8     in the sentence "Good number <number>."
9     """
10    return f"Good number {number:.1f}."
11
12 def PrintSillySentence(noun, verb):
13     """prints "All of our <the noun> wanted to <the verb>""
14     """
15    print(f"All of our {noun} wanted to {verb}.")
16
17 def main():
18    print("Testing JudgeNumber:")
19    for number in (2, 8.3, 'x'):
20        try:
21            print(JudgeNumber(number))
22        except ValueError:
23            print(f"JudgeNumber did not like {number}.")
24
25    print("Testing PrintSillySentence:")
26    for noun, verb in (("cherry", "run"), (3, 2)):
27        PrintSillySentence(noun, verb)
28
29 main()
```

```
$ lab03_1.py
Testing JudgeNumber:
Good number 2.0.
Good number 8.3.
JudgeNumber did not like x.
Testing PrintSillySentence:
All of our cherry wanted to run.
All of our 3 wanted to 2.
$
```

lab03_2.py

```
1 #!/usr/bin/env python3
2 """ Write a function that returns a total cost from the
3 sales price and sales tax. The default value for the
4 tax rate should be 8.25%.
5 """
6
7
8 def CalculateCost(price, tax=.0825):
9     return price * (1 + tax)
10
11 def main():
12     print(" price | .0825 | .0925")
13     for price in range(50, 1001, 50):
14         dollars = price/100
15         print(f"${dollars:5.2f} | ${CalculateCost(dollars):6.2f}"
16               f" | ${CalculateCost(tax=.0925, price=dollars):6.2f}")
17
18 main()
```

```
$ lab03_2.py
price | .0825 | .0925
$ 0.50 | $ 0.54 | $ 0.55
$ 1.00 | $ 1.08 | $ 1.09
$ 1.50 | $ 1.62 | $ 1.64
$ 2.00 | $ 2.17 | $ 2.19
$ 2.50 | $ 2.71 | $ 2.73
$ 3.00 | $ 3.25 | $ 3.28
$ 3.50 | $ 3.79 | $ 3.82
$ 4.00 | $ 4.33 | $ 4.37
$ 4.50 | $ 4.87 | $ 4.92
$ 5.00 | $ 5.41 | $ 5.46
$ 5.50 | $ 5.95 | $ 6.01
$ 6.00 | $ 6.50 | $ 6.55
$ 6.50 | $ 7.04 | $ 7.10
$ 7.00 | $ 7.58 | $ 7.65
$ 7.50 | $ 8.12 | $ 8.19
$ 8.00 | $ 8.66 | $ 8.74
$ 8.50 | $ 9.20 | $ 9.29
$ 9.00 | $ 9.74 | $ 9.83
$ 9.50 | $ 10.28 | $ 10.38
$10.00 | $ 10.82 | $ 10.93
$
```

©Marilyn Davis, 2007-2020

lab03_3.py

```
1 #!/usr/bin/env python3
2 """Write a DoBreakfast function that takes five arguments:
3 meat, eggs, potatos, toast, and beverage. The default
4 meat is bacon, eggs are over easy, potatos is hash browns,
5 toast is white, and beverage is coffee.
6
7 The function prints:
8
9 Here is your bacon and scrambled eggs with home fries
10 and rye toast. Can I bring you more milk?
11
12 Call it at least 3 different times, scrambling the arguments.
13 """
14
15 def DoBreakfast(meat="bacon", eggs="over easy",
16                  potatos="hash browns", toast="white",
17                  beverage="coffee"):
18     print(f"Here is your {meat} and {eggs} eggs with {potatos}"
19           f" and {toast} toast.",
20           f"Can I bring you more {beverage}?", sep='\n')
21
22 def main():
23     DoBreakfast()
24     DoBreakfast("ham", "basted", "cottage cheese",
25                 "cinnamon", "orange juice")
26     DoBreakfast("sausage", beverage="chai", toast="wheat")
27
28 main()
```

```
$ lab03_3.py
```

```
Here is your bacon and over easy eggs with hash browns and white toast.
```

```
Can I bring you more coffee?
```

```
Here is your ham and basted eggs with cottage cheese and cinnamon toast.
```

```
Can I bring you more orange juice?
```

```
Here is your sausage and over easy eggs with hash browns and wheat toast.
```

```
Can I bring you more chai?
```

```
$
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 4 Star Power

- Unpacking sequences into a call
- Variable number of arguments
- Unpacking sequences into identifiers
- Swapping elements

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

analyze.py

```
1 #!/usr/bin/env python3
2 """Function with many arguments and many return values."""
3
4
5 def Analyze(one, other):
6     total = sum((one, other)) # sum demands a sequence
7     average = total/2.0        # min and max are flexible
8     return min(one, other), max((one, other)), total, average
9
10 def Print(min_number, max_number, total, average):
11     print(f"min_number={min_number},",
12           f"max_number={max_number},",
13           f"total={total:.2f},",
14           f"average={average:.2f}.")
15
16 def main():
17     min_number, max_number, total, average = Analyze(1, 8)
18     Print(min_number, max_number, total, average)
19     """
20     Output so far:
21
22 min_number=1, max_number=8, total=9.00, average=4.50.
23     """
24     answer = Analyze(100, 50)
25     min_number, max_number, total, average = answer
26     Print(min_number, max_number, total, average)
27     """
28     Output from this:
29
30 min_number=50, max_number=100, total=150.00, average=75.00.
31     """
32     numbers = 50, 20
33     answers = Analyze(*numbers)
34     Print(*answers)
35     """
36     Output from this:
37
38 min_number=20, max_number=50, total=70.00, average=35.00.
39     """
40
41 main()
```

```
$ analyze.py
min_number=1, max_number=8, total=9.00, average=4.50.
min_number=50, max_number=100, total=150.00, average=75.00.
min_number=20, max_number=50, total=70.00, average=35.00.
$
```

* in a function call. If we have:

```
colors = ("red", "orange", "yellow")
```

this function call:

```
Fn(*colors)
```

is interpreted as:

```
Fn("red", "orange", "yellow")
```

print_colors.py

```
1 #!/usr/bin/env python3
2 """Demonstrates using * unwrap a sequence into a call.
3 """
4 def PrintColors(one_color, two_color, three_color):
5     print(f"{one_color}, {two_color}, {three_color}.")
6
7 def main():
8     """All sequences are good for unwrapping into a function call.
9     """
10    colors_plain = "red", "orange", "yellow"
11    PrintColors(*colors_plain)
12    colors_parens = ("red", "orange", "yellow")
13    PrintColors(*colors_parens)
14    colors_squares = ["red", "orange", "yellow"]
15    PrintColors(*colors_squares)
16    a_str = "hey"
17    PrintColors(*a_str)
18
19 main()
```

```
$ print_colors.py
red, orange, yellow.
red, orange, yellow.
red, orange, yellow.
h, e, y.
$
```

* in a function definition

```
def Fn(*args):  
    allows any number of arguments in the call:  
    Fn("red", "orange", "yellow")
```

then, inside the function, automatically, you are given:

```
args == ("red", "orange", "yellow")
```

print_favorite_things.py

```
1 #!/usr/bin/env python3  
2 """Demonstrates variable number of arguments with *."""  
3  
4 def PrintFavoriteThings(*things):  
5     print("These are a few of my favorite things: ")  
6     for thing in things:  
7         print(thing, end=" ")  
8     print()  
9  
10 def main():  
11     my_things = "raindrops", "roses", "whiskers", "kittens"  
12     PrintFavoriteThings(*my_things)  
13  
14 main()
```

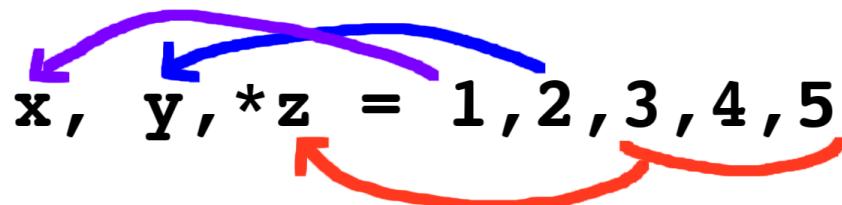
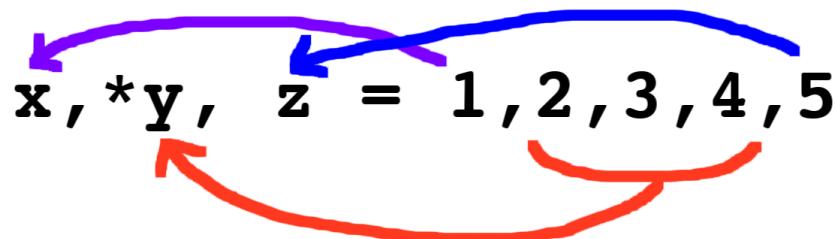
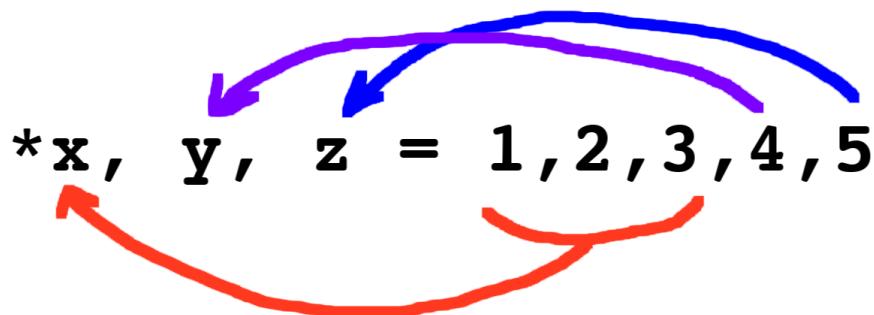
```
$ print_favorite_things_call.py  
These are a few of my favorite things:  
raindrops roses whiskers kittens  
$
```

* For Unpacking Sequences

We've seen that the * unpacks a sequence into individual arguments in a function protocol.

The * is also useful for packing sequence elements into identifiers.

The * gathers the leftovers:



Swapping:

```
>>> bottom, middle, top = ("ice_cream", "whipped cream", "cherry")
>>> print(bottom, middle, top, sep=', ')
ice_cream, whipped cream, cherry
>>> middle, top, bottom = bottom, middle, top
>>> print(bottom, middle, top, sep=', ')
cherry, ice_cream, whipped cream
```



Lab 04 – Exercises:

1. You have the tuple:

```
favorites = ("ham", "poached", "home fried", "English muffin")
```

What's the easy way to use these strings as the first four arguments in a call to your DoBreakfast() function from the last lab?

2. You have:

```
tools = "hammer", "wrench", "scissors", "pliers", "ruler"
```

- (a) How would you isolate "scissors" into the identifier to_cut and put the rest into meaningless identifiers, as few as possible?
- (b) What would rest will result?

```
to_hit, to_torque, to_cut, to_pinch, to_measure, *rest = tools
```

3. Make a MakePizza() function. It takes in any number of arguments. The first two are the main ingredients and are required.

Experiment with these calls, and a few more:

```
ingredients = "anchovies", "garlic", "oregano", "tomato", "peppers"
MakePizza(*ingredients)
MakePizza("cheese", *ingredients)
```

The output from these calls is:

```
Here's your anchovies and garlic pizza, with oregano tomato peppers
Enjoy!
Here's your cheese and anchovies pizza, with garlic oregano tomato peppers
Enjoy!
```

lab04_1.py

```

1 #!/usr/bin/env python3
2 """DoBreakfast() with 5 items. -- From last lab.
3 """
4 def DoBreakfast(meat="bacon", eggs="over easy",
5                 potatos="hash browns", toast="white",
6                 beverage="coffee"):
7     print(f"Here is your {meat} and {eggs} eggs with {potatos}"
8           f" and {toast} toast.",
9           f"Can I bring you more {beverage}?", sep='\n')
10
11 def main():
12     items = ("ham", "poached", "home fried", "English muffin")
13     DoBreakfast(*items)
14
15 main()

```

\$ lab04_1.py

Here is your ham and poached eggs with home fried and English muffin toast.

Can I bring you more coffee?

\$

lab04_2.py

```

1 #!/usr/bin/env python3
2 """
3 Function that shows how to isolate "scissors" into the identifier
4 to_cut}
5 and put the rest into meaningless identifiers, as few as possible?
6 And demonstrate what happens when there are no identifiers left fo
7 r the *.
8 """
9 def DoLab():
10     tools = "hammer", "wrench", "scissors", "pliers", "ruler"
11     to_hit, to_torque, to_cut, *rest = tools
12     print(f"a: to_cut = {to_cut}")
13
14     to_hit, to_torque, to_cut, to_pinch, to_measure, *rest = tools
15     print(f"b: rest = {rest}")
16
17 def main():
18     DoLab()
19
19 main()

```

\$ lab04_2.py

a: to_cut = scissors
b: rest = []

lab04_3.py

```
1 #!/usr/bin/env python3
2 """
3 lab04_3.py - a a MakePizza() function. It takes in any number of
4 arguments. The first two are the main ingredients and are required
5 .
6 """
7 def MakePizza(first, second, *rest):
8     print(f"Here's your {first} and {second} pizza, with", end=' ')
9     for ingredient in rest:
10         print(f" {ingredient}", end=' ')
11     print("\nEnjoy!")
12
13 def main():
14     ingredients = "anchovies", "garlic", "oregano", "tomato", "pep
15 pers"
16     MakePizza(*ingredients)
17     MakePizza("cheese", *ingredients)
18
19 main()
```

```
$ lab04_3.py
```

```
Here's your anchovies and garlic pizza, with oregano tomato peppers
```

```
Enjoy!
```

```
Here's your cheese and anchovies pizza, with garlic oregano tomato peppers
```

```
Enjoy!
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 5 Libraries

- Reference/Assignment
- global declaration
- importing Libraries
- Modules: random, math
- Introspection

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0



Pythonic Thinking: About Identifier Visibility

Assignment:

```
 cups = 10
```

Reference:

```
 value = .25 * cups
```

namespaces.py

```

1 #!/usr/bin/env python3
2 """Global/module-level identifier behavior."""
3
4 pies = 1 # global identifier
5
6 def DoApple():
7     """alters the local pies, shadows the global identifier"""
8     pies = 25
9     print(f"Apple: local pies in DoApple() is {pies}.")
10    pies += 1
11    print(f"Apple ends: local pies = {pies}")
12
13 def DoBerry():
14     """assigning the global identifier pies"""
15     global pies
16     print(f"Berry: global pies is {pies}.")
17     pies *= 10
18     print(f"Berry ends: global pies is {pies}.")
19
20 def DoCherry():
21     """referencing the global identifier"""
22     print(f"Cherry: no problem with referencing global pies = {pies}")
22 }
23
24 def DoMud():
25     """assigning after reference -- reference doesn't work"""
26     print(f"Mud(): pies = {pies}")
27     pies = 999
28
29 DoApple()
30 DoBerry()
31 DoCherry()
32 DoMud()

```

```

$ namespaces.py
Apple: local pies in DoApple() is 25.
Apple ends: local pies = 26
Berry: global pies is 1.
Berry ends: global pies is 10.
Cherry: no problem with referencing global pies = 10
Traceback (most recent call last):
  File "./namespaces.py", line 32, in <module>
    DoMud()
  File "./namespaces.py", line 26, in DoMud
    print(f"Mud(): pies = pies")
UnboundLocalError: local variable 'pies' referenced before assignment
$
```

```
import something
```

is the command to bring something, i.e., other modules (files, libraries, and packages) into your program.

Two important notes:

- import *runs* what it imports.
- import will not import the same module twice.

dice.py

```
1 #!/usr/bin/env python3
2 """Rolls dice, demonstrating random.randrange(), and a
3 tuple with accessing a particular element with an index.
4 """
5 import random
6
7 def Rollem():
8     """Rolls a pair of dice and reports the result."""
9
10    DOUBLES = ("Can't happen", "Snake eyes!", "Little joe!",
11                "Hard six!", "Hard eight!", "Fever!",
12                "Box cars!")
13
14    dice = random.randrange(1, 7), random.randrange(1, 7)
15    answer = f"{dice[0]} and {dice[1]} = {sum(dice)} "
16    if dice[0] == dice[1]:
17        answer += DOUBLES[dice[0]]
18    return answer
19
20 def main():
21     while True:
22         response = input("Ready to roll? 'q' to quit. ")
23         if response and response[0] in "Qq":
24             break
25         print(Rollem())
26 main()
```

```
$ dice.py
Ready to roll? 'q' to quit.
5 and 6 = 11
Ready to roll? 'q' to quit.
6 and 6 = 12 Box cars!
Ready to roll? 'q' to quit.
1 and 1 = 2 Snake eyes!
Ready to roll? 'q' to quit. q
$
```

Lab 05 – Exercises:



1. Write a function called `FlipCoin` that emulates the flip of a coin, returning "heads" or "tails".

Find a way to test it.

2. Use the interpreter to *introspect*:

In Idle or on the Python command line, import the math module:

```
>>> import math
```

Now try:

```
>>> help(math)
```

and there's the documentation for the module! Is that more than you wanted to know?

```
>>> dir(math)
```

This produces a list of all the attributes in the module.

The attributes without leading underscores are meant for you to use via the “dot” operator. You can get help on specific attributes available:

```
>>> help(math.sqrt)
```

Try calling a function call:

```
>>> math.sqrt(9)
```

and, just to check the precision, try this:

```
>>> math.sqrt(1.23456789) * math.sqrt(1.23456789)
```

3. Modify your Lab05.1 by adding a function called `GetHeads(target)` that uses your `FlipCoin()`. This new function accepts an integer as the only argument, `target`, and then flips coins until it gets `target` heads in a row. It returns the number of flips it took to get `target` heads in a row.

Write another function that calls `GetHeads(target)` over and over, `number_of_experiments` times where `number_of_experiments` is given as an argument, as is `target`. Have this new function calculate the average result and print a report.

©Marilyn Davis, 2007-2020

lab05_1.py

```
1 #!/usr/bin/env python3
2 """A coin flipping emulation."""
3 import random
4
5 def FlipCoin():
6     """Simulates the flip of a coin."""
7
8     if random.randrange(0, 2) == 0:
9         return "tails"
10    return "heads"
11
12 def main(num_times):
13     """Driver for testing."""
14     heads = 0
15     for n in range(num_times):
16         if FlipCoin() == "heads":
17             heads += 1
18     print(f'With {num_times} flips, "heads" came up {heads} '
19           f"times, or {heads/num_times:.1%} of the flips.")
20
21 main(100)
```

```
$ lab05_1.py
With 100 flips, "heads" came up 43 times, or 43.0% of the flips.
$ lab05_1.py
With 100 flips, "heads" came up 55 times, or 55.0% of the flips.
```

lab05_3.py

```
1 #!/usr/bin/env python3
2 """Coin flip Experiments, continued."""
3
4 import random
5
6 def FlipCoin():
7     """Simulates the flip of a coin."""
8
9     if random.randrange(0, 2) == 0:
10         return "tails"
11     return "heads"
12
13 def GetHeads(target):
14     """Flips coins until it gets target heads in a row."""
15
16     heads = count = 0
17     while heads < target:
18         count += 1
19         if FlipCoin() == "heads":
20             heads += 1
21         else:          # "tails"
22             heads = 0
23     return count
24
25 def GetAverage(number_of_experiments, target):
26     """Calls GetHeads(target) that number_of_experiments
27     times and reports the average."""
28
29     total = 0
30     for n in range(number_of_experiments):
31         total += GetHeads(target)
32     print(f"Averaging {number_of_experiments} experiments, it took
32 {total/number_of_experiments:.1f} coin flips to get {target} in a
32 row.")
33
34 GetAverage(100, 3)
```

```
$ lab05_3.py
Averaging 100 experiments, it took 13.5 coin flips to get 3 in a row.
$
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 6 Sequences

- Sequence types: str, tuple, list
- Sequence slicing and other manipulations
- list facilities

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

Notes on Sequence types:

str	tuple	list
-----	-------	------

These are not keywords but they are built-in type names so don't use them to name your identifiers.

They have a lot in common.

```
a_string = "abc"           a_tuple = (1, 2, 3)           a_list = [1, 2, 3]
```

Accessing [i] accesses a particular element of any sequence:

a_string[0]	a_tuple[0]	a_list[0]
'a'	1	1

a_string[-1] is the last element: 'c'

a_string[-4] gets an error if it doesn't exist!

Slicing [::] makes a new sequence object:

```
>>> twist = "Somebody danced."
>>>
>>> twist[4:8]                      >>> twist[4:8:2]
'body'                                'bd'
>>>                                     >>> twist[6:2:-1]
                               'dobe'
```

- Syntax: **[start_index : almost_end_index [: increment=1]]**
- [:3] gets first three: 0, 1, 2
- [3:] gets from [3] to the end: 3, 4, ..., last_index
- [:] gets from start to end: 0, 1, ..., last_index

```
>>> buy_it = "wholesale"
>>> print(buy_it[:5], buy_it[5:])
whole sale
```

Slicing never gets an error, just an empty object!

Negative Increment – for reversing a sequence

You use this to reverse any sequence:

```
>>> twist[::-1]
'.decnad ydobemos'
```

Concatenation: `+` is supported for all sequence types:

- Both operands must be the same sequence type.
- The result is always a new sequence of the same type.

Plus augmented assignment `+=` is supported. The rules are:

- `a_string += another_string`
- `a_tuple += another_tuple`
- `a_list += any_sequence_type`

```
>>> some_list = [1, 2, 3]
>>> some_list += [4]
>>> some_list
[1, 2, 3, 4]
>>> some_list += (5, 6)
>>> some_list
[1, 2, 3, 4, 5, 6]
>>> some_list += "abc"
>>> some_list
[1, 2, 3, 4, 5, 6, 'a', 'b', 'c']
```

`+=` is often used to accumulate objects:

```
>>> new_list = []
>>> for i in range(5):
...     new_list += [i]
...
>>> print(new_list)
[0, 1, 2, 3, 4]
```

©Marilyn Davis, 2007-2020

For this sort of task, you must know how to create an empty sequence of each type.

Empty sequence:

`,` `()` `[]`

Sometimes you want to create a one-element, or singleton sequence, to start. Here are singletons of each type:

Singleton sequence:

`'1'` `1,` `[1]`

The one-element tuple is the interesting one:

```
>>> b_tuple
('d', 'e')
>>> b_tuple += 'f'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate tuple (not "str") to tuple
>>> b_tuple += ('f')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate tuple (not "str") to tuple

>>> b_tuple += 'f',
>>> b_tuple
('d', 'e', 'f')
>>>
```

Repetition * is supported for all three sequence types:

```
>>> twister = ("wood", "chuck")
>>> 3 * twister
('wood', 'chuck', 'wood', 'chuck', 'wood', 'chuck')

>>> call = "kitty "
>>> 3 * call
'kitty kitty kitty '

>>> chord = ["do", "mi", "so"]
>>> 2 * chord
['do', 'mi', 'so', 'do', 'mi', 'so']
```

You know that **in** is coupled with **for** to iterate any sequence:

```
>>> for neighbor in neighbors:
...     print(neighbor, end=' ')
...
Amy Joe Alice
>>>
```

Also, **in** is good for testing membership. The result is True or False:

```
>>> neighbors = ["Amy", "Joe"]
>>> "Amy" in neighbors
True

>>> if "Alice" not in neighbors:
...     neighbors += ["Alice"]
...
>>> neighbors
['Amy', 'Joe', 'Alice']
```

Builtin functions: len, max, min, sum, str, repr:

len(seq), max(seq), min(seq), sum(seq) do exactly what you'd expect.

str(seq) and repr(seq) make printable representations of the sequence. str provides a person-friendly string; repr provides a string that the interpreter likes: it can be fed into a call to eval which returns a new object that is == to the original object:

```
>>> c_tuple = 1, 2
>>> str(c_tuple)
'(1, 2)'
>>> repr(c_tuple)
'(1, 2)'
>>> eval(repr(c_tuple))
(1, 2)
>>>
```

is, a keyword, can be useful:

one_object is another_object == True

if they are the exact same object, kept in the same spot in memory.

Factory Functions (Classes or Types):

tuple(seq) makes a tuple of any sequence:

```
>>> c_string = "cloud"
>>> tuple(c_string)
('c', 'l', 'o', 'u', 'd')
```

list(seq) makes a list of any sequence:

```
>>> list(c_string)
['c', 'l', 'o', 'u', 'd']
```

str(seq) makes a str of any sequence:

```
>>> str(list(c_string))
"[‘c’, ‘l’, ‘o’, ‘u’, ‘d’]"
```

What's different?

1. A string can only have characters as members while tuples and lists can have any object as a member.
2. Only lists support item assignment:

a_list[2] = 'a' is good.

But:

a_tuple[2] = 'a' is an error.

and

a_string[2] = 'a' is an error.

Note: This attribute of a list is called *mutability*:

- **tuples** are not mutable.
- **strings** are not mutable.
- **lists** are mutable.

3. Each sequence type has a different list of builtin methods available.

Magic methods, i.e. those whose names start and end with `_` are not meant for us – yet. There are lots of them in each sequence type. They are ignored here.

Strings have lots of very useful builtin methods for string manipulation:

```
>>> dir('')                                [or dir(a_string) or dir(str)]
['__magic__', ..., '__more_magic__', 'capitalize', 'casefold',
'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind',
'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

Tuples have only two methods for us to use:

```
>>> dir(a_tuple)
['__magic__', ..., '__more_magic__', 'count', 'index']
```

Lists have a few methods in common with the other sequence types, and a few, very powerful facilities, of their own:

```
>>> dir(a_list)
['__magic__', ..., '__more_magic__', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

All sequences have `count` and `index` in common:

```
>>> help(list.count)
```

Help on method_descriptor:

```
count(self, value, /)
    Return number of occurrences of value.
```

```
>>> [1, 2, 3, 2, 1].count(2)
2
>>>
```

```
>>> help(str.count)
Help on method_descriptor:

count(...)
    S.count(sub[, start[, end]]) -> int

    Return the number of non-overlapping occurrences of substring
    sub in string S[start:end].  Optional arguments start and end
    are interpreted as in slice notation.
```

```
>>> "supercalifragilisticexpialidocious".count('a')
3
>>>
```

```
>>> help(tuple.count)
Help on method_descriptor:

count(self, value, /)
    Return number of occurrences of value.
```

```
>>> (1, 2, 3, 2, 1).count(1)
2
>>>
```

```
>>> help(list.index)
Help on method_descriptor:

index(self, value, start=0, stop=9223372036854775807, /)
    Return first index of value.

    Raises ValueError if the value is not present.
```

```
>>> [1, 2, 3, 2, 1].index(2, 2)
3
>>>
```

©Marilyn Davis, 2007-2020

```
>>> help(tuple.index)
Help on method_descriptor:

index(...)
    T.index(value, [start, [stop]]) -> integer -- return first
    index of value. Raises ValueError if the value is not present.

-----
```

```
>>> (1, 2, 3, 2, 1).index(1, 2)
4
>>>
```

```
>>> help(str.index)
Help on method_descriptor:

index(...)
    S.index(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Raises ValueError when the substring is not found.

-----
```

```
>>> "supercalifragilisticexpialidocious".index("li", 9)
14
>>>
```

list has some very powerful methods that the other sequence types do not have. That's one benefit of being **mutable**:

```
>>> help(list.copy)
help(list.copy)
Help on method_descriptor:

copy(self, /)
    Return a shallow copy of the list.

-----
```

So `list.copy` is the same as `list[:]`.

Only lists have `append`, `clear`, `extend`, `insert`, `pop`, `remove`, `reverse`, and `sort`. This makes sense because these methods alter (mutate) the sequence, and only lists can be altered.

It's good to understand the difference between `list.append` and `list.extend`:

```
>>> help(list.append)
Help on method_descriptor:
```

```
append(self, object, /)
    Append object to the end of the list.
```

```
>>> help(list.extend)
Help on method_descriptor:
```

```
extend(self, iterable, /)
    Extend list by appending elements from the iterable.
```

`list.append` appends one item to the list.

`list.extend` appends all the items in the iterable to the list.

Watch for this:

```
>>> herbs = ["basil", "thyme"]
>>> herbs.extend("dill")
>>> herbs
['basil', 'thyme', 'd', 'i', 'l', 'l']
```

```
>>> help(list.clear)
help(list.clear)
Help on method_descriptor:
```

```
clear(self, /)
    Remove all items from list.
```

```
>>> money = ["$1", "$2"]
>>> money.clear()
>>> money
[]
```

```
>>> help(list.insert)
help(list.insert)
Help on method_descriptor:
```

```
insert(self, index, object, /)
    Insert object before index.
```

```
>>> a_list
[1, 2]
>>> a_list.insert(1, 3)
>>> a_list
[1, 3, 2]
>>>
```

```
>>> help(list.pop)
Help on method_descriptor:

pop(self, index=-1, /)
    Remove and return item at index (default last).

    Raises IndexError if list is empty or index is out of range.

-----
>>> stack = [1, 2, 3, 2, 1]
>>> stack.pop()
1
>>> stack
[1, 2, 3, 2]
>>>

>>> help(list.remove)
help(list.remove)
Help on method_descriptor:

remove(self, value, /)
    Remove first occurrence of value.

    Raises ValueError if the value is not present.

>>> stack.remove(3)
>>> stack
[1, 2, 2]
>>>

>>> help(list.reverse)
help(list.reverse)
Help on method_descriptor:

reverse(self, /)
    Reverse *IN PLACE*.

>>> help(list.sort)
Help on method_descriptor:

sort(self, /, *, key=None, reverse=False)
    Stable sort *IN PLACE*.
```

The “*IN PLACE*” on the `list.sort` and `list.reverse` methods means that you must do this:

```
>>> some_list = [2, 1, 0, 5, 3, 6, 8, 7, 9, 4]
>>> some_list.sort()
>>> print(some_list)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

If, instead, you write a line like this:

```
>>> some_list = some_list.sort()
>>> print(some_list)
None
>>>
```

you threw away your list!!! The `list.sort()` returns `None` and you moved the `some_list` label onto the `None`.

Happily, there is another builtin function that works on all sequence types, `sorted(sequence)`. It works on all sequence types because it makes a copy of your sequence, which can be expensive (space-wise). But, you can do this:

```
>>> ride = "cab"
>>> print(sorted(ride))
['a', 'b', 'c']
>>>
```

Both `L.sort` and `sorted(iterable)` have an optional argument, `key`. It's the one you want to learn. We'll study it in the next lab.

```
>>> help(sorted)
Help on built-in function sorted in module builtins:
```

```
sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in
    ascending order.
```

A custom key function can be supplied to customize the sort order, and the reverse flag can be set to request the result in descending order.

©Marilyn Davis, 2007/2020

Lab 06 – Exercises:



- Predict the output:

```
>>> says = "Hello Moon"  
  
>>> print(says[6:])                                  >>> print(says[-4:])  
  
-----  
  
>>> print(says[:6] + "World")                    >>> print(says[:])  
  
-----
```

Now give it a try in the interpreter.

- Fill in the blanks:

```
>>> L = [1, 2, 3]  
>>> T = (1, L, 3)  
>>> print(T)
```

```
-----  
  
>>> L[1] = 0  
>>> print(T)
```

```
-----  
  
>>> T[1] = 0  
  
-----
```

©Marilyn Davis, 2007-2020

3. You have:

```
>>> friends = ["Abe", "Bob", "Carl"]
```

How would you make an enemies list from this list but replace "Bob" with "Brian". Your enemies list must be independent from your friends list.

What is the result if you do this:

```
>>> friends = ["Abe", "Bob", "Carl"]
>>> enemies = friends
>>> enemies[1] = "Brian"
>>> print(friends)
```

4. Draw a line from each type to its mutability:

numbers	mutable
strings	
tuples	immutable
lists	

5. Introspect (help and dir) str to find at least three ways to extract "cake" from "birthday cake".
6. From this string: "silver gold copper platinum": use str.split, string manipulation, and str.join to create the following sentence.
"Silver and gold and copper and platinum are only worth money."
7. Write a function that takes in any number of colors. It finds the first two primary colors given and returns the result of mixing them together. If there are not two primary colors, the result should be "grey".
Primary colors are: red, yellow and blue.

©Marilyn Davis, 2007-2020

```
2. >>> L = [1, 2, 3]
>>> T = (1, L, 3)
>>> print(T)
(1, [1, 2, 3], 3)
>>> L[1] = 0
>>> print(T)
(1, [1, 0, 3], 3)
>>> T[1] = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
```

3. You have:

```
>>> friends = ["Abe", "Bob", "Carl"]
```

How would you make an enemies list from this list but replace "Bob" with "Brian". Your enemies list must be independent from your friends list.

```
>>> friends = ["Abe", "Bob", "Carl"]
>>> enemies = friends[:]
>>> enemies[1] = "Brian"
>>> print(friends)
['Abe', 'Bob', 'Carl']
>>> print(enemies)
['Abe', 'Brian', 'Carl']
```

What is the result if you do this:

```
>>> friends = ["Abe", "Bob", "Carl"]
>>> enemies = friends
>>> enemies[1] = "Brian"
>>> print(enemies)
['Abe', 'Brian', 'Carl']
>>> print(friends)
['Abe', 'Brian', 'Carl']
>>>
```

That's a bad way because you make your friends vulnerable.

4. Draw a line from each object to its mutability:

numbers	immutable
strings	"
tuples	"
lists	mutable

5. Introspect (help and dir) str to find at least three ways to end with "cake" from "birthday cake".

```
>>> more = "birthday cake"
>>> more[-4:], more[9:], more[9:13]
('cake', 'cake', 'cake')
>>> more.split()[-1], more.split()[1]
('cake', 'cake')
>>> more.rsplit(None, 1)[1], more.rsplit(None, 1)[-1]
('cake', 'cake')
```

6. From this string: "silver gold copper platinum": use str.split, string manipulation, and str.join to create the following sentence.

"Silver and gold and copper and platinum are only worth money."

```
>>> metals = "silver gold copper platinum"
>>> " and ".join(meals.capitalize().split()) + " are only worth money."
'Silver and gold and copper and platinum are only worth money.'
```

©Marilyn Davis, 2007-2020

lab06_7.py

```
1 #!/usr/bin/env python3
2 """
3 Defines MixColors() which takes in any number of colors and mixes
4 together the first two primary colors.
5 """
6 import random
7
8 def MixColors(*colors):
9     """Find the first two primary colors given and mixes them.
10    Any problem, "grey" is returned.
11    """
12    PRIMARY_COLORS = "red", "yellow", "blue"
13    both_colors = []
14    number_colors = 0
15    for color in colors:
16        if color in PRIMARY_COLORS:
17            both_colors.append(color)
18            number_colors += 1
19            if number_colors == 2:
20                break
21    else:
22        return "grey"
23
24    if both_colors[0] == both_colors[1]:
25        return both_colors[0]
26
27    if "red" in both_colors:
28        if "yellow" in both_colors:
29            return "orange"
30        if "blue" in both_colors:
31            return "purple"
32
33    return "green"
34
35 def TestMixColors():
36     """Tests MixColors with all possible combinations of colors.
37     """
38     colors = ["red", "yellow", "blue", "aqua"]
39     for one in colors:
40         for other in colors:
41             now = one, other
42             print(f"{now[0]} + {now[1]} = {MixColors(*now)}")
43
44     print("Randoms:")
45     colors += "green", "gold"
46
```

```
47     for trial in range(3):
48         random.shuffle(colors)
49         print(f"MixColors({colors}) = {MixColors(*colors)}")
50
51 def main():
52     TestMixColors()
53
54 main()
```

```
$ lab06_7.py
red + red = red.
red + yellow = orange.
red + blue = purple.
red + aqua = grey.
yellow + red = orange.
yellow + yellow = yellow.
yellow + blue = green.
yellow + aqua = grey.
blue + red = purple.
blue + yellow = green.
blue + blue = blue.
blue + aqua = grey.
aqua + red = grey.
aqua + yellow = grey.
aqua + blue = grey.
aqua + aqua = grey.
Randoms:
MixColors(['gold', 'red', 'blue', 'yellow', 'green', 'aqua']) = purple
MixColors(['blue', 'green', 'gold', 'aqua', 'red', 'yellow']) = purple
MixColors(['gold', 'aqua', 'yellow', 'blue', 'red', 'green']) = green
$
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 7 Sorting

- Sorting by a key

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.



Remember the sort method for list

```
>>> help(list.sort)
Help on method_descriptor:

sort(self, /, *, key=None, reverse=False)
    Stable sort *IN PLACE*.
```



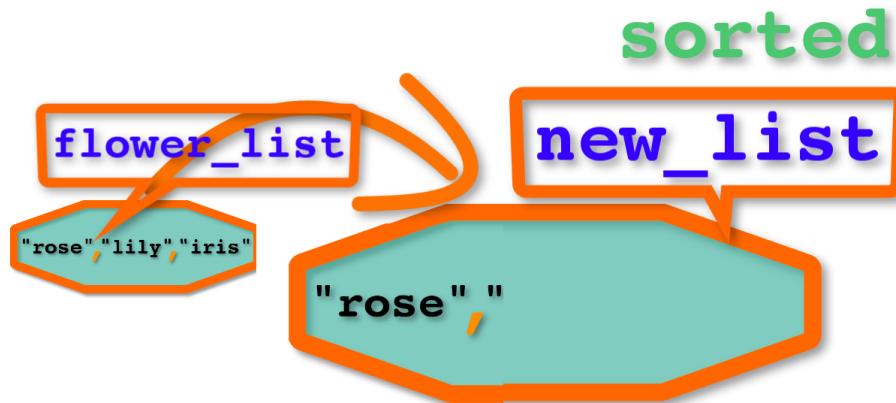
```
>>> flower_list = ["rose", "lily", "iris"]
>>> flower_list.sort()
>>> flower_list
['iris', 'lily', 'rose']
```

And remember the sorted function for any iterable.

```
>>> help(sorted)
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```



```
>>> new_list = sorted(flower_list)
>>> new_list
['iris', 'lily', 'rose']
>>> flower_list
['rose', 'lily', 'iris']
```

Let's see how key works for both facilities (next page):

key_sort.py

```

1 #!/usr/bin/env python3
2 """Demonstrates the key keyword for list.sort.
3 """
4
5 def MagicNumber(string):
6     """Returns the sum of the mapping of each
7     character into its place in the alphabet.
8     """
9     ALPHABET = "abcdefghijklmnopqrstuvwxyz"
10    total = 0
11    for char in string.lower():
12        total += ALPHABET.index(char)
13    return total
14
15 def ReportInternalSortKey(names):
16     """Reports the tuple that the sort will use when this
17     function is named as the 'key'.
18     """
19     sort_keys = []
20     for name in names:
21         sort_keys += [(MagicNumber(name), name)]
22     print(f"MagicNumber keys: {sort_keys}")
23
24 def main():
25     names = ["June", "Alejandro", "Ann", "I", "Izzy"]
26
27     print(f"Names list: {names}")
28     print(f"Default sort: {sorted(names)}")
29     print(f"Sorted by length: {sorted(names, key=len)}")
30     ReportInternalSortKey(names)
31     names.sort(key=MagicNumber)
32     print(f"Sorted by magic number: {names}")
33
34 main()

```

```

$ key_sort.py
    Names list: ['June', 'Alejandro', 'Ann', 'I', 'Izzy']
    Default sort: ['Alejandro', 'Ann', 'I', 'Izzy', 'June']
    Sorted by length: ['I', 'Ann', 'June', 'Izzy', 'Alejandro']
    MagicNumber keys: [(46, 'June'), (71, 'Alejandro'), (26, 'Ann'),
                       (8, 'I'), (82, 'Izzy')]
    Sorted by magic number: ['I', 'Ann', 'June', 'Alejandro', 'Izzy']
$
```

Lab 07 – Exercises:



1. Write a function that collects words from the user until the user does not give one. (Use a list for this. We'll learn about sets later.) If the word given was given before, capitalized or not, report that. When the user stops giving words, return a single string that has all the words given, once each, all upper-case, in alphabetical order, with " AND " between them, and '!!!' at the end.

```
Word please: silver
Word please: copper
Word please: Copper
Copper is already given.
Word please: Gold
Word please: TIN
COPPER AND GOLD AND SILVER AND TIN!!!
$
```

2. You have a list of strings representing names:

```
["Jack Sparrow", "George Washington", "Tiny Sparrow",
 "Jean Ann Kennedy"]
```

Sort them by last name and print them, last name first. The output should be:

```
Kennedy, Jean Ann
Sparrow, Jack
Sparrow, Tiny
Washington, George
```

Hints:

- A function that takes in the name and returns a string with the last name first is doubly useful here.
- `str.split()` and `str.join()` help with string manipulation. In fact, these two string methods are often useful.

©Marilyn Davis, 2007-2020

lab07_1.py

```
1 #!/usr/bin/env python3
2 """
3 Write a function that collects words from the user
4 until the user does not give one. If the word given
5 was given before, capitalized or not, report that.
6 When the user stops giving words, return a single
7 string that has all the words given, once each, all
8 upper-case, in alphabetical order, with " AND "
9 between them, and '!!!!' at the end.
10 """
11 def CollectWords():
12     """Collects words into a sentence.
13     """
14     words = []
15     while True:
16         new_word = input("Word please: ")
17         if not new_word:
18             break
19         upped_word = new_word.upper()
20         if upped_word in words:
21             print(f"{new_word} is already given.")
22         else:
23             words.append(upper_word)
24     words.sort()
25     return ' AND '.join(words) + '!!!!'
26
27 def main():
28     print(CollectWords())
29
30 main()
```

```
$ lab07_1.py
Word please: nuts
Word please: BOLTS
Word please: Nuts
Nuts is already given.
Word please: screws
Word please: tacks
Word please: nails
Word please:
BOLTS AND NAILS AND NUTS AND SCREWS AND TACKS!!!
$
```

lab07_2.py

```
1 #!/usr/bin/env python3
2 """Sorts name strings by last name."""
3
4 def ReverseName(name):
5     """Returns the "last_name, first_names" version
6     of the name.
7     """
8     parts = name.split()
9     return parts[-1] + ', ' + ' '.join(parts[:-1])
10
11 def ReverseName(name):
12     parts = name.split()
13     return parts.pop() + ', ' + ' '.join(parts)
14
15 def ReverseName(name):
16     parts = name.rsplit(None, 1)
17     return parts[-1] + ', ' + parts[0]
18
19 def ReverseName(name):
20     *first_names, last_name = name.split()
21     return last_name + ', ' + ' '.join(first_names)
22
23 def main():
24     NAMES = ["Jack Sparrow", "George Washington",
25              "Tiny Sparrow", "Jean Ann Kennedy"]
26     def ReverseThem():
27         for name in sorted(NAMES, key=ReverseName):
28             print(ReverseName(name))
29     def ReverseThem():
30         reversed_names = []
31         for name in NAMES:
32             reversed_names += [ReverseName(name)]
33         for name in sorted(reversed_names):
34             print(name)
35     ReverseThem()
36 main()
```

```
$ lab07_2.py
Kennedy, Jean Ann
Sparrow, Jack
Sparrow, Tiny
Washington, George
$
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 8 Important Trick

- Important trick:
- `__name__` and "`__main__`"

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

Important Re-Use Trick

`dir()` gives a list of all the names in the current scope. Here is the result of `dir()` when the interpreter first comes up:

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__']
```

Now let's add something:

```
>>> x = 3
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'x']
>>>
>>> import math
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
 '__package__', '__spec__', 'math', 'x']
```

But, what's in that math module? `dir(math)` to find out:

```
>>> dir(math)
['__doc__', '__file__', '__name__', 'acos', (much deleted), 'tan', 'tanh']
>>> def DoIt(x):
...     return x*x
>>> dir()
['DoIt', '__annotations__', '__builtins__', '__doc__', '__loader__',
 '__name__', '__package__', '__spec__', 'math', 'x']
>>>
```

More exploration:

```
>>> print(__name__)
__main__
>>> print(math.__name__)
math
```

We want to study the behavior of the `__name__` attribute in two different circumstances:

1. when it is "`__main__`", and
2. when it is not.

We'll use a small module of code for this study (next page):

trick.py

```
1 #!/usr/bin/env python
2 """Simple code to demonstrate __name__"""
3
4 print(f"trick.py's __name__ is {__name__}")
```

```
$ trick.py
trick.py's __name__ is __main__
```

The output is no surprise, given the experiment we did at the interpreter's prompt.

But, look at the value of `__name__` when `trick.py` is imported:

```
>>> import trick
trick.py's __name__ is trick
>>>
```

And, we can get to it this way:

```
>>> trick.__name__
'trick'
>>>
```

A module's `__name__` matches the file name and the name on the `import` line, unless the module is the main module being run, that is, unless it is the module being run to start the program, in which case the name is "`__main__`".

We use this fact for an important testing/developing trick. Python programmers who know the trick write code that only tests when the module's name is "`__main__`".

Here's some code written that way:

tables.py

```
1 #!/usr/bin/env python3
2 """Unwraps and prints out a 2-D sequence.
3 Note that the testing only happens when this module
4 is the "__main__" module.
5 """
6 def PrintTable(table):
7     """Prints out a 2-D sequence"""
8     for row in table:
9         for column in row:
10            print(column, end=' ')
11            print()
12    print()
13
14 def Test():
15     tests = ([["Hi", "Hola"],
16               (('H', 'i'), ('H', 'o', 'l', 'a')),
17               [["Hi"], ["Hola"]]
18             )
19     for test in tests:
20         print(test)
21         PrintTable(test)
22
23 if __name__ == "__main__":
24     Test()
```

```
$ tables.py
['Hi', 'Hola']
H i
H o l a

((('H', 'i'), ('H', 'o', 'l', 'a'))
H i
H o l a

[['Hi'], ['Hola']]
Hi
Hola

$
```

©Marilyn Davis, 2007-2020

Here's the trick:

```
>>> import tables  
>>>
```

Nothing happened! In particular, the call to `Test()` on line 24 didn't happen. This is because the `__name__` of the imported `tables.py` module is "tables", not "`__main__`", if `__name__ == "__main__"` resolves to `False`, so all the testing gets skipped.

```
>>> tables.__name__  
'tables'
```

Demonstrating another piece of Pythonic magic:

```
>>> help(tables)  
Help on module tables:  
  
NAME  
    tables  
  
FILE  
    /Users/marilyn/Python/MM3/Labs/Lab08_Important_Trick/tables.py  
  
DESCRIPTION  
    tables.py Unwraps and prints out a 2-D sequence.  
    Note that the testing only happens when this module  
    is the __main__ module.  
  
FUNCTIONS  
    PrintTable(table)  
        Prints out a 2-D sequence  
  
    Test()  
        ©Marilyn Davis, 2020/2020
```

All that documentation was lifted from the code. From that, I know I can:

```
>>> tables.PrintTable(((('X', '0', '0'),  
...                      ('X', 'X', ' '),  
...                      ('0', '0', 'X'))))  
  
X 0 0  
X X  
0 0 X
```

Or, I can include the `tables.py` module in some other code.

tables2.py

```
1 #!/usr/bin/env python3
2 """Interactive 2-D string unwrapper.
3 """
4 import tables
5
6 def main():
7     while True:
8         response = input("Say something: ")
9         if not response:
10             break
11         words = response.split()
12         tables.PrintTable(words)
13
14 if __name__ == "__main__":
15     main()
```

```
$ tables2.py
Say something: "Pythonic Thinking"
" P y t h o n i c
T h i n k i n g "
Say something:
$
```

©Marilyn Davis, 2007-2020

Lab 08 – Exercises:



1. Write a function that, when passed a string of characters, returns the number of vowels in the string.

Vowels are 'a', 'e', 'i', 'o', 'u', and sometimes 'y'.

Ignore the *sometimes* 'y' and just count the others.

You might like this construct: `if char in 'aeiou':`

Test it with this sentence: "Math, science, history, unraveling the mysteries, that all started with the big bang!". It has 22 vowels (excluding 'y's).

Be sure that your test does not run when your module is `imported`.

In the interpreter, import your module and run `help` on your module and fix up the documentation.

2. Make another program module, perhaps `lab08_2.py` in the same directory as your `lab08_1.py`.

`lab08_2.py` will ask the user for some input and then report the number of vowels in the input. It will import `lab08_1.py` to do the vowel counting.

For now, place the module that you want to import in the same directory as the module that is importing it. For importing, your module's file name must follow the same rules as any Python identifier, plus it must end with `.py`.

©Marilyn Davis, 2007-2020

lab08_1.py

```
1 #!/usr/bin/env python3
2 """Provides CountVowels, a vowel counting function.
3 """
4 def CountVowels(text):
5     """Returns the number of vowels in the "text" input,
6     excluding 'y's.
7     """
8     count = 0
9     for character in text.lower():
10         if character in "aeiou":
11             count += 1
12     return count
13
14 def main():
15     """Tests the CountVowels function."""
16     for test in ("!", "Math, science, history, unraveling the"
17                  " mysteries, that all started with the big"
18                  " bang!", ""):
19         print(CountVowels(test))
20 if __name__ == "__main__":
21     main()
```

```
$ lab08_1.py
0
22
0
$
```

lab08_2.py

```
1 #!/usr/bin/env python3
2 """Interactive vowel counter."""
3
4 import lab08_1
5
6 def main():
7     while True:
8         count_this = input("Phrase: ")
9         if not count_this:
10             break
11         print(f" ... has {lab08_1.CountVowels(count_this)} vowels.")
11 )
12
13 if __name__ == "__main__":
14     main()
15
```

```
$ lab08_2.py
Phrase: Our whole universe was in a hot dense state,
... has 16 vowels.
Phrase: Then nearly fourteen billion years ago expansion started. Wait..
... has 22 vowels.
```

Phrase:

```
$
>>> import lab08_1
>>> help(lab08_1)
Help on module lab08_1:
```

NAME
lab08_1 - Provides CountVowels, a vowel counting function.

FILE
/Users/marilyn/Python/MM3/Labs/Lab08_Important_Trick/lab08_1.py

FUNCTIONS
CountVowels(text)
 Returns the number of vowels in the "text" input,
 excluding 'y's.

main()
 Tests the CountVowels function.

>>>

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 9 Functional Programming

- List Comprehensions
- Functional Programming

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

List Comprehensions

A *list comprehension* constructs a list:

```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

The syntax of a list comprehension is:

[pattern_using_x_or_not for x in iterable]

```
>>> [0 for x in range(5)]
[0, 0, 0, 0, 0]
>>> [[] for x in range(5)]
[[], [], [], [], []]
>>>
```

You can add an **if** to filter some elements from the sequence:

```
>>> odds = [x for x in range(10) if x % 2]
>>> odds
[1, 3, 5, 7, 9]
```

Your pattern can be a tuple:

```
>>> odd_tuples = [(x, x**2) for x in odds]
>>> odd_tuples
[(1, 1), (3, 9), (5, 25), (7, 49), (9, 81)]
```

You can add another **for** to the syntax to nest looping:

```
>>> four = range(1, 5)
>>> list(four)
[1, 2, 3, 4]
>>> by_tens = list(range(10, 50, 10))
>>> by_tens
[10, 20, 30, 40]
>>> cross_sums = [x + y for x in four for y in by_tens]
      # "x in four" is outer, invariant loop
>>> cross_sums
[11, 21, 31, 41, 12, 22, 32, 42, 13, 23, 33, 43, 14, 24, 34, 44]
```

You can call a function:

```
>>> [str(x) for x in range(13)]
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']
```

Other Fancy Things: Functional Programming

Like some other languages, Python also provides some builtin functional programming functions: `zip()`, `map()` and `filter()`; and a `lambda` keyword. These can be useful.

```
>>> places = [1, 2, 3, 4]
>>> chars = "abcde"
>>> ords = (97, 98, 99, 100)
>>> list(zip(places, chars, ords))
[(1, 'a', 97), (2, 'b', 98), (3, 'c', 99), (4, 'd', 100)]
>>>
```

`zip()` takes any number of sequences and makes a list of tuples and stores it in a `zip` object, which is iterable.

```
tuple[0] has (seq1[0], seq2[0], seq3[0], ...)
tuple[1] has (seq1[1], seq2[1], seq3[1], ...)
tuple[2] has (seq1[2], seq2[2], seq3[2], ...)
```

Notice that `zip` quits whenever any of the sequences is out of elements.

Maybe you'll use `zip()` to make the index into a sequence available in a for loop since `range(len(some_sequence))` gives you a list of the indices into the sequence:

```
>>> yums = "chocolate", "whipped cream", "nuts"
>>> for (index, yum) in zip(range(len(yums)), yums):
...     print(f"yums[{index}] = {yum}")
...
yums[0] = chocolate
yums[1] = whipped cream
yums[2] = nuts
>>>
```

But there's an `enumerate` function just for that purpose:

```
>>> for (index, yum) in enumerate(yums):
...     print(f"yums[{index}] = {yum}")
...
yums[0] = chocolate
yums[1] = whipped cream
yums[2] = nuts
>>>
```

`filter()` and `map()` can always be replaced by a list comprehension and some style guides prefer that you use comprehensions, especially when it's easier to read.

`map()` takes a function defined with one argument, and an iterator, calls the function for each member of the sequence, and forms the resulting list from the return values:

```
>>> def Sqr(x):
...     return x * x
...
>>> list(map(Sqr, range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

This invites a `lambda` expression. `lambda` is a keyword and a `lambda` expression is used instead of a function name. It is *in-place* functionality:

```
>>> list(map(lambda x:x**2, range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Or a list comprehension:

```
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Any sequence is good, if it makes sense. Here we square each of the elements in a tuple to make our resulting list:

```
>>> list(map(lambda x:x**2, (1, 2, 3)))
[1, 4, 9]
```

Let's just multiply by 2 so we can use other sequences types:

```
>>> list(map(lambda x:x*2, ("hi", "ho")))
['hihi', 'hoho']

>>> list(map(lambda x:x*2, "hi"))
['hh', 'ii']
>>>
```

`filter()` needs a function that returns 1 or 0. It will return a sequence whose members passed a 1 back from the function:

```
>>> def IsEven(x):
...     return not x % 2
...
>>> list(filter(IsEven, range(10)))
[0, 2, 4, 6, 8]
```

Again, a `lambda` expression is tempting:

```
>>> list(filter(lambda x:not x % 2, range(10)))
[0, 2, 4, 6, 8]
```

But a list comprehension is more attractive:

```
>>> [x for x in range(10) if not x % 2]
[0, 2, 4, 6, 8]
```

Don't forget `range` alone for this example -> `range(0, 10, 2)`

Lab 09 – Exercises:



Here are some helpful links. Click on *Functional Programming*.

<http://www.python.org/doc/current/tut/node7.html>
<http://www.freenetpages.co.uk/hp/alan.gauld>

Lab 09

There are too many exercises here, unless you, like me, love to work with list comprehensions.

1. What are the results?

```
>>> [str(x) for x in range(3)]  
-----  
>>> [[0] for x in range(3)]  
-----  
>>> [x for x in range(10) if not x % 2]  
-----  
>>> '*' .join([ch for ch in "Chocolate Chips"[9:] if ch.islower()])  
-----
```

Try it in the interpreter to check yourself.

2. For each of the following concoct a list comprehension, and get the work done in one line, using the interpreter if you wish:

(a) Make 201 0's: [0, 0, (198 more), 0]

(b) Make a list comprehension that results in a list of the string version of the digits, 0 through 9:
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

Use that list comprehension and a `str.join` to make this string:

"0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9"

(c) The Wu family has 4 members: "Bo", "Li", "On" and "An". Use this, and a list comprehension, and `str.join` to make this alphabetized report of the family, all in one line:
"An Wu, Bo Wu, Li Wu, On Wu"

(d) Use a list comprehension and `str.join` to make the following column of numbers in one line of code:

0.1
10.1
20.1
30.1

3. Provide a `MakeString` function that receives a glue string, and any number of words and returns a string containing all of the words, each separated by the glue string.

Test it well.

4. Collect `labs.zip` from the our online class site.

Find: `Labs/Lab09_Functional_Programming/colors`

The colors file has the colors typed. Cut and pasting from a pdf often gets errors.

```
"beige", "silver", "charcoal", "royal blue", "aquamarine", "forest green",
"chartreuse", "lime", "golden", "goldenrod", "coral", "salmon",
"hot pink", "fuchsia", "lavender", "plum", "indigo", "maroon", "crimson",
"lemon"
```

Write a function that uses `enumerate` to display them sorted and formatted four across like:

aquamarine	beige	charcoal	chartreuse
coral	crimson	forest green	fuchsia
golden	goldenrod	hot pink	indigo
lavender	lemon	lime	maroon
plum	royal blue	salmon	silver

5. Use list comprehension to make a function that returns a list of strings, each string emulating one card, and the whole list emulating a deck of cards. In your caller, print out the cards, comma-separated, except stick in "and" before the last card.

\$ `lab09_5.py`

```
2 of Clubs, 3 of Clubs, 4 of Clubs, 5 of Clubs, 6 of Clubs, 7 of Clubs,
8 of Clubs, 9 of Clubs, 10 of Clubs, Jack of Clubs, Queen of Clubs, King of Clubs, Ace of Clubs, 2 of Diamonds, 3 of Diamonds, 4 of Diamonds, 5 of Diamonds, 6 of Diamonds, 7 of Diamonds, 8 of Diamonds, 9 of Diamonds, 10 of Diamonds, Jack of Diamonds, Queen of Diamonds, King of Diamonds, Ace of Diamonds, 2 of Hearts, 3 of Hearts, 4 of Hearts, 5 of Hearts, 6 of Hearts, 7 of Hearts, 8 of Hearts, 9 of Hearts, 10 of Hearts, Jack of Hearts, Queen of Hearts, King of Hearts, Ace of Hearts, 2 of Spades, 3 of Spades, 4 of Spades, 5 of Spades, 6 of Spades, 7 of Spades, 8 of Spades, 9 of Spades, 10 of Spades, Jack of Spades, Queen of Spades, King of Spades, Ace of Spades, Joker, and Joker.
```

\$

6. (extra optional practice) Use the `enumerate` function to improve your `lab08_1.py` (about counting vowels in phrases) so that it counts 'y's as vowels according to this specification:

Count a 'y' as a consonant (non vowel):

- if it is the first character.
- if it is preceded by a vowel: boy, way, hey.

Count a 'y' as a vowel:

- if the word ends with *ying*: flying, spying.
- if it is at the end of a word, and is preceded by a consonant: fly.
- if it preceded and followed by a consonant: myth, satyr.

2 In the interpreter:

- (a) Many repetitions of the same element in a list:

```
>>> print([0 for each in range(201)])  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- (b) Numbers as strings:

```
>>> print(" + ".join([str(number) for number in range(10)]))  
0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
```

- (c) The same manipulation on many strings:

```
>>> print(", ".join(["f'{name} Wu" for name in sorted(("Bo", "Li", "On", "An"))]))  
An Wu, Bo Wu, Li Wu, On Wu
```

- (d) Math manipulations:

```
>>> print('\n'.join(['f"{}:6.1f"'.format(10 * mul + .1) for mul in range(20)]))  
0.1  
10.1  
20.1  
30.1  
40.1  
50.1  
60.1  
70.1  
80.1  
90.1  
100.1  
110.1  
120.1  
130.1  
140.1  
150.1  
160.1  
170.1  
180.1  
190.1
```

©Marilyn Davis, 2007-2020

lab09_3.py

```
1 #!/usr/bin/env python3
2 """
3 lab09_3.py Provides a MakeString function that receives a
4 glue string and any number of any objects, returns a string
5 containing all of the elements, each separated by the glue
6 string.
7 """
8 def MakeString(glue, *objects):
9     return glue.join([str(obj) for obj in objects])
10
11 def main():
12     print(MakeString(", ", *range(10, 0, -1)))
13     print(MakeString('*', '1', 2, "word"))
14
15 if __name__ == "__main__":
16     main()
```

```
$ lab09_3.py
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
*2*word
$
```

©Marilyn Davis, 2007-2020

lab09_4.py

```
1 #!/usr/bin/env python3
2 """
3 Defines function that displays colors, 4 on a line.
4 """
5 def DisplayColors():
6     colors = ["beige", "silver", "charcoal", "royal blue",
7               "aquamarine", "forest green", "chartreuse",
8               "lime", "golden", "goldenrod", "coral", "salmon",
9               "hot pink", "fuchsia", "lavender", "plum",
10              "indigo", "maroon", "crimson", "lemon"]
11     for i, color in enumerate(sorted(colors)):
12         print(f"{color:15}", end=' ')
13         if i % 4 == 3:
14             print()
15
16 def main():
17     DisplayColors()
18
19 if __name__ == "__main__":
20     main()
```

```
$ lab09_4.py
aquamarine      beige        charcoal       chartreuse
coral           crimson      forest green   fuchsia
golden          goldenrod    hot pink       indigo
lavender         lemon       lime           maroon
plum            royal blue  salmon          silver
$
```

lab09_5.py

```
1 #!/usr/bin/env python3
2 """Use list comprehensions to make a deck of cards. Print them
3 comma-separated except stick in an "and" before the last.
4 """
5
6 def GetCards():
7     """Return a deck of cards as a list of strings."""
8     values = [str(x) for x in range(2, 11)
9               ] + ["Jack", "Queen", "King", "Ace"]
10    suits = ("Clubs", "Diamonds", "Hearts", "Spades")
11    deck = [f"{value} of {suit}" for suit in suits
12            for value in values] + ["Joker"] * 2
13    return deck
14
15 def main():
16     deck = GetCards()
17     print(f'{", ".join(deck[:-1])}, and {deck[-1]}')
18
19 if __name__ == "__main__":
20     main()
```

```
$ lab09_5.py
2 of Clubs, 3 of Clubs, 4 of Clubs, 5 of Clubs, 6 of Clubs, 7 of Clubs, 8 of Clubs, 9 of Clubs, 10 of Clubs, Jack of Clubs, Queen of Clubs, King of Clubs, Ace of Clubs, 2 of Diamonds, 3 of Diamonds, 4 of Diamonds, 5 of Diamonds, 6 of Diamonds, 7 of Diamonds, 8 of Diamonds, 9 of Diamonds, 10 of Diamonds, Jack of Diamonds, Queen of Diamonds, King of Diamonds, Ace of Diamonds, 2 of Hearts, 3 of Hearts, 4 of Hearts, 5 of Hearts, 6 of Hearts, 7 of Hearts, 8 of Hearts, 9 of Hearts, 10 of Hearts, Jack of Hearts, Queen of Hearts, King of Hearts, Ace of Hearts, 2 of Spades, 3 of Spades, 4 of Spades, 5 of Spades, 6 of Spades, 7 of Spades, 8 of Spades, 9 of Spades, 10 of Spades, Jack of Spades, Queen of Spades, King of Spades, Ace of Spades, Joker, and Joker.
$
```

lab09_6.py

```
1 #!/usr/bin/env python3
2 """Provides CountVowels, a vowel counting function."""
3 import string
4 def CountVowels(phrase):
5     """Returns the number of vowels in the "phrase" input.
6     'y' is not counted if it is the first character.
7         is not counted if it is preceded by a vowel.
8         is counted in "ying" if it is at the end of the word.
9         is counted at the end of the word, if preceded by
10            a consonant.
11        is counted if it is preceded and followed by a
12            consonant.
13 """
14     ALWAYS_VOWELS = "aeiou"
15     spurious = string.punctuation + "0123456789_"
16     count = 0
17     for word in phrase.lower().split():
18         word = word.strip(spurious)
19         l_word = len(word)
20         for index, char in enumerate(word):
21             if char in ALWAYS_VOWELS:
22                 count += 1
23                 continue
24             if char != 'y' or index == 0:
25                 # now, char is 'y' and not the first char
26                 continue
27             if word[index-1] in ALWAYS_VOWELS:
28                 # preceded by a vowel
29                 continue
30             if word.endswith("ying") and index == l_word - 4:
31                 count += 1
32                 continue
33             # now, it is a 'y' preceded by a consonant
34             if (index == l_word - 1 # at end of word
35                 or word[index+1] not in ALWAYS_VOWELS):
36                 # or followed by a consonant
37                 count += 1
38                 continue
39             return count
40 def main():
41     """Tests the CountVowels function."""
42     for test in (
43         """Math, science, history, unraveling the mysteries,
44             that all started with the big bang!""",
45         "boy way hey myth satyr fly flying spying",):
46         print(CountVowels(test), test)
```

```
47
48 if __name__ == "__main__":
49     main()

$ lab09_6.py
24 Math, science, history, unraveling the mysteries,
    that all started with the big bang!
11 boy way hey myth satyr fly flying spying
$
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 10 Dictionaries

- Dictionaries
- Dictionary Comprehensions

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

py_dict_def.py

```
1 #!/usr/bin/env python3
2 """Dictionary implementation for demonstrating a
3 Python dictionary."""
4
5 def SetUpPyDict():
6     py_dict = {}      # empty dictionary
7
8     # initializing a dictionary
9
10    py_dict2 = {"break": "break from a loop and skip the else",
11                "continue": "go to the next iteration of the loop",
12                "for": "set up looping"}
13
14    # Updating py_dict1 with py_dict2's keys and values.
15    # If py_dict2 has keys already in py_dict, py_dict2's
16    # values will replace the old values for the key.
17
18    py_dict.update(py_dict2)
19
20    # And you can just add an entry
21
22    py_dict["pass"] = "throw the ball"
23
24    # If you add an entry with a duplicate key, the new
25    # meaning will be the one that sticks:
26
27    py_dict["pass"] = "do nothing"
28
29    return py_dict
30
31
32
33 def CollectEntries(py_dict):
34     """Collects a bunch of new entries for the dictionary"""
35     while True:
36         word = input("Word: ")
37         if not word:
38             return
39         meaning = input("Meaning: ")
40         py_dict[word] = meaning
41
42
43
44
45
46
```

```
47
48 def FindDefinitions(py_dict):
49     """Reports a key:value pair for a given key"""
50     while True:
51         word = input("Word to find: ")
52         if not word:
53             return
54         try:
55             print(f"{word} : {py_dict[word]}")
56         except KeyError:
57             print(f"{word} is not in the dictionary.")
58
59 def MakePrompt(choices):
60     choice_list = sorted(choices)
61     guts = ", ".join([f"{{choice[0]}}{choice[1:]}" for choice in choice_list])
62     return f"Choose {guts} (enter to quit) "
63
64
65 def PrintEntries(py_dict):
66     """Prints out the dictionary entries, sorted by key"""
67     for word in sorted(py_dict):
68         print(f"{word} : {py_dict[word]}")
69
70 def RunMenu(py_dict):
71     """Runs the user interface for dictionary manipulation."""
72     # The choices dictionary has function names for values.
73     choices = {"add": CollectEntries, "find": FindDefinitions,
74                "print": PrintEntries}
75     prompt = MakePrompt(choices)
76
77     while True:
78         raw_choice = input(prompt)
79         if not raw_choice:
80             break
81         given_choice = raw_choice[0].lower()
82         for maybe_choice in choices:
83             if maybe_choice[0] == given_choice:
84                 # The appropriate function is called
85                 # using the dictionary value for the name
86                 # of the function.
87                 choices[maybe_choice](py_dict)
88                 break
89             else:
90                 print(f"{raw_choice} is not an acceptable choice.")
91
92 def main():
93     py_dict = SetUpPyDict()
94     RunMenu(py_dict)
```

```
95
96 if __name__ == "__main__":
97     main()

$ py_dict_def.py
Choose (a)dd, (f)ind, (p)rint (enter to quit) p
break : break from a loop and skip the else
continue : go to the next iteration of the loop
for : set up looping
pass : do nothing
Choose (a)dd, (f)ind, (p)rint (enter to quit) a
Word: yield
Meaning: return and start here with next call
Word:
Choose (a)dd, (f)ind, (p)rint (enter to quit) f
Word to find: for
for : set up looping
Word to find: range
range is not in the dictionary.
Word to find:
Choose (a)dd, (f)ind, (p)rint (enter to quit)
$
```

©Marilyn Davis, 2007-2020

dict_comprehension.py

```
1 #!/usr/bin/env python3
2 """
3 Demonstrates a dictionary comprehension.
4 """
5 def MakeSqrs(seq):
6     return {str(n):n*n for n in seq}
7
8 def MakeDict(*seqs):
9     return {data[0]:data[1:] for data in zip(*seqs)}
10
11 def main():
12     print(MakeSqrs(range(5)), end="\n\n")
13     people = ["Lorena", "Isabel", "Ricardo", "Paco"]
14     instruments = ["guitar", "salterio", "marimba", "vihuela"]
15     gender = ["woman", "woman", "man", "man"]
16     band = MakeDict(people, instruments, gender)
17     print('\n'.join(f'{k:>10}:{band[k]}' for k in sorted(band)))
18
19 if __name__ == "__main__":
20     main()
21
```

```
$ dict_comprehension.py
'0': 0, '1': 1, '2': 4, '3': 9, '4': 16

    Isabel:('salterio', 'woman')
    Lorena:('guitar', 'woman')
    Paco:('vihuela', 'man')
    Ricardo:('marimba', 'man')
$
```

Lab 10 – Exercises:



1. The built-in `dict()` function provides a very flexible way to create a dictionary:

```
squares = dict(one=1, two=4, three=9)
```

Try it in the Python shell. Then see what functions are available for dictionaries:

```
dir(squares)                  or dir(dict)
```

Popular functions are `keys`, `items` and `values`. Check out the documentation for these functions like this:

```
help(squares.keys)
```

If you have any questions about these or others that interest you, raise your hand. In particular, you will like `dict.items()` as you do this lab.

You also might like the behaviors of `for`, `in`, and `sorted` when applied to dictionaries.

Collect `labs.zip` from the our online class site.

Find: `Labs/Lab10_Dictionaries/py_dict_def.py`

Edit the program so that is has another choice in the menu: `(d)efinitions`.

This new option will print out the dictionary alphabetically by the meanings:

```
Choose (a)dd, (d)efinitions, (f)ind, (p)rint (enter to quit) d
break out of a loop and skip the else : break
do nothing : pass
go to the next iteration of the loop : continue
set up looping : for
Choose (a)dd, (d)efinitions, (f)ind, (p)rint (enter to quit)
$
```

2. Make a dictionary of the words in this text:

Keep a fire burning in your eye.

The key will be the lower-case version of the word, the value will be the word backwards.

3. Here are the temperatures at noon Monday through Friday:

80, 82, 85, 87, 76

Here is the percent chance of precipitation at noon Monday through Friday:

0, 2, 44, 100, 0

Here is the wind speed and direction at noon Monday through Friday:

2SW, 7SW, 3S, 10S, 2W

Make a dictionary with the keys being each day's name, and the values are a tuple of (temp, precip, wind).

lab10_1.py

```
1 #!/usr/bin/env python3
2 """Dictionary implementation for demonstrating a dictionary.
3
4     ... with a new option will print out the dictionary
5         alphabetically by the meanings. """
6
7 from py_dict_def import * # Careful of this!
8
9 def ListDefinitions(py_dict):
10     """Prints out the dictionary, alphabetically by the
11         meanings"""
12     defs = []
13     for k, v in py_dict.items():
14         defs += [(v, k)]
15     # or: defs = [(v, k) for (k, v) in py_dict.items()]
16     defs.sort()
17     for (v, k) in defs:
18         print(v, ':', k)
19
20 def ListDefinitions(py_dict):
21     """Prints out the dictionary, alphabetically by
22         the meanings -- implemented via the sort key option."""
23     def ValueKey(k):
24         return py_dict[k]
25
26     for each in sorted(py_dict, key=ValueKey):
27         print(py_dict[each], ':', each)
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

```
47
48 def RunMenu(py_dict):
49     """Runs the user interface for dictionary manipulation."""
50     # The choices dictionary has function names for values.
51     choices = {"add": CollectEntries,
52                "definitions": ListDefinitions,
53                "find": FindDefinitions, "print": PrintEntries}
54     prompt = MakePrompt(choices)
55
56     while True:
57         raw_choice = input(prompt)
58         if not raw_choice:
59             break
60         given_choice = raw_choice[0].lower()
61         for maybe_choice in choices:
62             if maybe_choice[0] == given_choice:
63                 # The appropriate function is called
64                 # using the dictionary value for the name
65                 # of the function.
66                 choices[maybe_choice](py_dict)
67                 break
68             else:
69                 print(f"{raw_choice} is not an acceptable choice.")
70
71 def main():
72     py_dict = SetUpPyDict()
73     RunMenu(py_dict)
74
75 if __name__ == "__main__":
76     main()
```

```
$ lab10_1.py
Choose (a)dd, (d)einitions, (f)ind, (p)rint (enter to quit) d
break from a loop and skip the else : break
do nothing : pass
go to the next iteration of the loop : continue
set up looping : for
Choose (a)dd, (d)einitions, (f)ind, (p)rint (enter to quit)
$
```

lab10_2.py

```
1 #!/usr/bin/env python3
2 """
3 Demonstrates a dictionary comprehension.
4 """
5 def ReverseDict(text):
6     return {k.lower():k.lower()[:-1] for k in text.split()}
7
8 def main():
9     word_dict = ReverseDict("Keep a fire burning in your eye.")
10    print('\n'.join(f'{k:>10}:{word_dict[k]}'
11                   for k in sorted(word_dict)))
12
13 if __name__ == "__main__":
14     main()
15
```

```
$ lab10_2.py
      a:a
      burning:gninrub
      eye...eye
      fire:erif
      in:ni
      keep:peek
      your:ruoy
$
```

©Marilyn Davis, 2007-2020

lab10_3.py

```
1 #!/usr/bin/env python3
2 """
3 A dictionary with the keys being each day's name , and
4 the values are a tuple of (temp , precip , wind) .
5 """
6 def ShowDailies(*seqs):
7     days = "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday"
8     dailies = {data[0]:data[1:] for data in zip(days , *seqs)}
9     for day in sorted(dailies , key=lambda d:days.index(d)):
10         print(f"{day:>20}:",
11               '\t'.join(f"{n:>10}" for n in dailies[day]))
12
13 def main():
14     temps = [80 , 82 , 85 , 87 , 76]
15     precip = [0 , 2 , 44 , 100 , 0]
16     wind = ["2SW" , "7SW" , "3S" , "10S" , "2W"]
17     ShowDailies(temps , precip , wind)
18
19 if __name__ == "__main__":
20     main()
```

```
$ ./lab10_3.py
    Monday:      80          0        2SW
    Tuesday:     82          2        7SW
    Wednesday:   85          44       3S
    Thursday:    87         100      10S
    Friday:      76          0        2W
$
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 11 File IO

- File I/O
- Context handler:with keyword
- Reading unicode
- sys library
- Modules: shutil, tempfile

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0

read_file.py

```
1 #!/usr/bin/env python3
2 """Demonstrates reading a file line by line, automatic
3 exception handling in a "context"."""
4
5 def PrintFile(f_name):
6     try:
7         with open(f_name) as open_file:
8             for line in open_file:
9                 print(line, end='')
10    except OSError as info:
11        print(info)
12
13 def main():
14     PrintFile("ram_tzu.txt")
15     print()
16     PrintFile("absent_file")
17
18 if __name__ == "__main__":
19     main()
```

```
$ read_file.py
```

```
Ram Tzu knows this:
```

```
When God wants you to do something,
you think it's your idea.
```

```
[Errno 2] No such file or directory: 'absent_file'
```

```
$
```

raw_read_file.py

```
1 #!/usr/bin/env python3
2 """Reading a file and catching errors, and closing the file.
3 """
4 def PrintFile(file_name):
5     try:
6         file_obj = open(file_name)
7         for line in file_obj:
8             print(line, end='')
9     except OSError as info:
10         print(info)
11     finally:
12         try:
13             file_obj.close()
14         except UnboundLocalError:
15             pass
16
17 def main():
18     PrintFile("ram_tzu.txt")
19     print()
20     PrintFile("absent_file")
21
22 if __name__ == "__main__":
23     main()
```

```
$ raw_read_file.py
Ram Tzu knows this:
When God wants you to do something,
you think it's your idea.
```

```
[Errno 2] No such file or directory: 'absent_file'
```

read_write_file.py

```
1 #!/usr/bin/env python3
2 """Demonstrates writing a file."""
3
4 def CapFile(starting_file):
5     file_name, ext = starting_file.rsplit('.', 1)
6     write_file_name = file_name + "_caps." + ext
7     with open(starting_file) as read_file:
8         with open(write_file_name, "w") as write_file:
9             for line in read_file:
10                 write_file.write(line.upper())
11     return write_file_name
12
13 def main(starting_file):
14     import os
15     new_file = CapFile(starting_file)
16     os.system(f"cat {new_file}")
17
18 if __name__ == "__main__":
19     main("ram_tzu.txt")
```

```
$ read_write_file.py
RAM TZU KNOWS THIS:
WHEN GOD WANTS YOU TO DO SOMETHING,
YOU THINK IT'S YOUR IDEA.
```

\$

read_unicode.py

```
1 #!/usr/bin/env python3
2 """Demonstrates changing encoding.
3 """
4 import locale, os
5
6 def ReadFile(f_name, encoding):
7     with open(f_name, encoding=encoding) as file_object:
8         for line in file_object:
9             for char in line:
10                 if ord(char) > 128:
11                     print(line, end=' ')
12
13 def ChangeEncoding(f_name, now_in, change_to):
14     path, name = os.path.split(f_name)
15     front, ext = name.rsplit('.', 1)
16     new_f_name = os.path.join(path, f'{front}_{change_to}.{ext}')
17     try:
18         with open(f_name,
19                    encoding=now_in) as read_file:
20             with open(new_f_name, "w",
21                        encoding=change_to) as write_file:
22                 for line in read_file:
23                     write_file.write(line)
24     except OSError as info:
25         print(info)
26     return new_f_name
27
28 def main():
29     default_encoding = locale.getpreferredencoding()
30     the_file = '../Lab040_Formatting_Strings/uni.py'
31     print("Reading with preferred encoding for the computer = "
32           f'{default_encoding}:')
33     ReadFile(the_file, default_encoding)
34
35     other_encoding = "utf-16"
36     new_file = ChangeEncoding(the_file, default_encoding,
37                               other_encoding)
38     print(f'\nReading with "{other_encoding}":')
39     ReadFile(new_file, other_encoding)
40
41     print(f'\nReading with wrong encoding: {default_encoding}:')
42     ReadFile(new_file, default_encoding)
43
44 if __name__ == "__main__":
45     main()
```

```
$ read_unicode.py
Reading with preferred encoding for the computer = UTF-8:
yen_symbol = ¥
# print("b'¥' =", b'¥')
# print("b'¥' =", b'¥')

Reading with "utf-16":
yen_symbol = ¥
# print("b'¥' =", b'¥')
# print("b'¥' =", b'¥')

Reading with wrong encoding: UTF-8:
Traceback (most recent call last):
  File "./read_unicode.py", line 45, in <module>
    main()
  File "./read_unicode.py", line 42, in main
    ReadFile(new_file, default_encoding)
  File "./read_unicode.py", line 8, in ReadFile
    for line in file_object:
  File "/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/codecs.py", line 322
, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xff in position 0: invalid start byte
$
```

©Marilyn Davis, 2017-2020

Notes about files

```
>>> help(open)
help(open)
Help on built-in function open in module io:

open(file, mode='r', buffering=-1, encoding=None,
      errors=None, newline=None, closefd=True, opener=None)
    Open file and return a stream.  Raise OSError upon failure.

[some deleted]
```

In text mode, if encoding is not specified the encoding used is platform dependent: locale.getpreferredencoding(False) is called to get the current locale encoding. (For reading and writing raw bytes use binary mode and leave encoding unspecified.)

To learn the default encoding on your computer:

```
>>> import locale
>>> locale.getpreferredencoding()
'UTF-8'
>>>
```

Ways to read a file. First:

```
file_object = open('my.txt')
```

then:

```
text = file_object.read()          --> text is a string of the whole file

text = file_object.read(1024)       --> reads 1024 bytes.

lines = file_object.readlines()     --> lines is a list of strings, each a line

one_line = file_object.readline()   --> reads one line.

for line in file_object:           ???
    print(line, end='')             for line in file_object.readlines():
                                    print(line, end='')

^ this form iterates through the   ^ this form puts the whole file in
file, line by line.                  memory at the same time!
```

You want add end="" in your print statement because each line already has a newline.

For an open file object, any mode, you can manipulate the physical read/write header:

```
current_position = file_object.tell()  
gives the read/write head position.
```

To move it:

```
file_object.seek(offset, whence=0)  
whence = 0 -> beginning of file  
= 1 -> current position  
= 2 -> from end of file - use a negative offset
```

You can ask:

file_object.closed	True/False
file_object.mode	last mode
file_object.name	str name

Ways to write a file. First:

```
file_object = open('my.txt', 'w')    <-- or mode can be 'a' or 'r+'.
```

then:

```
file_object.write(str)  
file_object.writelines(list_of_strs)
```

sys_demo.py

```
1 #!/usr/bin/env python3
2 """Demonstrating the sys module."""
3
4 import sys
5
6 def DemoOpenStreams():
7     """Demos stderr, stdout and stdin. Also sys.exit()"""
8     sys.stderr.write("You can write to stderr.\n")
9     sys.stderr.flush()
10    print("You might want to redirect print's output.", file=sys.s
10 tder)
11    sys.stdout.write("A fancier way to write to stdout.\n")
12    sys.stdout.flush()
13    print("Type something: ")
14    text = sys.stdin.readline()
15    print("You said:", text)
16
17 def DemoCommandLine():
18     """Shows the command line."""
19     print(f"This program is named: {sys.argv[0]}")
20     print(f"The command line arguments are: {sys.argv[1:]}")
21
22 def main():
23     DemoCommandLine()
24     DemoOpenStreams()
25 main()
26
```

```
$ sys_demo.py -muchos tacos 2> error_output
# On Mac/Unix command line, 2> file_name
# redirects stderr to file_name
# On Windows, stderr goes to stdout
This program is named: ./sys_demo.py
The command line arguments are: ['-muchos', 'tacos']
A fancier way to write to stdout.
Type something:
jalapenos
You said: jalapenos

$ cat error_output
You can write to stderr.
You might want to redirect print's output.
```

Lab 11 – Exercises:



1. Remember these functions:

```
>>> def Move(label):
...     label = "Anything"
...
>>> def Use(label):
...     label[1] = ":^)"
...
```

Predict the results. Use your pencil and don't spend too much time.

```
>>> a_dict = {'1': "one"}
>>> Move(a_dict)
>>> print(a_dict)
```

```
>>> a_dict = {'1': "one"}
>>> Use(a_dict)
>>> print(a_dict)
```

2. Remember lab09_Functional_Programming/lab09_6.py which provides the CountVowels(phrase) function. You need to copy it into your working directory to do this exercise. Soon we'll learn to import modules from anywhere on our directory structure so is the last time you'll have to copy a module.

Import lab09_6.py into a new module. In this module, make a CountVowelsInFile(file_name) function which returns the number of vowels in the file.

3. From Lab11_File_IO you need do_swap.py and cats.txt.

do_swap.py, contains a do_swap.DoSwap(text, apple, orange) function. It has some major faults but we will write a good one if we study regular expressions.

Change (read and rewrite) the file Labs/Lab11_File_IO/cats.txt so that every *cat* becomes a *dog* and every *dog* becomes a *cat*. Use do_swap.DoSwap. Be sure to put your functionality into a function. You'll need it for the next lab. Check the results of importing your module into the interpreter and running help(your_module).

4. Lab11_File_IO/movies.csv is a comma-separated list of movies and movie details. Discover which movies required unicode to read.

5. Use the lab09_6.py again. This time use it to make another interactive vowel counter. This time, use sys.stdin and sys.stdout instead of input and print.

Lab 11_1

```
>>> def Move(label):
...     label = "Anything"
...
>>> def Use(label):
...     label[1] = ":^)"
...
>>> a_dict = {'1': "one"}
>>> Move(a_dict)
>>> print(a_dict)
{'1': 'one'}
```

```
>>> Use(a_dict)
>>> print(a_dict)
{'1': 'one', 1: ':^)'}
```

```
>>>
```

©Marilyn Davis, 2007-2020

lab11_2.py

```
1 #!/usr/bin/env python3
2 """
3 lab11_2.py
4 Write a function that takes a file name and returns
5 how many times a non-y vowel appears in the file's text.
6 """
7 import sys
8 import lab09_6 as count_vowels
9
10 def CountVowelsInFile(file_name):
11     """Returns the number of vowels in the file named.
12     """
13     vowel_count = 0
14     with open(file_name) as file_obj:
15         for line in file_obj:
16             vowel_count += count_vowels.CountVowels(line)
17     return vowel_count
18
19 def main():
20     try:
21         file_name = sys.argv[1]
22     except IndexError:
23         file_name = input("File name: ")
24     if not file_name:
25         file_name = "cats.txt"
26     try:
27         print(f"There are {CountVowelsInFile(file_name)} vowels in
27 {file_name}.")
28     except IOError as info:
29         print(info, file=sys.stderr)
30         sys.exit(1)
31
32 if __name__ == "__main__":
33     main()
34
```

```
$ lab11_2.py
File name: cats.txt
There are 96 vowels in cats.txt.
$ lab11_2.py
File name:
There are 96 vowels in cats.txt.
$ lab11_2.py
File name: Bogus
[Errno 2] No such file or directory: 'Bogus'
```

lab11_3.py

```
1 #!/usr/bin/env python3
2 """
3 Copy the cats.txt into your area. Change the file so
4 that all the cats become dogs and dogs become cats.
5 """
6 import do_swap
7 import shutil
8
9 def Swapper(file_name, apple, orange):
10     """ Changes the text in the file, replacing apples with
11     oranges and oranges with apples."""
12     try:
13         with open(file_name, "r+") as open_file:
14             text = open_file.read()
15             text = do_swap.DoSwap(text, apple, orange)
16             open_file.seek(0, 0)
17             open_file.truncate()
18             open_file.write(text)
19     except OSError as info:
20         print(f"Can't open {file_name} because {info}")
21
22 def main():
23     shutil.copy("cats.txt", "cats2.txt")
24     Swapper("cats2.txt", "cat", "dog")
25     Swapper("www.txt", "www", "yyy")
26
27 if __name__ == "__main__":
28     main()
```

```
$ lab11_3.py
Can't open www.txt because [Errno 2] No such file or directory: 'www.txt'
$ cat cats2.txt
```

In my house we have 3 dogs who love to tease our old cat. The old cat gets quite bewildered when they take turns running in front of him and getting in his way. He steps around one dog, then the next dog, then the next dog, just to find the first dog again in his way. However, at nap time, they all curl up together, 3 dogs and one old cat.

lab11_3_2.py

```
1 #!/usr/bin/env python3
2 """lab11_3 again. This time using the builtin file
3 iterator, so that all the file isn't in memory at one time,
4 and using tempfile."""
5 import os
6 import shutil
7 import tempfile
8 import do_swap
9
10 def Swapper(file_name, apple, orange):
11     """ Changes the text in the file, replacing apples with
12     oranges and oranges with apples."""
13     with tempfile.NamedTemporaryFile(delete=False) as temp_file:
14         # temp_file is an open file object, open for writing
15         try:
16             with open(file_name) as open_file:
17                 for line in open_file:
18                     swapped_line = do_swap.DoSwap(line, apple, ora
19 nge)
20                     temp_file.write(bytes(swapped_line,
21 encoding="utf-8"))
22             except OSError as info:
23                 print(f"Problem with {file_name} because {info}")
24             os.rename(temp_file.name, file_name) # For Windows, first:
25                                         # os.remove(file_name)
26 def main():
27     shutil.copy("cats.txt", "cats2.txt")
28     Swapper("cats2.txt", "cat", "dog")
29     Swapper("www.txt", "www", "yyy")
30
31 if __name__ == "__main__":
32     main()
```

```
$ lab11_3_2.py
Problem with www.txt because [Errno 2] No such file or directory: 'www.txt'
$ cat cats2.txt
```

In my house we have 3 dogs who love to tease our old cat. The old cat gets quite bewildered when they take turns running in front of him and getting in his way. He steps around one dog, then the next dog, then [The rest of the output is deleted.]

lab11_4.py

```
1 #!/usr/bin/env python3
2 """Finding the movie titles that need unicode."""
3
4 def ReportTitlesNeedingUnicode(data_file = "movies.csv"):
5     movie_titles_needing_unicode = []
6     with open(data_file, encoding="utf-8") as f_obj:
7         for line in f_obj:
8             title, *fields = line.split(',')
9             title = title.strip()
10            # if you don't strip it, there is a strange white-space after each title.
11            for char in title:
12                if ord(char) > 127:
13                    movie_titles_needing_unicode.append(title)
14                    break
15    print(', '.join(movie_titles_needing_unicode))
16
17
18 def main():
19     ReportTitlesNeedingUnicode()
20
21 if __name__ == "__main__":
22     main()
23
```

```
$ lab11_4.py
WALL·E
$
```

©Marilyn Davis, 2007-2020

lab11_5.py

```
1 #!/usr/bin/env python3
2 """Interactive vowel counter."""
3
4 import lab09_6
5 import sys
6
7 def main():
8     while True:
9         sys.stdout.write("Phrase: ")
10        sys.stdout.flush()
11        count_this = sys.stdin.readline()
12        if count_this == '\n':
13            break
14        sys.stdout.write(f"... has {lab09_6.CountVowels(count_this
14 )} vowels.\n")
15
16 if __name__ == "__main__":
17     main()
18
```

```
$ lab11_5.py
Phrase: Be yourself.
... has 4 vowels.
Phrase: Be truthful.
... has 3 vowels.
Phrase: Be present.
... has 3 vowels.
Phrase:
$
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 12 Portable Python

- Module: os
- Walking A Directory

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

os Module

Portability issues are packaged into the os module:

Linux and OS X:

```
>>> import os
>>> os.linesep
'\n'
>>> os.sep
'/'
>>> import os
>>> os.linesep
'\r\n'
>>> os.sep
'\'
```

Windows:

Here is a sample of some functions available in os. Use help(os) for more details:

```
chdir(...)
    Change the current working directory to the specified path.
chmod(...)
    Change the access permissions of a file.
getcwd(...)
    Return a string representing the current working directory.
listdir(...)
    Return a list containing the names of the entries in the
    directory.
mkdir(...)
    Create a directory.
remove(...)
    Remove a file (same as unlink(path)).
rename(...)
    Rename a file or directory.
rmdir(...)
    Remove a directory.
stat(...)
    Perform a stat system call on the given path.
```

©Marilyn Davis, 2007-2020

os.path:

There are a lot of important functions in the os.path module, which you get for free when you import os.

```
>>> dir(os.path)
```

```
[('__all__', '__builtins__', '__doc__', '__file__', '__name__',
 '_resolve_link', '_varprog', 'abspath', 'altsep', 'basename',
 'commonprefix', 'curdir', 'defpath', 'devnull', 'dirname',
 'exists', 'expanduser', 'expandvars', 'extsep', 'getatime',
 'getctime', 'getmtime', 'getsize', 'isabs', 'isdir', 'isfile',
 'islink', 'ismount', 'join', 'lexists', 'normcase', 'normpath',
 'os', 'pardir', 'pathsep', 'realpath', 'samefile',
 'sameopenfile', 'samestat', 'sep', 'split', 'splitdrive',
 'splitext', 'stat', 'supports_unicode_filenames', 'walk']
```

os.path.split and os.path.join are particularly helpful:

```
>>> import os
>>> where_am_i = os.getcwd()
>>> where_am_i
'/Users/marilyn/Python/MM3/Labs/Lab12_Portable_Python'
>>> parts = os.path.split(where_am_i)
>>> parts
('/Users/marilyn/Python/MM3/Labs', 'Lab12_Portable_Python')
>>> os.path.join(parts[0], "..")
'/Users/marilyn/Python/MM3/Labs/..'
>>>
```

```
>>> help(os.walk)
```

Help on function walk in module os:

```
walk(top, topdown=True, onerror=None, followlinks=False)
    Directory tree generator.
```

For each directory in the directory tree rooted at top (including top itself, but excluding '.' and '..'), yields a 3-tuple

```
    dirpath, dirnames, filenames
```

dirpath is a string, the path to the directory. dirnames is a list of the names of the subdirectories in dirpath (excluding '.' and '..'). filenames is a list of the names of the non-directory files in dirpath. Note that the names in the lists are just names, with no path components. To get a full path (which begins with top) to a file or directory in dirpath, do os.path.join(dirpath, name).

```
[more details deleted]
```

```
>>>
```

walk_.py

```
1 #!/usr/bin/env python3
2 """Demonstrates the os.walk function, one of many
3 very useful things given to us in the os module.
4 """
5 import time
6 import os
7
8 def Process(this_dir, dir_names, file_names):
9     print("In", os.path.abspath(this_dir))
10    for node_name in sorted(dir_names + file_names):
11        whole_name = os.path.join(this_dir, node_name)
12        if os.path.isdir(whole_name):
13            print(f"    directory: {node_name}")
14        else:
15            print(f"""      {node_name}:
16 {time.ctime(os.path.getmtime(whole_name))}""")
17
18 if __name__ == "__main__":
19     for (this_dir, dir_names, file_names) in os.walk("cats"):
20         Process(this_dir, dir_names, file_names)
```

```
$ walk_.py
In /Users/marilyn/Python/MM3/Labs/Lab12_Portable_Python/cats
    cats.txt:
        Thu Nov 29 11:58:46 2018
    directory: deep_cats
    more_cats.txt:
        Thu Nov 29 11:58:46 2018
In /Users/marilyn/Python/MM3/Labs/Lab12_Portable_Python/cats/deep_cats
    cats.txt:
        Thu Nov 29 11:58:46 2018
    directory: deeper_cats
    more_cats.txt:
        Thu Nov 29 11:58:46 2018
In /Users/marilyn/Python/MM3/Labs/Lab12_Portable_Python/cats/deep_cats/deeper_cats
    cats.txt:
        Thu Nov 29 11:58:46 2018
    more_cats.txt:
        Thu Nov 29 11:58:46 2018
$
```

Lab 12 - Exercises:



For these exercises, because we don't yet know how to import from an arbitrary directory, you need to copy modules into your working directory.

From Lab11_File_IO, or from your work, copy lab11_2.py and lab11_3.py

1. Make a module that imports lab11_2.py. Use it to provide a `CountVowelsInDir(dir_name)` function which return the number of vowels in all the files in the directory and all subdirectories.
 2. Draw a line from each type to its mutability:

numbers

strings

tuples

mutable

lists

immutable

dictionaries

dictionary keys

3. (Optional but the solutions demonstrates the use of the `shutil` and `tempfile` libraries.) Import your `lab11_3.py`, or my `lab11_3_2.py`, program into a new program that swaps `cat` and `dog` throughout the files in the `cats` directory structure. Use `os.walk`.

Check that it worked by printing all the files in the directory structure, also using an `os.walk`

©Marilyn Davis, 2007-2020

lab12_1.py

```
1 #!/usr/bin/env python3
2 """
3 Reports the number of vowels in the directory structure given.
4 """
5 import os, sys
6 import lab11_2 as count_vowels_in_file
7
8 def CountVowelsInDir(dir_name):
9     total = 0
10    for this_dir, dir_names, file_names in os.walk(dir_name):
11        for file_name in file_names:
12            whole_path = os.path.join(this_dir, file_name)
13            total += \
14                count_vowels_in_file.CountVowelsInFile(whole_path)
15    return total
16
17 def main(dir_name="cats"):
18     print(f"{CountVowelsInDir(dir_name)} vowels in {dir_name} direc-
19 tory")
20 if __name__ == "__main__":
21     if len(sys.argv) > 1:
22         main(sys.argv[1])
23     else:
24         main()
```

```
$ lab12_1.py
576 vowels in cats directory
```

lab12_2.py

```
1 #!/usr/bin/env python3
2 """Uses os.walk() and the previous exercise to change
3 all the cats to dogs and dogs to cats through a directory
4 structure."""
5
6 import os
7 import sys
8 import shutil
9 import lab11_3_2 as swapper # was the previous exercise
10
11 def SwapTextFiles(starting_dir, swappers):
12     """ Changes the text in a files in the starting_dir
13     directory structure. swappers is a tuple of strings
14     to swap.
15     """
16     apple, orange = swappers # let errors raise to caller
17
18     for this_dir, dir_names, file_names in os.walk(starting_dir):
19         for file_name in file_names:
20             whole_path = os.path.join(this_dir, file_name)
21             swapper.Swapper(whole_path, apple, orange)
22
23 def PrintDeepFiles(starting_dir):
24     for this_dir, dir_names, file_names in os.walk(starting_dir):
25         for file_name in file_names:
26             whole_path = os.path.join(this_dir, file_name)
27             print(whole_path + ':')
28             with open(whole_path) as file_object:
29                 for line in file_object:
30                     print(line, end=' ')
31             print("----")
32
33 def main():
34     if len(sys.argv) > 1:
35         starting_dir = sys.argv[1]
36     else:
37         try:
38             shutil.rmtree("cats2")
39         except OSError: # not there
40             pass
41         shutil.copytree("./cats", "./cats2")
42         starting_dir = "cats2"
43     PrintDeepFiles(starting_dir)
44     SwapTextFiles(starting_dir, ("cat", "dog"))
45     PrintDeepFiles(starting_dir)
46
```

```
47 if __name__ == "__main__":
48     main()
```

```
$ lab12_2.py
cats2/more_cats.txt:
```

In my house we have 3 cats who love to tease our old dog. The old dog gets quite bewildered when they take turns running in front of him and getting in his way. He steps around one cat, then the next cat, then the next cat, just to find the first cat again in his way. However, at nap time, they all curl up together, 3 cats and one old dog.

```
---
cats2/cats.txt:
[much deleted]
---
cats2/deep_cats/deeper_cats/cats.txt:
```

In my house we have 3 dogs who love to tease our old cat. The old cat gets quite bewildered when they take turns running in front of him and getting in his way. He steps around one dog, then the next dog, then the next dog, just to find the first dog again in his way. However, at nap time, they all curl up together, 3 dogs and one old cat.

```
---
```

©Marilyn Davis, 2007-2020

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 13 set and from

- Passing mutables/immutables into functions
- Importing with the `from` keyword
- `sets`
- `nonlocal` declaration

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

mutability_calls.py

```
1 #!/usr/bin/env python3
2 """
3 Demonstrates what happens in the caller when passing in mutable
4 and immutable types.
5 """
6 def Manipulate(this):
7     this[1] = "tomatoes"
8
9 def TestManipulate(this):
10    print(f"this = {this}")
11    try:
12        Manipulate(this)
13    except TypeError as msg: # msg has more info.
14        print(f"Oops {this} broke with {msg}.")
15    else:
16        print(f"OK. Now we have {this}.")
17
18 def main():
19     a_list = ["sirloin", "t-bone", "chuck"]
20     a_str = "coleslaw"
21     a_tuple = ("berries", "peaches", "watermelon")
22     a_number = 4.4
23     for thing in a_list, a_str, a_tuple, a_number:
24         TestManipulate(thing)
25
26 if __name__ == "__main__":
27     main()
```

```
$ mutability_calls.py
this = ['sirloin', 't-bone', 'chuck']
OK. Now we have ['sirloin', 'tomatoes', 'chuck'].
this = coleslaw
Oops coleslaw broke with 'str' object does not support item assignment.
this = ('berries', 'peaches', 'watermelon')
Oops ('berries', 'peaches', 'watermelon') broke with 'tuple' object does not support item as
signment.
this = 4.4
Oops 4.4 broke with 'float' object does not support item assignment.
$
```

mod0.py

```
1 #!/usr/bin/env python3
2 """You can use the "from" keyword to import a module so that
3 you can bring the specified attributes of the module into
4 your local namespace.
5 """
6
7 from math import pi
8
9 def CalculateArea(radius):
10     return pi * radius * radius
11
12 print(f"{CalculateArea(3):.1f}")
13
```

```
$ mod0.py
28.3
$
```

mod1.py

```
1 #!/usr/bin/env python3
2 """You can use -> from module import *
3 to bring all the attributes into your local namespace.
4 But this is usually considered to be bad practice.
5 You lose track of which attributes are local and are
6 not local. And, here's another problem.
7 """
8
9 immutable = 1
10 mutable = [1, 2, 3]
11
12 def PrintImmutable():
13     print(immutable)
14
15 def PrintMutable():
16     print(mutable)
```

```
>>> from mod1 import *
>>> PrintImmutable()
1
>>> immutable = 2
>>> PrintImmutable()      <-- gets mod1 version
1
>>> print(immutable)     <-- gets local version
2
>>> PrintMutable()
[1, 2, 3]
>>> mutable[1] = 'a'    <-- a list is mutable!
>>> print(mutable)
[1, 'a', 3]      <-- local version
>>> PrintMutable()
[1, 'a', 3]      <-- mod1 version
>>>
```

mod1 is not accessible:

```
>>> mod1.immutable = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'mod1' is not defined
```

```
>>> import mod1
>>> mod1.immutable = "NO"
>>> PrintImmutable()
NO
>>> print(immutable)
2

>>> mod1.mutable[0] = "harsh"
>>> PrintMutable()
['harsh', 'a', 3]
>>> mod1.PrintMutable()
['harsh', 'a', 3]
>>>
```

You can import certain attributes:

```
>>> from mod1 import PrintImmutable, PrintMutable
>>> PrintImmutable()
1
```

©Marilyn Davis, 2007-2020

mod2.py

```
1 #!/usr/bin/env python3
2 """You can keep your identifiers from being imported with
3 from ... import * by prefixing a '_' to the name."""
4
5 _immutable = 1
6 _mutable = [1, 2, 3]
7
8 def PrintImmutable():
9     print(_immutable)
10
11 def PrintMutable():
12     print(_mutable)
13
```

```
>>> from mod2 import *
>>> dir()
['PrintImmutable', 'PrintMutable', '__builtins__', '__doc__', '__name__',
 '__package__']
```

However, if they are asked for explicitly, they come:

```
>>> from mod2 import _immutable, _mutable
>>> dir()
['PrintImmutable', 'PrintMutable', '__builtins__', '__doc__', '__name__',
 '__package__', '_immutable', '_mutable']
```

mod3.py

```
1 #!/usr/bin/env python3
2 """__all__ specifies attributes for export"""
3
4 __all__ = ("PrintImmutable", "PrintMutable")
5
6 immutable = 1
7 mutable = [1, 2, 3]
8
9 def PrintImmutable():
10     print(immutable)
11
12 def PrintMutable():
13     print(mutable)
```

```
>>> from mod3 import *
>>> dir()
['PrintImmutable', 'PrintMutable', '__builtins__', '__doc__', '__name__',
 '__package__']

>>> from mod3 import immutable, mutable
>>> dir()
['PrintImmutable', 'PrintMutable', '__builtins__', '__doc__', '__name__',
 '__package__', 'immutable', 'mutable']
>>>
```

Another SAMPLE RUN:

```
>>> import mod3 as other
>>> other.immutable = 88
>>> other.PrintImmutable()
88
>>>
```

©Marilyn Davis, 2007/2020

The set builtin container type: unique items with logical operations

From <https://docs.python.org/3/tutorial/datastructures.html#sets>

```
>>> basket = {"apple", "orange", "apple", "pear", "orange", "banana"}  
>>> print(basket) # duplicates have been removed  
{'orange', 'banana', 'pear', 'apple'}  
  
>>> "orange" in basket # fast membership testing  
True  
>>> "crabgrass" in basket  
False
```

Demonstrates set operations on unique letters from two words:

```
>>> spell = set("abracadabra")  
>>> spell # unique letters in spell  
{'a', 'r', 'b', 'c', 'd'}  
>>> charm = {*"alacazam"}  
>>> charm  
{'l', 'z', 'm', 'a', 'c'}  
  
>>> spell - charm # letters in spell but not in charm  
{'r', 'd', 'b'} # same as spell.difference(charm)  
  
>>> spell | charm # letters in spell or charm or both  
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'} # same as spell.union(charm)  
  
>>> spell & charm # letters in both spell and charm  
{'a', 'c'} # same as spell.intersection(charm)  
  
>>> spell ^ charm # letters in spell or charm but not both  
{'r', 'd', 'b', 'm', 'z', 'l'} # same as spell.symmetric_difference(charm)
```

Perhaps most useful is set.add:

```
>>> help(set.add)
```

Help on method_descriptor:

```
add(...)  
    Add an element to a set.
```

This has no effect if the element is already present.

```
>>>
```

demo_set.py

```
1 #!/usr/bin/env python3
2 """Reads all the .py files in the current directory and
3 lists the unique words.
4 """
5 import os, string
6
7 def FindUniqueWords():
8     """Finds the set of unique words in all the ".txt" files in
9     the current directory.
10    """
11    unique_words = set()
12    for file_name in os.listdir('.'):
13        if not file_name.endswith(".txt"):
14            continue
15        with open(file_name) as f_obj:
16            for line in f_obj:
17                for word in line.split():
18                    word = word.strip(string.punctuation)
19                    unique_words.add(word.lower())
20    return unique_words
21
22 def FormatWords(words, width=50):
23     def ProcessWord(word):
24         nonlocal line_len # for namespace between here and global
25         word_len = len(word)
26         line_len += word_len
27         if line_len > width:
28             line_len = word_len
29             ret_lines.append(''.join(this_lines_words))
30             this_lines_words.clear()
31             this_lines_words.append(word)
32
33     ret_lines = []
34     this_lines_words = []
35     line_len = 0
36     words = sorted(list(words))
37     for word in words[:-1]:
38         word = word + ", "
39         ProcessWord(word)
40     ProcessWord("and " + words[-1] + '.')
41     ret_lines.append(''.join(this_lines_words))
42     return '\n'.join(ret_lines)
43
44 def main():
45     unique_words = FindUniqueWords()
46     if unique_words:
```

```
47         print(FormatWords(unique_words))
48
49 if __name__ == '__main__':
50     main()
51
```

```
$ cd ../Lab11_File_IO
$ ../Lab13_set_and_from/demo_set.py
3, again, all, and, around, at, bewildered, cat,
cats, curl, do, dog, dogs, find, first, front,
gets, getting, god, have, he, him, his, house,
however, idea, in, it's, just, knows, love, my,
nap, next, of, old, one, our, quite, ram,
running, something, steps, take, tease, the,
then, they, think, this, time, to, together,
turns, tzu, up, wants, way, we, when, who, you,
and your.
$
```

©Marilyn Davis, 2007-2020

Lab 13 – Exercises:



1. Give this some thought, and a quick guess. Don't test, you learn more by getting it wrong than by testing. We will discuss the answers.

We have two functions defined:

```
>>> def Move(label):           >>> def Use(label):  
...     label = "Anything"    ...     label[1] = ":^)"  
...                                         ...
```

Predict the results. Use your pencil and don't spend too much time.

Continued:

```
>>> a_list = [1, 2, 3]          >>> a_string = "bird"  
>>> Move(a_list)             >>> Move(a_string)  
>>> print(a_list)            >>> print(a_string)
```

```
-----  
>>> a_list = [1, 2, 3]          >>> a_string = "bird"  
>>> Use(a_list)              >>> Use(a_string)  
>>> print(a_list)            >>> print(a_string)
```

```
-----  
>>> a_tuple = (4, 5, 6)        >>> a_number = 3  
>>> Move(a_tuple)             >>> Move(a_number)  
>>> print(a_tuple)            >>> a_number
```

```
-----  
>>> a_tuple = (4, 5, 6)        >>> a_number = 3  
>>> Use(a_tuple)              >>> Use(a_number)
```

```
-----  
>>> print(a_tuple)            >>> print(a_number)
```

2. Make a function that gathers some texts interactively, stopping when the user doesn't give any text, and reports the non-white-space characters that all the texts have in common.

3. Study these two modules, just enough to make a quick guess:

lab13_3a.py

```
1 #!/usr/bin/env python3
2
3 def GoOnVacation():
4     print("Viva Mexico!")
5
6 def main():
7     GoOnVacation()
8
9 if __name__ == "__main__":
10    main()
11
```

lab13_3b.py

```
1 #!/usr/bin/env python3
2
3 from lab13_3a import *
4
5 def GoOnVacation():
6     print("Aloha Hawaii!")
7
8 if __name__ == "__main__":
9    main()
```

When lab13_3a.py is run, where are we going?

When lab13_3b.py is run, where we going?

©Marilyn Davis, 2007-2020

```
lab13_1

>>> def Move(label):
...     label = "Anything"
...
>>> def Use(label):
...     label[1] = ":^)"
...

>>> a_list = [1, 2, 3]
>>> Move(a_list)
>>> print(a_list)
[1, 2, 3]

>>> a_list = [1, 2, 3]
>>> Use(a_list)
>>> print(a_list)
[1, ':^)', 3]

>>> a_tuple = (4, 5, 6)
>>> Move(a_tuple)
>>> print(a_tuple)
(4, 5, 6)
>>>

>>> Use(a_tuple)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<stdin>", line 2, in Use
      TypeError: 'tuple' object does not support item assignment

>>> print(a_tuple)
(4, 5, 6)
```

```
>>> def Move(label):
...     label = "Anything"
...
>>> def Use(label):
...     label[1] = ":^)"
...
>>> a_string = "bird"
>>> Move(a_string)
>>> print(a_string)
bird

>>> Use(a_string)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<stdin>", line 2, in Use
      TypeError: 'str' object does not support item assignment

>>> print(a_string)
bird

>>> a_number = 3
>>> Move(a_number)
>>> print(a_number)
3

>>> Use(a_number)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<stdin>", line 2, in Use
      TypeError: 'int' object does not support item assignment

>>> print(a_number)
3
>>>
```

lab13_2.py

```
1 #!/usr/bin/env python3
2 """
3 A function that gather texts interactively and reports the
4 non-white-space characters that all the texts have in common.
5 """
6 import string
7 def ReportCharsInCommon():
8     """Collects number_of_texts texts from the user and reports
9     the printable characters that all the texts have in common.
10    """
11    common_chars = set(string.printable)
12    while True:
13        text = input("Type something: ")
14        if not text:
15            break
16        text_chars = set(''.join(text.split()))
17        common_chars = common_chars.intersection(text_chars)
18
19    in_common_list = sorted(list(common_chars))
20    if in_common_list:
21        print(f"Your texts have the characters: {', '.join(in_common_list[:-1])} and {in_common_list[-1]} in common.")
22    else:
23        print(f"Your texts have no characters in common.")
24
25 def main():
26     ReportCharsInCommon()
27
28 if __name__ == "__main__":
29     main()
```

```
$ lab13_2.py
Type something: Why do you stay in prison when the door is so wide open?
Type something: Don't wait any longer, dive in the ocean, leave and let the sea be you.
Type something: Set your life on fire. Seek those who fan your flames.
Type something:
Your texts have the characters: a, e, h, i, n, o, r, s, t, u, w and y in common.
```

lab13_3a.py

```
1 #!/usr/bin/env python3
2
3 def GoOnVacation():
4     print("Viva Mexico!")
5
6 def main():
7     GoOnVacation()
8
9 if __name__ == "__main__":
10    main()
11
```

```
$ lab13_3a.py
Viva Mexico!
```

lab13_3b.py

```
1 #!/usr/bin/env python3
2
3 from lab13_3a import *
4
5 def GoOnVacation():
6     print("Aloha Hawaii!")
7
8 if __name__ == "__main__":
9    main()
```

```
$ lab13_3b.py
Viva Mexico!
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 14 Packages

- copy library
- Python Packages

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

copies.py

```
1 #!/usr/bin/env python3
2 """copies.py
3
4 Demonstrating shallow and deep copies. With dict.copy(), you
5 get a shallow copy where the dictionary values are references
6 of the same values that are in the original dictionary.
7 """
8
9 import copy
10
11 nest = {'a':[1, 2, 3], 'b':[11, 12, 13]}
12 nest_copy = nest.copy()
13 print(f"    nest: {nest}")
14 print()
15
16 print("Hopefully, if you change one, you don't change the other.")
17 nest['b'] = [21, 22, 23]
18 print("After nest['b'] = [21, 22, 23]")
19 print(f"    nest: {nest}")
20 print(f"nest_copy: {nest_copy}")
21 print()
22
23 print("OK. That worked. But what if you change an element of a list (item assignment)? Because the copy has a reference to the list, both reflect the change.")
24 nest['a'][1] = "surprise"
25 print(''After nest['a'][1] = "surprise"''')
26 print(f"    nest: {nest}")
27 print(f"nest_copy: {nest_copy}")
28 print()
29
30 print("""If you don't like that behavior, you can do a "deepcopy".
30 """)
31 deep_copy = copy.deepcopy(nest)
32 nest['a'][1] = "independence"
33 print(''After nest['a'][1] = "independence"''')
34 print(f"    nest:{nest}")
35 print(f"deep_copy: {deep_copy}")
```

```
$ copies.py
nest: 'a': [1, 2, 3], 'b': [11, 12, 13]
```

Hopefully, if you change one, you don't change the other.

```
After nest['b'] = [21, 22, 23]
```

```
nest: 'a': [1, 2, 3], 'b': [21, 22, 23]
```

```
nest_copy: 'a': [1, 2, 3], 'b': [11, 12, 13]
```

OK. That worked. But what if you change an element of a list (item assignment)? Because the copy has a reference to the list, both reflect the change.

```
After nest['a'][1] = "surprise"
```

```
nest: 'a': [1, 'surprise', 3], 'b': [21, 22, 23]
```

```
nest_copy: 'a': [1, 'surprise', 3], 'b': [11, 12, 13]
```

If you don't like that behavior, you can do a "deepcopy".

```
After nest['a'][1] = "independence"
```

```
nest: 'a': [1, 'independence', 3], 'b': [21, 22, 23]
```

```
deep_copy: 'a': [1, 'surprise', 3], 'b': [21, 22, 23]
```

```
$
```

©Marilyn Davis, 2007-2020

Python Packages

Packages are a collection of modules, in a directory structure.

Here's part of the directory structure for our class materials:

```
/Labs
|-- /Lab09_Functional_Programming
|   |-- lab09_4.py    # About cards
|   |-- lab09_6.py    # About vowels
|-- /Lab13_Dictionaries
|   |-- py_dict_def.py
|-- /Lab11_File_IO
|-- /Lab12_Portable_Python
|-- /Lab14_Packages
|   |-- /apple
|   |   |-- /banana
|   |   |   |-- total_text.py
|   |   |-- /work_here
|   |-- numbers.txt
|   |-- poker.py
|   |-- total_text.py
```

I want to import the Lab09_Functional_Programming/lab09_4.py module while working in Lab14_Packages.

There are two steps:

1. The import facility uses `sys.path` to look for imports. I must identify the root directory of my package and put it as the first entry in `sys.path`:

```
>>> import sys
>>> sys.path
[', ', '/Library/Frameworks/Python.framework/Versions/3.7/lib/python37.zip', (more deleted) ]
>>> sys.path.insert(0, "..")
>>> sys.path
['..', '..', '/Library/Frameworks/Python.framework [more deleted]
```

Now the Labs directory is first on the search path for imports.

You don't want to use the absolute path name in this step or you won't be able to sell your package.

2. To have a successful import, I must concoct an `import` line that starts with a subdirectory of some directory on my `sys.path`. That starting directory is dotted with any subdirectories I want the `import` facility to descend into. After the last dot comes the name of the module I want to import, but without the `.py`.

```
>>> import Lab09_Functional_Programming.lab09_4
```

Note that all importable modules *must* end in `.py` and must not have any other dots in their name; and that the `import` line has no `.py`'s.

Now I can introspect my imported module:

```
>>> dir(Lab09_Functional_Programming.lab09_4)
['GetCards', '__builtins__', '__doc__', '__file__', '__name__', 'main']
>>> help(Lab09_Functional_Programming.lab09_4.GetCards)
Help on function GetCards in module Lab09_Functional_Programming.lab09_4:

GetCards()
    Return a deck of cards as a list of strings.

>>> Lab09_Functional_Programming.lab09_4.GetCards()[:5]
['2 of Clubs', '3 of Clubs', '4 of Clubs', '5 of Clubs', '6 of Clubs']
>>>
```

Here you might like the keyword as:

```
>>> import Lab09_Functional_Programming.lab09_4 as cards
```

Then you can type cards instead of Lab09_Functional_Programming.lab09_4, and still keep your code clear about the fact that cards.GetCards is defined in another module:

```
>>> cards.GetCards() [36]
'Queen of Hearts'
>>>
```

There is a new problem to face when we do it in a new module.

We will import Lab09_Functional_Programming/lab09_4.py so that we can use the GetCards() function in a new program, Lab14_Packages/poker.py. The relevant parts of our directory tree:

```
/Labs
|-- /Lab09_Functional_Programming
|   |-- lab09_4.py
|-- /Lab14_Packages
|   |-- poker.py
```

Our code begins:

```
#!/usr/bin/env python3
"""
Deals a poker hand.
"""

import sys, random
sys.path.insert(0, "..")
import Lab09_Functional_Programming.lab09_4 as cards
```

The code works great until we try running it from a different directory.

Here, we run the new module from the Labs directory, like this:

```
$ Lab14_Packages/poker.py
```

But, the import doesn't work:

```
./Lab14_Packages/poker.py
Traceback (most recent call last):
  File "./Lab14_Packages/poker.py", line 8, in <module>
    import Lab09_Functional_Programming.lab09_4 as cards
ModuleNotFoundError: No module named 'Lab09_Functional_Programming'
```

We'll import os in our poker.py so we can put this line before the import line and see the problem:

```
print(f"Inserted into sys.path is: os.path.abspath(sys.path[0])")
```

Now we see some useful information when we run it. Before the ModuleNotFoundError we see this output:

```
Inserted into sys.path is: /Users/marilyn/Python/MM3
```

It could not achieve the import because sys.path.insert(0, "..") put the parent directory of the interpreter's run location (MM3 is the parent of Labs in my file system) on the sys.path, not the parent of the module we are running.

To solve this, we will use our module's `__file__` attribute, set by the interpreter to be the absolute path of our module, like this:

```
sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
print(f"Inserted into sys.path is: os.path.abspath(sys.path[0])")
import Lab09_Functional_Programming.lab09_4 as cards
```

Now our new module can be run or imported from anywhere:

```
$ Lab14_Packages/poker.py
Inserted into sys.path is: /Users/marilyn/Python/MM3/Labs
```

... and the import succeeds!

poker.py

```
1 #!/usr/bin/env python3
2 """
3 Deals a poker hand.
4 Demonstrates an import from within a package.
5 """
6 import sys, os, random
7 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
8 print(f"Inserted into sys.path is: {os.path.abspath(sys.path[0])}")
8 )
9 import Lab09_Functional_Programming.lab09_5 as cards
10
11 def DealPokerHand(shuffled_deck, number_of_cards=5):
12     hand = [shuffled_deck.pop() for card in range(number_of_cards)]
12 ]
13     return hand
14
15 def main():
16     the_deck = cards.GetCards()
17     random.shuffle(the_deck)
18     ordinals = "First", "Second", "Third", "Fourth"
19     for player in range(4):
20         print(f"{ordinals[player]} player: {''.join(DealPokerHan
20 d(the_deck))}")
21
22 if __name__ == "__main__":
23     main()
```

```
$ Lab14_Packages/poker.py    # running from the labs dir
Inserted into sys.path is: /Users/marilyn/Python/MM3/Labs
First player: 4 of Diamonds, 7 of Diamonds, Jack of Hearts, 9 of Clubs, 2 of Clubs
Second player: Queen of Clubs, 8 of Clubs, 9 of Spades, Ace of Hearts, 3 of Spades
Third player: 10 of Diamonds, 3 of Hearts, 9 of Hearts, 10 of Hearts, Joker
Fourth player: Ace of Spades, King of Spades, 2 of Diamonds, 6 of Clubs, Ace of Clubs
$
```

Lab 14 – Exercises:



Collect `labs.zip` from the our online class site.

You need:

- `Labs/Lab14_Packages/apple` directory structure
- `Labs/Lab14_Packages/numbers.txt`

1. The apple directory structure looks like:

```
/apple
|--- /banana
|   |--- total_text.py
|--- /work_here
```

Bring up the interpreter while the `work_here` directory is your current working directory, alter the `sys.path` to force the `import` facility to look into the `apple` directory; and use dot notation to import `total_text.py` into the interpreter and use `dir` and `help` to discover why it is useful to you for this exercise, and how to use it. The exercise:

Now, do the import in a program in the `work_here`. Have your program get the file path interactively, and report the total of all the numbers in the file. My test file is `numbers.txt`. Put the test file anywhere you like.

If you have time and interest, make your program robust, i.e., catch all possible errors and react appropriately.

Also, test it by running it from your `apple` directory.

2. Now, make a program in the `Lab14_Packages` directory that imports your previous module and walks the `Labs` directory structure, adding up all the numbers in all the files in the directory structure.

3. **Predict the output:**

```
>>> xlist = [1, 2, ['a', 'b']]
>>> xlist_ref = xlist
>>> xlist_copy = xlist[:]
>>> import copy
>>> xlist_deepcopy = copy.deepcopy(xlist)
>>> xlist[0] = '***'
>>> xlist[2][0] = '***'
>>> print xlist
```

```
>>> print xlist_ref
```

```
>>> print xlist_copy
```

```
>>> print xlist_deepcopy
```

```
-----
```

4. More predictions:

```
>>> xdict = {'a':[1, 2, 3], 'b':[4, 5, 6]}
>>> xdict_ref = xdict
>>> xdict_copy = xdict.copy()
>>> import copy
>>> xdict_deepcopy = copy.deepcopy(xdict)
>>> xdict['a'] = 'ok'
>>> xdict['b'][0] = 888
>>> xdict['c'] = 'watermelon'
>>> print xdict
```

```
-----
```

```
>>> print xdict_ref
```

```
-----
```

```
>>> print xdict_copy
```

```
-----
```

```
>>> print xdict_deepcopy
```

```
-----
```

©Marilyn Davis, 2007-2020

©Marilyn Davis, 2007-2020

lab14_1.py

```
1 #!/usr/bin/env python3
2 """
3 Write a function that reads a file and finds all the
4 numbers in the file and adds them up.
5 """
6
7 import sys, os
8
9 # putting apple on the search path
10 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
11
12 import banana.total_text      # banana must have __init__.py
13
14 def TotalIt(stream, total=0):
15     """Returns the sum of all the numbers in stream, which is
16     an open file object."""
17     try:
18         for line in stream:
19             total += banana.total_text.TotalText(line)
20     except UnicodeDecodeError:
21         print(stream.name)
22         sys.exit()
23     return total
24
25 def TotalFile(file_name):
26     """Returns the sum of all the numbers in the file."""
27     with open(file_name) as open_file:
28         return TotalIt(open_file)
29
30 def main():
31     while True:
32         try:
33             file_name = input("File name: ")
34         except (KeyboardInterrupt, EOFError):
35             print()
36             break
37         if file_name == '':
38             break
39         total = TotalFile(file_name)
40         if total:
41             print(file_name, "totals to", total)
42
43 if __name__ == "__main__":
44     main()
```

```
$ lab14_1.py
```

```
File name: ../../numbers.txt
../../numbers.txt totals to 205.8
File name: no_file
I can't read "no_file".
File name:      (I hit Ctrl-d -->EOFError)
$ lab14_1.py
File name:      (I hit Ctrl-c Ctrl-c -->KeyboardInterrupt)
$ cat ../../numbers.txt
Here is 1. Add 2 makes 3 or maybe 12, depending on how you operate.
You might like 2.2 and that's enough unless you like "8.8" or maybe
1 more or maybe 87. . . 5. But she said ".3" Let's do 88!
$
```

©Marilyn Davis, 2007-2020

lab14_2.py

```
1 #!/usr/bin/env python3
2 """Using os.walk to accumulate total through the files walked.
3 """
4
5 import os
6 import apple.work_here.lab14_1 as totaler
7
8 def TotalDeep(starting_dir):
9     """Returns (no_of_files, total) where no_of_files is the
10    number of files founds in the directory structure starting
11    at starting_dir, and total is the sum of all the numbers
12    found in those files.
13    """
14    total = no_of_files = 0
15    for this_dir, dir_names,
16        file_names) in os.walk(starting_dir):
17        for this_file in file_names:
18            path_name = os.path.join(this_dir, this_file)
19            try:
20                total += totaler.TotalFile(path_name)
21                no_of_files += 1
22            except OSError as info:
23                print(path_name, info)
24    return (no_of_files, total)
25
26 def main():
27     stats = TotalDeep("..")
28     print(f"That's {stats[0]} files, totaling to {stats[1]}.")
29
30 if __name__ == "__main__":
31     main()
32
```

```
$ lab14_2.py
That's 48 files, totaling to 43391252952.
$ cd ..
$ Lab14_Packages/lab14_2.py 2> /dev/null
That's 698 files, totaling to 58781476069420.59.
$
```

Answers for Lab 14_3)

1. Predict the output:

```
>>> xlist = [1, 2, ['a', 'b']]
>>> xlist_ref = xlist
>>> xlist_copy = xlist[:]
>>> import copy
>>> xlist_deepcopy = copy.deepcopy(xlist)
>>> xlist[0] = '***'
>>> xlist[2][0] = '***'
>>> print xlist
['***', 2, ['***', 'b']]
>>> print xlist_ref
['***', 2, ['***', 'b']]
>>> print xlist_copy
[1, 2, ['***', 'b']]
>>> print xlist_deepcopy
[1, 2, ['a', 'b']]
>>>
```

2. More predictions:

```
>>> xdict = {'a':[1, 2, 3], 'b':[4, 5, 6]}
>>> xdict_ref = xdict
>>> xdict_copy = xdict.copy()
>>> import copy
>>> xdict_deepcopy = copy.deepcopy(xdict)
>>> xdict['a'] = 'ok'
>>> xdict['b'][0] = 888
>>> xdict['c'] = 'watermelon'
>>> print xdict
{'a': 'ok', 'c': 'watermelon', 'b': [888, 5, 6]}
>>> print xdict_ref
{'a': 'ok', 'c': 'watermelon', 'b': [888, 5, 6]}
>>> print xdict_copy
{'a': [1, 2, 3], 'b': [888, 5, 6]}
>>> print xdict_deepcopy
{'a': [1, 2, 3], 'b': [4, 5, 6]}
>>>
```

3. Functional Programming: What are the results?

```
>>> [str(x) for x in range(3)]  
['0', '1', '2']  
  
>>> [[0] for x in range(3)]  
[[0], [0], [0]]  
  
>>> map(lambda x:x**2, range(5))  
[0, 1, 4, 9, 16]  
  
>>> [x for x in range(10) if not x % 2]  
[0, 2, 4, 6, 8]
```

©Marilyn Davis, 2007-2020

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 15 Dynamic Code

- Dynamic Code Generation
- Modules:
 - subprocess(Optional)
 - cProfile(Optional)

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

dynamic.py

```
1 #!/usr/bin/env python3
2 """Demonstrates the exec and eval builtin functions, used
3 for dynamic code generation.
4 """
5 import sys
6
7 ATTRIBUTES = ("name", "zip", "phone", "SSN")
8
9 def GetAttributes(attributes):
10     for each in attributes:
11         answer = input(f"{each} please: ")
12         exec(f"{each} = '{answer}'", globals())
13
14 def PrintAttributes(attributes):
15     for each in attributes:
16         print(f"{each} = {eval(each)}")
17
18 def main():
19     if len(sys.argv) > 1:
20         attributes = sys.argv[1:]
21     else:
22         attributes = ATTRIBUTES
23     GetAttributes(attributes)
24     PrintAttributes(attributes)
25
26 main()
```

```
$ dynamic.py name money_in_pocket
name please: Linda
money_in_pocket please: $1.25
name = Linda
money_in_pocket = $1.25
$ dynamic.py
name please: Marilyn
zip please: 94043
phone please: 650 814-4435
SSN please: XXX-XX-XXXX
name = Marilyn
zip = 94043
phone = 650 814-4435
SSN = XXX-XX-XXXX
$
```

dynamic2.py

```
1 #!/usr/bin/env python3
2 """
3 Demonstrates the setattr and getattr functions for dynamic
4 code generation, which is preferred to exec and eval.
5
6 The first argument to setattr and getattr is the namespace
7 where you expect the variable to land. sys.modules is
8 helpful here if you want it in the current namespace.
9 """
10 import sys
11
12 ATTRIBUTES = ("name", "zip", "phone", "SSN")
13
14 def GetAttributes(attributes):
15     for each in attributes:
16         answer = input(f"{each} please: ")
17         setattr(sys.modules[__name__], each, answer)
18
19 def PrintAttributes(attributes):
20     for each in attributes:
21         print(each, '=', getattr(sys.modules[__name__], each))
22
23 def main():
24     """Same main() as dynamic.py.
25     """
26     if len(sys.argv) > 1:
27         attributes = sys.argv[1:]
28     else:
29         attributes = ATTRIBUTES
30     GetAttributes(attributes)
31     PrintAttributes(attributes)
32
33 if __name__ == "__main__":
34     main()
```

Same output as dynamic.py.

piper.py

```
1 #!/usr/bin/env python3
2 """piper.py (Optional) -- demonstrates running a shell-level
3 command. Stdout is collected and piped into a file object
4 which can be read as if it was an open file.
5 """
6 import os, sys
7 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
8
9 import Lab14_Packages.apple.banana.total_text as total_text
10 import subprocess
11
12 def Total_ps():
13     """Returns the sum of all the numbers in a list
14     of the processes running."""
15
16     with subprocess.Popen(("ps", "-ef"),
17                          stdout=subprocess.PIPE) as popen_obj:
18         # stdout is delivered as bytes. You need to make
19         # a unicode str:
20         output = str(popen_obj.stdout.read(), encoding="utf-8")
21     return total_text.TotalText(output)
22
23
24 if __name__ == "__main__":
25     print("Your lucky number:", Total_ps())
26
```

```
$ piper.py
Your lucky number: 1055528.0
$
```

prof.py

```
1 #!/usr/bin/env python3
2 """prof.py (Optional) Demonstrates the profiler which reports
3 info about the time it takes to run functions."""
4 import cProfile
5
6 def TryWay(data, index):
7     """Uses try/except to return data or None.
8     """
9     try:
10         return data[index]
11     except IndexError:
12         return None
13
14 def TestWay(data, index):
15     """Tests the index to return None if there is no
16     data for the index.
17     """
18     if index < -len(data) or index > len(data) - 1:
19         return None
20     return data[index]
21
22 def TestWay2(data, index):
23     """Checks the i (index) against the len(data) to
24     determine if there is no data.
25     """
26     data_len = len(data)
27     if index < -data_len or index > data_len - 1:
28         return None
29     return data[index]
30
31 def TestPercentWrong(percent_wrong, trials=10000):
32     """Tests the three functions with the a certain
33     percent of failures. """
34     print(f"Testing {percent_wrong:.2%} wrong.")
35     percent_wrong *= 100
36     data = list(range(100))
37     good_count = bad_count = 0
38     for trial in range(trials):
39         for index in range(100):
40             if index < percent_wrong:
41                 index = -index
42                 bad_count += 1
43             else:
44                 good_count += 1
45             TryWay(data, index)
46             TestWay(data, index)
```

```

47             TestWay2(data, index)
48     assert bad_count/(bad_count + good_count) == percent_wrong/100
49
50 def main():
51     for percent in (.10, .01):
52         cProfile.run(f"TestPercentWrong({percent})")
53
54 if __name__ == "__main__":
55     main()

```

```
$ prof.py
Testing 10.00% wrong.
6000005 function calls in 1.897 seconds
```

Ordered by: standard name

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	1.897	1.897	<string>:1(<module>)
1000000	0.479	0.000	0.628	0.000	prof.py:14(TestWay)
1000000	0.363	0.000	0.437	0.000	prof.py:22(TestWay2)
1	0.697	0.697	1.897	1.897	prof.py:31(TestPercentWrong)
1000000	0.136	0.000	0.136	0.000	prof.py:6(TryWay)
1	0.000	0.000	1.897	1.897	built-in method builtins.exec
3000000	0.224	0.000	0.224	0.000	built-in method builtins.len
1	0.000	0.000	0.000	0.000	built-in method builtins.print
1	0.000	0.000	0.000	0.000	method 'disable' of '_lsprof.Profiler' objects

```
Testing 1.00% wrong.
6000005 function calls in 1.884 seconds
```

Ordered by: standard name

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	1.884	1.884	<string>:1(<module>)
1000000	0.475	0.000	0.623	0.000	prof.py:14(TestWay)
1000000	0.361	0.000	0.435	0.000	prof.py:22(TestWay2)
1	0.693	0.693	1.884	1.884	prof.py:31(TestPercentWrong)
1000000	0.132	0.000	0.132	0.000	prof.py:6(TryWay)
1	0.000	0.000	1.884	1.884	built-in method builtins.exec
3000000	0.223	0.000	0.223	0.000	built-in method builtins.len
1	0.000	0.000	0.000	0.000	built-in method builtins.print
1	0.000	0.000	0.000	0.000	method 'disable' of '_lsprof.Profiler' objects

\$

soccer_team.py

```
1 #!/usr/bin/env python3
2 """Processing team data using exec and eval."""
3 import sys
4 def NotifyForwards():
5     return "Go for the goal!"
6 def NotifyDefenders():
7     return "Block that kick!"
8 def NotifyMidfielders():
9     return "Get that ball!"
10 def NotifyGoalies():
11     return "Guard the goal!"
12
13 def ProcessTeam(stream):
14     position_strs = []
15     for line in stream:
16         line = line.strip()
17         if not line:
18             continue
19         if line.endswith(':'):
20             position = line[:-1]
21             exec(f"{position} = []", globals())
22             position_strs += [position]
23             continue
24         details = line.split(' ', 1)
25         exec(f'{position} += [{details}]')
26         exec(f'print("Yeh {details[1]} #{details[0]} ' + Notify{position}())')
27     return stream.name, position_strs
28
29 def PrintTeam(team_name, position_strs):
30     print(f"\n{team_name}:")
31     for each in position_strs:
32         print(f"  {each}:")
33         for player in sorted(eval(each)):
34             print("    " + ":".join(player))
35
36 def main(team_name = "Bees"):
37     with open(team_name) as file_obj:
38         team_name, position_strs = ProcessTeam(file_obj)
39     PrintTeam(team_name, position_strs)
40     print(f"\nThe {team_name} data file:")
41     with open(team_name) as file_obj:
42         sys.stdout.write(file_obj.read())
43
44 main()
```

```
$ soccer_team.py
```

Yeh Bruce Penge #7 Go for the goal!
Yeh Maureen Mezzabo #1 Go for the goal!
Yeh Samantha Smith #8 Go for the goal!
Yeh Juvenal Ramirez #6 Go for the goal!
Yeh Xavier Perra #4 Get that ball!
Yeh Laura Dot #2 Get that ball!
Yeh Malcolm Diamond #5 Get that ball!
Yeh Mary Bart #9 Get that ball!
Yeh Linda Jarvis #3 Block that kick!
Yeh Jose Acosta #11 Guard the goal!
Yeh Tracy Lowe #10 Guard the goal!

Bees:

Forwards:

1: Maureen Mezzabo
6: Juvenal Ramirez
7: Bruce Penge
8: Samantha Smith

Midfielders:

2: Laura Dot
4: Xavier Perra
5: Malcolm Diamond
9: Mary Bart

Defenders:

3: Linda Jarvis

Goalies:

10: Tracy Lowe
11: Jose Acosta

The Bees data file:

Forwards:

7 Bruce Penge
1 Maureen Mezzabo
8 Samantha Smith
6 Juvenal Ramirez

Midfielders:

4 Xavier Perra
2 Laura Dot
5 Malcolm Diamond
9 Mary Bart

Defenders:

3 Linda Jarvis

Goalies:

11 Jose Acosta
10 Tracy Lowe
\$

©Marilyn Davis, 2007-2020

Lab 15 – Exercises:



Optional

1. Find Lab15_Dynamic_Code/soccer_team.py, also printed above. Note that the ProcessTeam and PrintTeam functions have calls to eval and exec. Replace them with calls to setattr and getattr, if you can. Don't spend too much time on this.
The data are in Lab15_Dynamic_Code/Bees.
2. (Optional) Find Lab15_Dynamic_Code/sequences.py. This file contains 4 functions with 4 different methods of accessing the elements in a sequence. Profile these functions to convince yourself that the *Pythonic* way is the fastest way, besides being the easiest to read, write, and get correct.
3. (Optional) On windows the call:

```
c:\windows\system32\ipconfig.exe /all
```

or on OS X and unix:

```
ifconfig
```

gives a lot of info about your networks. Read this into a program that prints out your IP address.

On unix and OS X, it is called `inet addr` in the `ifconfig` report. On windows, it is called `IPv4 Address`

©Marilyn Davis, 2007-2020

lab15_1.py

```
1 #!/usr/bin/env python3
2 """soccer_team.py with setattr and setattr."""
3
4 import sys
5 from soccer_team import NotifyForwards, NotifyDefenders, NotifyMid
6 fielders, NotifyGoalies
7
8 this_module = sys.modules[__name__]
9
10 def ProcessTeam(stream):
11     position_strs = []
12     for line in stream:
13         line = line.strip()
14         if not line:
15             continue
16         if line.endswith(":"):
17             position = line[:-1]
18             setattr(this_module, position, [])
19             position_strs += [position]
20             continue
21         details = line.split(" ", 1)
22         setattr(this_module, position,
23                 getattr(this_module, position) + [details])
24         print(f"Yeh {details[1]} #{details[0]} \"\
25             + getattr(this_module, f"Notify{position}")())
26     return stream.name, position_strs
27
28 def PrintTeam(team_name, position_strs):
29     print(team_name + ":")
30     for each in position_strs:
31         print(" " + each + ":")
32         for player in sorted(eval(each)):
33             print(" " " + ":".join(player))
34
35 def main(team_name = "Bees"):
36     with open(team_name) as file_obj:
37         team_name, position_strs = ProcessTeam(file_obj)
38         PrintTeam(team_name, position_strs)
39         print(f"\nThe {team_name} data file:")
40         with open(team_name) as file_obj:
41             sys.stdout.write(file_obj.read())
42
43 if __name__ == "__main__":
44     main()
```

Same output as soccer_team.py

lab15_2.py

```
1 #!/usr/bin/env python3
2 """lab15_2.py  (Optional)
3
4 Profiles four methods of accessing all the elements in
5 a sequence.
6 """
7
8 def DoLens(sequence):
9     i = 0
10    while i < len(sequence):
11        sequence[i]
12        i += 1
13
14 def DoWhile(sequence):
15    i = 0
16    sequence_len = len(sequence)
17    while i < sequence_len:
18        sequence[i]
19        i += 1
20
21 def DoRange(sequence):
22    for i in range(len(sequence)):
23        sequence[i]
24
25 def DoIterator(sequence):
26    for element in sequence:
27        element
28
29 def Profile(times=1):
30     planets = ("Mercury", "Venus", "Earth", "Mars",
31                "Jupiter", "Saturn", "Uranus", "Neptune")
32     for time in range(times):
33         DoLens(planets)
34         DoWhile(planets)
35         DoRange(planets)
36         DoIterator(planets)
37
38 def main():
39     import cProfile
40     cProfile.run("Profile(5000)")
41
42 main()
43
```

```
$ lab15_2.py
    75004 function calls in 0.029 seconds
    [rest deleted]
```

Ordered by: standard name

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	0.191	0.191	<string>:1(<module>)
5000	0.015	0.000	0.019	0.000	lab15_2.py:10(DoWhile)
5000	0.018	0.000	0.029	0.000	lab15_2.py:17(DoRange)
5000	0.009	0.000	0.009	0.000	lab15_2.py:21(DoIterator)
1	0.025	0.025	0.191	0.191	lab15_2.py:25(Profile)
5000	0.063	0.000	0.108	0.000	lab15_2.py:4(DoLens)
55000	0.053	0.000	0.053	0.000	{len}
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler'}
5001	0.007	0.000	0.007	0.000	{range}

lab15_3.py

```
1 #!/usr/bin/env python3
2 """lab15_3.py (Optional)
3 Finds and reports the OS and IP address of this machine.
4 """
5 import subprocess
6 import sys
7
8 def GetLinuxIP():
9     """Returns the IP address for Linux machines."""
10    for line in GetOutput("inet addr:").split('\n'):
11        address_at = line.find("inet addr:")
12        if address_at == -1:
13            continue
14        return line[address_at + 10:].split()[0]
15
16 def GetOS_X_IP():
17     """Returns the IP address for Apple machines."""
18     output = GetOutput("/sbin/ifconfig")
19     inet_at = output.rfind("inet ") + 5
20     inet_end = output.find(" ", inet_at)
21     return output[inet_at:inet_end]
22
23 def GetOutput(command):
24     with subprocess.Popen(command, stdout=subprocess.PIPE) as pope
24 n_obj:
25         # bytes are delivered - converting to str
26         return str(popen_obj.stdout.read())
27
28
```

```
29
30
31
32 def GetWindowsIP():
33     """Returns the IP address for Windows machines."""
34
35     for line in GetOutput(("c:\\windows\\system32\\ipconfig.exe",
36                           "/all")).split('\\n'):
37         address_at = line.find("IPv4 Address")
38         if address_at == -1:
39             continue
40         ip_address = line.split()[-1]
41         if ip_address[-1] == ')':
42             ip_address = ip_address[:ip_address.index(')')]
43     return ip_address
44
45 def main():
46     this_os = sys.platform
47     print("Platform is", this_os)
48     if this_os in ("darwin",):
49         print(GetOS_X_IP())
50     elif this_os.find("win") > -1:
51         print(GetWindowsIP())
52     elif this_os.find("linux") > -1:
53         print(GetLinuxIP())
54     else:
55         print("Unknown OS")
56
57 if __name__ == "__main__":
58     main()
```

```
$ lab15_3.py
Platform is win32
192.168.1.108
```

```
$ lab15_3.py
Platform is linux2
192.168.1.109
```

```
$ lab15_3.py
Platform is darwin
192.168.0.16
```

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 16 Fancy Facilities

- Function protocols: variable number of keyword arguments
- Formatted strs using a dictionary
- Unpacking dictionaries
- locals() dictionary
- Generators
- iter and next
- Decorators (Optional)

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

** in a function definition

```
def Fn(**arg_dict):
```

allows any keyword str arguments in a call to
the function:

```
Fn(color= "red", length = 3)
```

then, inside the function, automatically, you are given:

```
arg_dict == {"color": "red", "length": 3}
```

~0%

** in a function call. If we have:

```
some_dict = {"grapes":22, "raisens":31}
```

this function call:

```
Fn(**some_dict)
```

is interpreted as:

```
Fn(grapes=22, raisens=31)
```

new_do_breakfast.py

```
1 #!/usr/bin/env python3
2 """ Remember this lab solution:
3 ---
4 def DoBreakfast(meat="bacon", eggs="over easy",
5                 potatos="hash browns", toast="white",
6                 beverage="coffee"):
7     print(f"Here is your {meat} and {eggs} eggs with {potatos}"
8           f" and {toast} toast.",
9           f"Can I bring you more {beverage}?", sep='\n')
10
11 def main():
12     DoBreakfast()
13     DoBreakfast("ham", "basted", "cottage cheese",
14                 "cinnamon", "orange juice")
15     DoBreakfast("sausage", beverage="chai", toast="wheat")
16 ----
17 Now we can do this:
18 """
19
20 def DoBreakfast(**substitutions):
21     order = {"meat": "bacon", "eggs": "over easy",
22               "potatos": "hash browns",
23               "toast": "white", "beverage": "coffee"}
24     # updating values in order from substitutions
25     order.update(substitutions)
26
27     print(f'Here is your {order["meat"]} and {order["eggs"]} with
27 {order["potatos"]} and {order["toast"]} toast.')
28     print(f'Can I bring you more {order["beverage"]}?')
29
30     print("str.format: Can I bring you more {beverage}?.format(**"
30 order))
31
32     print("%%-style of string formatting: Can I bring you more %(b
32 everage)s?" % order)
33
34 def main():
35     print("Call: Do Breakfast()")
36     DoBreakfast()
37     print('\nCall: DoBreakfast(meat="sausage", toast="wheat", beve
37 rage="chai")')
38     DoBreakfast(meat="sausage", toast="wheat", beverage="chai")
39
40 if __name__ == "__main__":
41     main()
```

```
$ new_do_breakfast.py
```

```
Call: Do Breakfast()
Here is your bacon and over easy with hash browns and white toast.
Can I bring you more coffee?
str.format: Can I bring you more coffee?
%-style of string formatting: Can I bring you more coffee?
```

```
Call: DoBreakfast(meat="sausage", toast="wheat", beverage="chai")
Here is your sausage and over easy with hash browns and wheat toast.
Can I bring you more chai?
str.format: Can I bring you more chai?
%-style of string formatting: Can I bring you more chai?
$
```

©Marilyn Davis, 2007-2020

locals.py

```
1 #!/usr/bin/env python3
2 """Demonstrates the locals() dictionary.
3 """
4
5 def ReportAnimals(insect, bird, **more):
6     cat = "Persian"
7     fish = "platies"
8     PrintDict("locals()", locals())
9     locals().update(more)
10    print("After update:")
11    PrintDict("locals()", locals())
12    print("Format tricks:")
13    print("dog={dog}, bird={bird}, cat={cat}".format(**locals()))
14    print("dog=%(dog)s, bird=%(bird)s, cat=%(cat)s" % locals())
15    # dog is in the locals but f-string cannot find it
16    # print(f"dog={dog}, bird={bird}, cat={cat}")
17
18 def PrintDict(message, the_dict):
19     print(message)
20     for key in sorted(the_dict):
21         if type(the_dict[key]) == dict:
22             PrintDict(f"{key} is internal dict in {message}:",
23                       the_dict[key])
24         print()
25     else:
26         print(f"{key:>20}:{the_dict[key]}")
27
28 def main():
29     ReportAnimals("moth", "robin", dog="Collie", horse="Arabian")
30
31 if __name__ == "__main__":
32     main()
```

```
$ locals.py
locals()
    bird:robin
    cat:Persian
    fish:platies
    insect:moth
more is internal dict in locals():
    dog:Collie
    horse:Arabian

After update:
locals()
    bird:robin
    cat:Persian
    dog:Collie
    fish:platies
    horse:Arabian
    insect:moth
more is internal dict in locals():
    dog:Collie
    horse:Arabian

Format tricks:
dog=Collie, bird=robin, cat=Persian
dog=Collie, bird=robin, cat=Persian
$
```

©Marilyn Davis, 2007-2020

generator.py

```
1 #!/usr/bin/env python3
2 """Function to generate random numbers without repetition.
3 Demonstrates generators and the "yield" keyword."""
4 import random
5
6 def Harvest():
7     yield "potatos"
8     yield "corn"
9     yield "peas"
10
11 def DemoHarvest():
12     for crop in Harvest():
13         print(crop, end=' ')
14     print()
15
16 def Unique(bot, over_top):
17     """Generator to deliver randomly chosen values from bot
18     to over_top - 1, delivering each value once only."""
19     answers = list(range(bot, over_top))
20     random.shuffle(answers)
21     for each in answers:
22         yield each
23
24 def ListUnique(bot, over_top):
25     """Returns a list of the generated numbers
26     """
27     gen = Unique(bot, over_top)
28     while True:
29         try:
30             print(next(gen), end=' ')
31         except StopIteration:
32             return
33
34 def ListUnique(bot, over_top):
35     """Same action! """
36     for number in Unique(bot, over_top):
37         print(number, end=' ')
38     print()
39 if __name__ == "__main__":
40     DemoHarvest()
41     print("ListUnique(10, 21) --> ", end=' ')
42     ListUnique(10, 21)
```

```
$ generator.py
potatos corn peas
ListUnique(10, 21) -->  16 19 10 12 17 18 15 20 11 13 14
$
```

next.py

```
1 #!/usr/bin/env python3
2 """The builtin next() function is handy to scrape one value
3 off an iterator.
4 """
5 import generator
6
7 veggie_generator = generator.Harvest()
8
9 print(next(veggie_generator))
10
11 for veggie in veggie_generator:
12     print(veggie)
13
14 toy_list = ["yo-yo", "ball", "doll"]
15 toy_generator = iter(toy_list)
16 print(next(toy_generator))
17 for toy in toy_generator:
18     print(toy)
19
20 print(toy_list[0])
21 print('\n'.join(toy for toy in toy_list[1:]))
22
23 print(next(toy_list))
24
```

```
$ next.py
potatos
corn
peas
yo-yo
ball
doll
yo-yo
ball
doll
Traceback (most recent call last):
  File "./next.py", line 23, in <module>
    print(next(toy_list))
TypeError: 'list' object is not an iterator
$
```

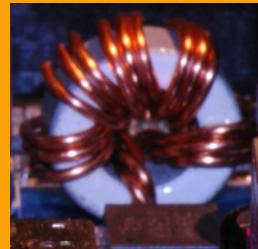
decorator.py

```
1 #!/usr/bin/env python3
2 """Optional: A decorator is a function for wrapping another
3 function, or many other functions. Here we are timestamping
4 the function calls."""
5 import time
6
7 def TimeDecorator(Func):
8     """Decorator function for reporting when the function
9     was called."""
10    def WrappedFunction(*args, **kw_args):
11        print(f"It's {time.ctime()}, time to {Func.__name__}:")
12        return Func(*args, **kw_args)
13    return WrappedFunction
14
15 def DoBreakfast(meat="bacon", eggs="scrambled"):
16     print(f"Have some {meat} and {eggs} eggs. Enjoy!")
17
18 DoBreakfast = TimeDecorator(DoBreakfast)
19
20 @TimeDecorator # syntax available in 2.5
21 def DoLunch(**substitutions):
22     menu = {"meat": "ham", "cheese": "american", "bread": "white"}
23     menu.update(substitutions)
24     print(("Here's your %(meat)s and %(cheese)s"
25           " on %(bread)s bread." % menu))
26
27 @TimeDecorator
28 def DoTea():
29     print("Tea time!")
30
31 @TimeDecorator
32 def DoDinner(menu):
33     print(f"{menu.title()} for dinner tonight.")
34
35 def main():
36     DoBreakfast(meat="sausage", eggs="basted")
37     time.sleep(1)
38     DoLunch(cheese="swiss", bread="rye")
39     time.sleep(1)
40     DoTea()
41     time.sleep(1)
42     DoDinner("roast beef")
43
44 if __name__ == "__main__":
45     main()
```

```
$ decorator.py
```

```
It's Tue Mar  4 12:26:11 2014, time to DoBreakfast:  
Have some sausage and basted eggs. Enjoy!  
It's Tue Mar  4 12:26:12 2014, time to DoLunch:  
Here's your ham and swiss on rye bread.  
It's Tue Mar  4 12:26:13 2014, time to DoTea:  
Tea time!  
It's Tue Mar  4 12:26:14 2014, time to DoDinner:  
Roast Beef for dinner tonight.  
$
```

Lab 16 – Exercises:



1. You have this dictionary of data:

```
event_d = {"what": "party", "date": "Oct 31", "theme": "Halloween"}
```

Write a generator function that receives a dictionary as an argument, and yields a three strings where one string is for a calendar:

"Oct 31:Halloween party"

and one is for an invitation:

"Come to a Halloween party on Oct 31."

and the last is for a banner:

"Halloween Party"

2. Collect labs.zip from the our online class site.

Find Labs/Lab09_Functional_Programming/lab09_4.py so that you can use the GetCards() function to make a generator-based function to deal cards.

Test it in a for loop.

3. (Optional) Extra practice and very interesting: The random module provides a `sample` function. The help facility starts:

```
sample(self, population, k) method of random.Random instance
    Chooses k unique random elements from a population sequence.
```

In this documentation, the `self` argument is not helpful. Pretend `self` is not in the documentation and the rest makes sense. We'll learn what the `self` when we learn about OOP.

How would you call this to deliver a list of 6 numbers between 1 and 52, for a lotto pick?

Make a Lotto function that returns the result of this call.

Make a decorator that logs the time and the output of the function it decorates. Decorate your Lotto function so that each call is logged, complete with the numbers generated. My log entries look like:

```
Wed Mar 28 12:39:58 2007 -> 41, 26, 7, 16, 21, 5
```

Notes:

- When you write a decorator function, you have no idea what sort of functions will be called with your decorator.
 - A *log* is a file that you continually append with new info.
 - After decoration with a logging decorator, a call to `Lotto()` should look the same to the caller.
4. Use the `next()` builtin function on open file objects to make a generator function that walks a directory structure and delivers tuples to the caller. Each tuple is (`file_path, first_line`).
5. (Optional) Use a generator function and dynamic coding to test the dice.py program:

Collect `labs.zip` from our online class site. Find: `Labs/Lab05_Functions/dice.py`

Hints:

- (a) Make a generator function that emulates the call to `random.randrange(1, 7)` but it returns determined (not random) values, in order so that all doubles are rolled.
- (b) Read `dice.py` as a text file, and change the text so that, when you `exec` it, it calls your generator instead of `random.randrange(1, 7)`.
- (c) The results are:
1 and 1= 2 Snake eyes!
2 and 2= 4 Little joe!
3 and 3= 6 Hard six!
4 and 4= 8 Hard eight!
5 and 5= 10 Fever!
6 and 6= 12 Box cars!

©Marilyn Davis, 2007-2020

lab16_1.py

```
1 #!/usr/bin/env python3
2 """
3 lab16_1.py
4 Write a function that receives a dictionary as an argument,
5 and returns a tuple of two formatted strings.
6 """
7
8 def FormatEvent(event_d):
9     """Returns three strings for the event:
10     one formatted for a calendar,
11     one formatted for an invitation,
12     one for a banner.
13     """
14     calendar = "{date}:{theme} {what}".format(**event_d)
15     invitation = "Come to a %(theme)s %(what)s on %(date)s!" % eve
15 nt_d
16     banner = f'{event_d["theme"]}-{event_d["what"]}'.title()
17     yield calendar
18     yield invitation
19     yield banner
20
21 def main():
22     event_d = {"what": "party", "date": "Oct 31",
23                "theme": "Halloween"}
24     for each in FormatEvent(event_d):
25         print(each)
26
27 if __name__ == "__main__":
28     main()
29
```

```
$ lab16_1.py
Oct 31:Halloween party
Come to a Halloween party on Oct 31!
Halloween Party
$
```

lab16_2.py

```
1 #!/usr/bin/env python3
2 """
3 A generator-based function to deal cards.
4 """
5 import os, random, sys
6 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
7 import Lab09_Functional_Programming.lab09_5 as cards
8
9 def DealCard():
10     """Generator to yield one card at a time from a deck."""
11     deck = cards.GetCards()
12     random.shuffle(deck)
13     for card in deck:
14         yield card
15
16 def main():
17     for i, card in enumerate(DealCard()):
18         print(card, end=' ')
19         if i % 5 == 4:
20             print()
21
22 if __name__ == "__main__":
23     main()
```

```
$ lab16_2.py
5 of Hearts Queen of Clubs Queen of Spades 7 of Diamonds 3 of Spades
6 of Spades Joker 8 of Spades 4 of Hearts 4 of Diamonds
King of Diamonds 7 of Clubs 2 of Hearts Ace of Hearts 5 of Clubs
10 of Hearts 2 of Diamonds 3 of Clubs Jack of Hearts 9 of Hearts
10 of Clubs 8 of Hearts 7 of Spades Queen of Hearts King of Clubs
Queen of Diamonds 6 of Diamonds 5 of Spades Ace of Clubs 4 of Clubs
9 of Clubs 3 of Diamonds 4 of Spades Jack of Spades Jack of Diamonds
5 of Diamonds King of Spades Jack of Clubs 8 of Diamonds 6 of Hearts
9 of Spades 6 of Clubs Ace of Spades 10 of Diamonds Ace of Diamonds
10 of Spades 8 of Clubs 7 of Hearts King of Hearts 2 of Spades
9 of Diamonds 2 of Clubs 3 of Hearts Joker
$
```

lab16_3.py

```
1 #!/usr/bin/env python3
2 """lab16_3.py (Optional) A logging lotto facility."""
3 import time
4 import random
5
6 LOG_FILE = "lotto.log"
7
8 def LogIt(Func):
9     """Decorator function for logging output from the Func."""
10    def LoggedFunction(*t_args, **kw_args):
11        with open(LOG_FILE, "a") as open_log:
12            got = Func(*t_args, **kw_args)
13            open_log.write(f"{time.ctime()} -> {Func.__name__}:{got}\n")
14        return got
15    return LoggedFunction
16
17 @LogIt
18 def Lotto():
19     return random.sample(range(1, 53), 6)
20
21 def PrintLotto(the_pick):
22     print(", ".join([str(x) for x in the_pick]))
23
24 def main():
25     PrintLotto(Lotto())
26     PrintLotto(Lotto())
27
28 if __name__ == "__main__":
29     main()
30
```

```
$ lab16_3.py
35, 10, 12, 47, 24, 48
34, 12, 22, 27, 38, 46
$ cat lotto.log
Mon Mar  1 11:41:42 2010 -> Lotto:[42, 27, 38, 35, 17, 3]
Mon Mar  1 11:41:42 2010 -> Lotto:[13, 27, 51, 49, 32, 30]
Mon Mar  1 11:41:52 2010 -> Lotto:[35, 10, 12, 47, 24, 48]
Mon Mar  1 11:41:52 2010 -> Lotto:[34, 12, 22, 27, 38, 46]
$
```

lab16_4.py

```
1 #!/usr/bin/env python3
2 """
3 next() is used to make a generator function that walks
4 a directory structure and delivers tuples to the caller.
5 Delivered is (file_path, first_line).
6 """
7 import os, sys
8
9 def DriveFirstLines():
10     if len(sys.argv) == 1:
11         start_dirs = ['.']
12     else:
13         start_dirs = sys.argv[1:]
14     for start_dir in start_dirs:
15         for path, first_line in FirstLines(start_dir):
16             if path:
17                 print(path, '\t' + first_line, end=' ')
18
19 def FirstLines(start_dir):
20     for this_dir, dirnames, file_names in os.walk(start_dir):
21         if this_dir == start_dir:
22             dirnames.sort()
23         for file_name in sorted(file_names):
24             whole_path = os.path.join(this_dir, file_name)
25             with open(whole_path) as file_object:
26                 try:
27                     first_line = next(file_object, "Can't read it.
27 ")
28                 except UnicodeDecodeError:
29                     first_line = "unicode error\n"
30             yield (whole_path, first_line)
31
32 def main():
33     DriveFirstLines()
34
35 if __name__ == "__main__":
36     main()
```

```
$ lab16_4.py ...
./.pdbrc      break ../utils/check_outputs.py:266
./file.log      Opening "words.txt" in mode 'a' on 2019-02-22 17:20:31.553191
./words.txt      Writing some words at 2019-02-22 17:20:31.553307.
./Lab01_Output/acknowledgements      \Large \bf Acknowledgements
[many deleted]
```

lab16_5.py

```
1 #!/usr/bin/env python3
2 """
3 Testing dice.py by reading it as a text file, making a little
4 change, and then executing it, via the exec command.
5 """
6 import sys, os
7
8 def GetDiceCodeString():
9     """Read the dice.py file as text and change random.randrange
10        to be next(doubles_generator_object), and return the text.
11    """
12     file_path = os.path.join("../", "Lab05_Libraries", "dice.py")
13     with open(file_path) as open_file:
14         dice_code = open_file.read()
15
16     dice_code = dice_code.replace(
17         "random.randrange(1, 7)",
18         "next(doubles_generator_object)")
19     # The __name__ is "__main__" so the code will run when
20     # exec-ed. Stopping that:
21     end_at = dice_code.find("def main")
22     dice_code = dice_code[:end_at]
23     return dice_code
24
25 def RollDeterminedDice():
26     """Returns a roll of the dice, in order so that doubles
27     are rolled.
28    """
29     for die in range(1, 7):
30         yield die
31         yield die
32
33 def TestRollem():
34     """Tests the Rollem function in dice.py.
35    """
36     global doubles_generator_object
37     # doubles_generator_object needs to be in the global
38     # space for Rollem() to find it.
39
40     dice_code = GetDiceCodeString()
41
42     doubles_generator_object = RollDeterminedDice()
43
44     exec(dice_code + "\nfor roll in range(6): print(Rollem())")
45     """ This worked in python2 but now exec has its own namespace
45 .
```

```
46     for roll in range(6):  
47         print(Rollem())  
48     """  
49  
50 TestRollem()
```

```
$ lab16_5.py  
1 and 1= 2 Snake eyes!  
2 and 2= 4 Little joe!  
3 and 3= 6 Hard six!  
4 and 4= 8 Hard eight!  
5 and 5= 10 Fever!  
6 and 6= 12 Box cars!  
$
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

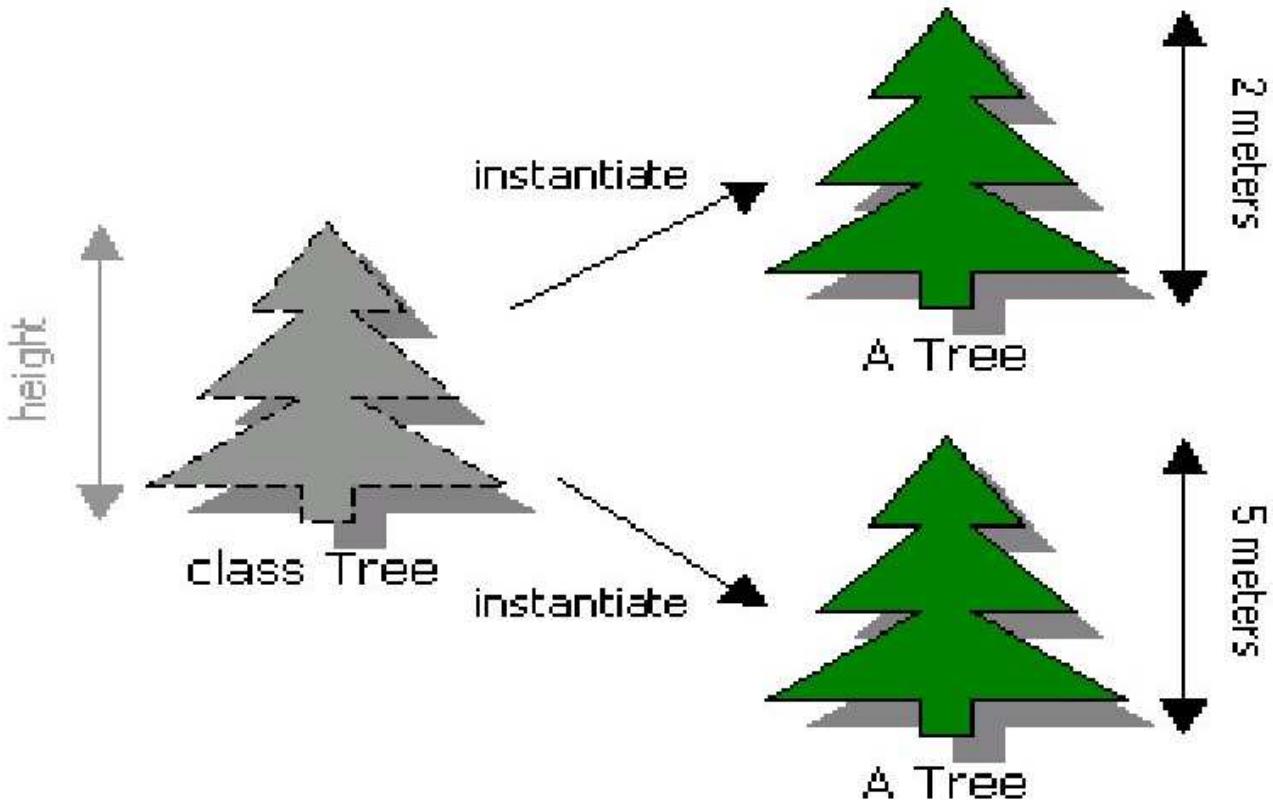
UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 17 OOP

- Classes
- Container class
- Inheritance
- Class attribute

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.



Object Oriented Programming

A **class** is:

a blueprint for a namespace.

An **object** is:

a namespace constructed from the blueprint.

greeter1_def.py

```
1 #!/usr/bin/env python3
2 """Here is a very simple object-oriented python program.
3 Note that we make 3 namespaces: the Greeter class, and two
4 instances of the Greeter class, fred and alma."""
5
6 class Greeter:
7     """The Greeter class makes Greeter objects that can
8     greet you."""
9
10    def Greet(self):           # "self" is always the first
11        print("Hello World")  # argument of every method in
12                           # every class.
13    def main():
14        fred = Greeter()
15        print("fred.Greet():")  # A call: fred.Greet() is
16        fred.Greet()           # interpreted as
17        alma = Greeter()      # Greeter.Greet(fred). So the
18        print("alma.Greet():")  # greeter object labeled
19        alma.Greet()          # "fred" is the namespace we
                           # are relabeling as "self"
20
21    PrintStringValue(fred=fred, alma=alma, Greeter=Greeter)
22
23 def PrintStringValue(**dict_version):
24     for namespace_string in dict_version:
25         print(namespace_string, ':', end=' ')
26         print(dict_version[namespace_string])
27
28 if __name__ == "__main__":
29     main()
30
```

```
$ greeter1_def.py
fred.Greet():
Hello World
alma.Greet():
Hello World
Greeter : <class '__main__.Greeter'>
alma : <__main__.Greeter object at 0x1033c4da0>
fred : <__main__.Greeter object at 0x1033c4d68>
$
```

greeter2_def.py

```
1 #!/usr/bin/env python3
2 """Here we add a name attribute to our class and a method
3 to set a value into the name.
4
5 The value of an attribute belongs solely to the object,
6 the methods belong both to the class and the objects."""
7
8 class Greeter:
9     """The Greeter class makes potentially named Greeter
10    objects that can greet you."""
11
12     def SetName(self, name_in):
13         self.name = name_in
14
15     def Greet(self):
16         try:
17             print(f"Hello World. I'm {self.name}")
18         except AttributeError:
19             print("Hello World.")
20
21 def main():
22     gracy = Greeter()
23     gracy.Greet()
24     gracy.SetName("Gracy")
25     gracy.Greet()
26     george = Greeter()
27     george.SetName("George")
28     george.Greet()
29     gracy.Greet()
30
31 if __name__ == "__main__":
32     main()
```

```
$ greeter2_def.py
Hello World.
Hello World. I'm Gracy
Hello World. I'm George
Hello World. I'm Gracy
$
```

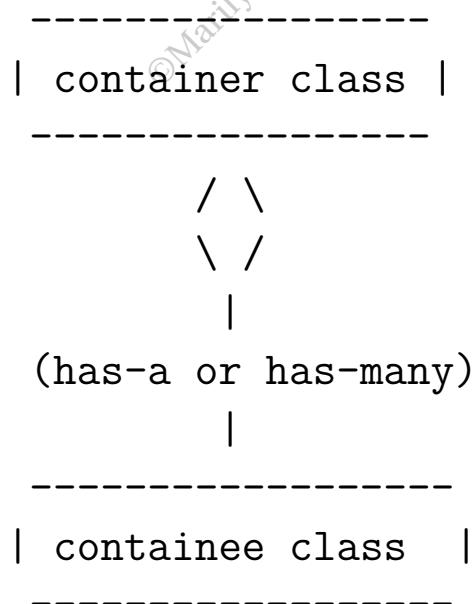
greeter3_def.py

```
1 #!/usr/bin/env python3
2 """Alternatively, we define an __init__ method so that the
3 name is given when the object is created."""
4
5 class Greeter:
6     # This Greeter class needs a name when instantiated.
7
8     def __init__(self, name):
9         # __init__ is Python's constructor method
10        self.name = name
11
12    def Greet(self):
13        print("Hello World. I'm", self.name)
14
15 def main():
16    fred = Greeter('Fred')
17    print("fred.Greet():")
18    fred.Greet()
19
20    x = Greeter()      # error!
21
22 if __name__ == "__main__":
23    main()
24
```

```
$ greeter3_def.py
fred.Greet():
Hello World. I'm Fred
Traceback (most recent call last):
  File "./greeter3_def.py", line 23, in <module>
    main()
  File "./greeter3_def.py", line 20, in main
    x = Greeter()      # error!
TypeError: __init__() missing 1 required positional argument: 'name'
$
```



An important relationship between classes is **containment**, where an object of one class instantiates within it (often many) object(s) of another class. It is diagrammed with a diamond shape touching the container class:



tree_def.py

```
1 #!/usr/bin/env python3
2 """
3 A Tree class, to make tree objects,
4 and a Forest class. The Forest
5 class is a "container" class because
6 it contains the tree objects.
7
8
9
10
11
12
13
14
15
16 """
17 import random
18
19 class Tree:
20     """Instantiate: Tree(20) to make a tree 20 ft tall.
21     """
22     def __init__(self, height):
23         self.height = height
24
25     def Print(self):
26         print(f"tree, {self.height:.1f} feet tall")
27
28 class Forest:
29     """Instantiate: Forest(size="medium")
30     if size == "large", it will have 8 trees.
31     == "medium", it will have 5 trees.
32     == "small", it will have 2 trees.
33     """
34     def __init__(self, size="medium"):
35         if size not in ("small", "medium", "large"):
36             raise ValueError(f"""Instantiate with:
37 Forest([size="medium"]) where size can be "small", "medium",
38 or "large", not "{size}".""")
39         self.size = size
40         self.number_of_trees = 8 if self.size == "large" else 5 if
41         self.size == "medium" else 2
42         # Here comes the implementation of the containment:
43
44         self.trees = [Tree(random.randrange(1, 200))
45                         for count in range(self.number_of_trees)]
```

```
46
47     def Print(self):
48         print(f"{self.size} forest with {self.number_of_trees} tre
49 es:")
50         for tree in self.trees:
51             tree.Print()
52         print()
53
54 def main():
55     for size in "small", "medium", "large":
56         forest = Forest(size)
57         forest.Print()
58     try:
59         forest = Forest("huge")
60     except ValueError as info:
61         print(info)
62 if __name__ == "__main__":
63     main()
```

```
$ tree_def.py
small forest with 2 trees:
tree, 157.0 feet tall
tree, 35.0 feet tall
```

```
medium forest with 5 trees:
tree, 22.0 feet tall
tree, 114.0 feet tall
tree, 12.0 feet tall
tree, 112.0 feet tall
tree, 129.0 feet tall
```

```
large forest with 8 trees:
tree, 102.0 feet tall
tree, 98.0 feet tall
tree, 113.0 feet tall
tree, 49.0 feet tall
tree, 185.0 feet tall
tree, 58.0 feet tall
tree, 130.0 feet tall
tree, 167.0 feet tall
```

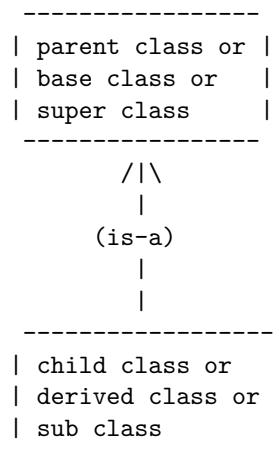
```
Intantiate with:
Forest([size="medium"]) where size can be "small", "medium", or "large", not "huge".
$
```



One of the greatest benefits of OOP is the ability to add functionality and complication to a class and yet leave the class alone, keeping it simple.

It's like eating cake and still having it.

This is **inheritance**, another relationship between classes. It is diagrammed with an arrow pointing to the parent class:



greeter4_def.py

```

1 #!/usr/bin/env python3
2 """Here we implement "inheritance"
3 to leave the original Greeter class
4 intact and add a NamedGreeter class,
5 that has all the functionality of
6 the Greeter class plus some new
7 things.
8
9
10
11
12
13
14 class Greeter:
15
16     def Greet(self):
17         print("Hello World")
18
19 class NamedGreeter(Greeter):
20     # Inherits the methods in the Greeter class, and adds some
21     # of its own.
22
23     def __init__(self, name):
24         self.name = name
25
26     def SayMyName(self):
27         print("I'm", self.name)
28
29 def main():
30     fred = NamedGreeter("Fred")
31     print("fred.Greet():")
32     fred.Greet()
33     fred.SayMyName()
34
35     # code that depends on the Greeter class is unaffected.
36     x = Greeter()
37     print("x.Greet():")
38     x.Greet()
39
40 if __name__ == "__main__":
41     main()

```

```

$ greeter4_def.py
fred.Greet():
Hello World
I'm Fred
x.Greet():
Hello World

```

greeter5_def.py

```
1 #!/usr/bin/env python3
2 """Here, both classes have a Greet method. The lingo is:
3 the NamedGreeter.Greet() method "overrides" the
4 Greeter.Greet() method. When an object of the NamedGreeter
5 class calls Greet(), it accesses and runs NamedGreeter.Greet()
6 while an object of the Greeter class executes Greeter.Greet().
7 """
8 class Greeter:
9
10     def Greet(self):
11         print("Hello World")
12     def Bye(self):
13         print("Bye now.")
14
15 class NamedGreeter(Greeter):
16
17     def __init__(self, name):
18         self.name = name
19     def Greet(self):
20         # NamedGreeter.Greet() calls the Greeter.Greet()
21         # method and then adds some functionality. This
22         # is a common and useful technique.
23         Greeter.Greet(self)
24         print(f"I'm {self.name}")
25
26 def main():
27     fred = NamedGreeter("Fred")
28     print("fred.Greet():")
29     fred.Greet()                      # Output:
30     print("fred.Bye():")              #
31     fred.Bye()                      # $ greeter5_def.py
32     x = Greeter()
33     print("x.Greet():")              # fred.Greet():
34     x.Greet()                       # Hello World
35     print("x.Bye():")               # I'm Fred
36     x.Bye()                         # fred.Bye():
37                               # Bye now.
38 if __name__ == "__main__":
39     main()                           # x.Greet():
40                               # Hello Word
41                               # x.Bye():
42                               # Bye now.
43                               # $
```

greeter6_def.py

```
1 #!/usr/bin/env python3
2 """Here we implement another class, a HipGreeter, and have it
3 further down the inheritance tree."""
4
5 class Greeter:
6     def Greet(self):
7         print("Hello World")
8     def Bye(self):
9         print("Bye now.")
10
11 class NamedGreeter(Greeter):
12     def __init__(self, name):
13         self.name = name
14     def Greet(self):
15         Greeter.Greet(self)
16         print(f"I'm {self.name}")
17
18 class HipGreeter(NamedGreeter):
19     def Greet(self):
20         NamedGreeter.Greet(self)
21         print("Wazzup.")
22
23 def main():
24     rocky = HipGreeter("Rocky")
25     rocky.Greet()
26     rocky.Bye()
27
28 if __name__ == "__main__":
29     main()
```

```
$ greeter6_def.py
Hello World
I'm Rocky
Wazzup.
Bye now.
$
```

Python Namespaces

Label	Assignment	Reference
Unqualified	x = value makes or changes a local x unless declared global	x as in print(x) Look in local __dict__, then enclosing namespace's, globals and then built-ins
Qualified	obj.x = value obj can be a package, module, class, or instantiation of a class ("object" in the oop sense)	obj.x Looks for x in obj. If obj is a class or an instantiation of a class, it looks in the class and in super-classes.

©Marilyn Davis, 2007-2020

soccer_team.py

```

1 #!/usr/bin/env python3
2 """
3 OO Implementation      | Team |
4 of a soccer team:      |-----|
5                                / \
6                                \ / ("has" or "contains")
7                                |
8 -----
9                                | Player |
10                               |-----|
11                               /|\      /|\      /|\      /|\
12                               | | ("is a" or "inherits from")
13                               | |           | |           | |
14 -----      -----      -----      -----
15 | Forward | | Midfielder | | Defender | | Goalie |
16 -----      -----      -----      -----
17 """
18 import sys
19
20 class Player:
21     """Base class for soccer players. A "virtual" class that
22     cannot be directly instantiated.
23     """
24     def __init__(self, line):
25         if self.__class__ == Player:
26             raise TypeError("Player class cannot be directly insta
27 nited.")
27         number, self.name = line.split(' ', 1)
28         self.number = int(number)
29
30     def Cheer(self):
31         """Cheers the player with an encouragement appropriate
32         for the position.
33         """
34         return f" Yeah {self.name}, #{self.number}! {self.encourage
34 ment}"
35 class Forward(Player):
36     encouragement = "Go for the goal!"
37
38 class Defender(Player):
39     encouragement = "Block that kick!"
40
41 class Midfielder(Player):
42     encouragement = "Get that ball!"
43
44

```

```
45 class Goalie(Player):
46     encouragement = "Guard the goal!"
47
48 class Team:
49     """
50     Container class for the players.
51     """
52     def __init__(self, data_file):
53         self.players = []
54         with open(data_file) as data_obj:
55             for line in data_obj:
56                 line = line.strip()
57                 if not line or line[0] == '#':
58                     continue
59                 if line.endswith(':'):
60                     position = line
61                     continue
62                 if position == "Forwards:":
63                     new_player = Forward(line)
64                 elif position == "Midfielders:":
65                     new_player = Midfielder(line)
66                 elif position == "Defenders:":
67                     new_player = Defender(line)
68                 elif position == "Goalies:":
69                     new_player = Goalie(line)
70                 else:
71                     raise ValueError(f"Unknown soccer position: {position}")
72             self.players.append(new_player)
73
74     def Cheer(self):
75         return "\n".join([player.Cheer() for player in sorted(
76             self.players, key=lambda p:p.number)])
77
78 def main():
79     try:
80         data_file = sys.argv[1]
81     except IndexError:
82         data_file = "../Lab15_Dynamic_Code/Bees"
83     team = Team(data_file)
84     print(team.Cheer())
85     Player("0 Can't happen")
86
87 if __name__ == "__main__":
88     main()
89
90
91
```

92
93
94

```
$ soccer_team.py
Yeah Maureen Mezzabo, #1! Go for the goal!
Yeah Laura Dot, #2! Get that ball!
Yeah Linda Jarvis, #3! Block that kick!
Yeah Xavier Perra, #4! Get that ball!
Yeah Malcolm Diamond, #5! Get that ball!
Yeah Juvenal Ramirez, #6! Go for the goal!
Yeah Bruce Penge, #7! Go for the goal!
Yeah Samantha Smith, #8! Go for the goal!
Yeah Mary Bart, #9! Get that ball!
Yeah Tracy Lowe, #10! Guard the goal!
Yeah Jose Acosta, #11! Guard the goal!
Traceback (most recent call last):
  File "./soccer_team.py", line 88, in <module>
    main()
  File "./soccer_team.py", line 85, in main
    Player("0 Can't happen")
  File "./soccer_team.py", line 26, in __init__
    raise TypeError("Player class cannot be directly instantiated.")
TypeError: Player class cannot be directly instantiated.
```

©Marilyn Davis, 2007-2023

Lab 17 – Exercises:



Sometimes when you have an object, you need to know: *from which class was this object instantiated?*

`any_object.__class__`

is a label to the instantiating class. But, sometimes you need the name of class as a string:

`any_object.__class__.__name__`

is the label you need.

1. Implement a Stack class. It should have two methods: one adds things to the top of your stack (usually called push), and the other takes things from the top of your stack, (usually called pop).

This is just wrapping some class definition syntax around a list.

2. Make an Egg class. Each egg has two attributes: shell color and bird type.

Your Egg class should have a Report method that reports, for example:

`blue song thrush egg`

And make a Basket class that has two attributes, the maximum number of eggs it will hold, and the list of eggs that are in it.

You need an AddEgg method for your Basket class. An extra egg is rejected with the message `Only 12 eggs fit in this basket.`, assuming that this basket's maximum number of eggs is 12.

Your Basket class should report the eggs that are in it, in alphabetical order of the bird type.

Here are some bird egg data, also at `Lab17_OOP/bird_eggs.py`, but you are welcome to invent your data:

```
{  
    "hummingbird": "white",  
    "song thrush": "blue",  
    "tawny owl": "beige",  
    "buzzard": "speckled brown",  
    "chicken": "brown",  
    "ostrich": "speckled yellow",  
    "emu": "black"  
}
```

3. Make a class that inherits from the Tree class in `Labs/Lab17_OOP/tree_def.py`: FruitTree. This new class will also have a Print method that will call the Tree class' Print method and then say, *Eat my <fruit type>..* Where the *<fruit type>* is fruit for trees of this class.

But, also make three classes that inherit from your FruitTree class: Apple, Banana and Fig. These three classes only have no methods defined, only the datum for all objects of the class: Apple trees have *apples*, etc. Note: `self.__class__.__name__` is always available and helpful here. My output:

```
$ lab17_3.py
Apple tree, 20.0 feet tall
Eat my apples.
Banana tree, 12.0 feet tall
Eat my bananas.
Fig tree, 8.0 feet tall
Eat my figs.
$
```

4. (Optional) Implement this inheritance tree:

```
Employee
    name

SalariedEmployee --> Inherits from Employee
    Has a yearly salary.

ContractEmployee --> Inherits from Employee
    Has an hourly rate.
```

This code:

```
joe = SalariedEmployee("Joe", 52000)
joe.PrintName()
print(f"here's ${joe.CalculatePay(1):.2f} for you. ")
joe.GiveRaise(2)
joe.PrintName()
print(f"here's ${joe.CalculatePay(2):.2f} for you. ")

susan = ContractEmployee("Susan", 100)
susan.PrintName()
print(f"here's ${susan.CalculatePay(80):.2f} for you. ")
susan.GiveRaise(2)
susan.PrintName()
print(f"here's ${susan.CalculatePay(80):.2f} for you. ")

fred = Employee("Fred", 100)
try:
    fred.CalculatePay(20) # Crash!
except AttributeError:      # No CalculatePay for Employee
    pass
```

Should produce this output:

```
Joe here's $1000.00 for you.
Joe here's $2040.00 for you.
Susan here's $8000.00 for you.
Susan here's $8160.00 for you.
```

lab17_1.py

```
1 #!/usr/bin/env python3
2 """
3 lab17_1.py  Implement a Stack class.  It should
4 have two functions: you add things to the top of your
5 stack (usually called push), and you take things from
6 the top of your stack, (usually called pop)."""
7
8 class Stack:
9     def __init__(self):
10         self.things = []
11
12     def Push(self, thing):
13         self.things += [thing]
14
15     def Pop(self):
16         try:
17             return self.things.pop()
18         except IndexError:
19             return None
20
21     def main():
22         box = Stack()
23         print(box.Pop())
24         box.Push("nickel")
25         box.Push("dime")
26         print(box.Pop())
27         print(box.Pop())
28         print(box.Pop())
29
30 if __name__ == "__main__":
31     main()
```

```
$ lab17_1.py
None
dime
nickel
None
$
```

lab17_2.py

```
1 #!/usr/bin/env python3
2 """
3 Egg and Basket classes.
4 """
5 import random
6 class Egg:
7
8     def __init__(self, bird_type, color):
9         self.bird_type, self.color = bird_type, color
10
11    def Report(self):
12        return f"{self.color} {self.bird_type} egg"
13
14 class Basket:
15
16    def __init__(self, max_eggs=12, **egg_dict):
17        self.eggs = []
18        self.max_eggs = max_eggs
19        self.number_of_eggs = 0
20        for bird_type in egg_dict:
21            self.AddEgg(bird_type, egg_dict[bird_type])
22
23    def AddEgg(self, bird_type, color):
24        if self.max_eggs == self.number_of_eggs:
25            print(f"Only {self.max_eggs} are allowed in this basket.")
26            return
27        self.eggs.append(Egg(bird_type, color))
28        self.number_of_eggs += 1
29
30    def Report(self):
31        report = f"Basket for {self.max_eggs} with {self.number_of_eggs}:\n\t"
32        number_width = len(str(self.max_eggs))
33        report += "\n\t".join(f"\t{i:{number_width}}: {b.Report()}" for (i,b) in
34        enumerate(sorted(self.eggs, key=lambda b:b.bird_type),
35                    start=1))
36
37    def main():
38        egg_data = {"hummingbird": "white",
39                    "song thrush": "blue",
40                    "tawny owl": "beige",
41                    "buzzard": "speckled brown",
```

```
43         "chicken": "brown",
44         "ostrich": "speckled yellow",
45         "emu": "black" }
46
47     basket = Basket(**egg_data)
48     birds = list(egg_data.keys())
49     how_many_left = basket.max_eggs - len(birds)
50     for bird in range(how_many_left + 1):
51         this_bird = random.choice(birds)
52         basket.AddEgg(this_bird, egg_data[this_bird])
53
54     print(basket.Report())
55
56 if __name__ == "__main__":
57     main()
58
```

```
$ lab17_2.py
```

```
Only 12 are allowed in this basket.
```

```
Basket for 12 with 12:
```

```
1: speckled brown buzzard egg
2: speckled brown buzzard egg
3: brown chicken egg
4: brown chicken egg
5: black emu egg
6: white hummingbird egg
7: speckled yellow ostrich egg
8: speckled yellow ostrich egg
9: blue song thrush egg
10: blue song thrush egg
11: blue song thrush egg
12: beige tawny owl egg
```

```
$
```

lab17_3.py

```
1 #!/usr/bin/env python3
2 """
3 lab17_3.py A FruitTree class that inherits from the Tree class in
4 the tree_def.py module. Also, three specific fruit trees classes
5 are defined that inherit from the FruitTree class: Apple, Banana,
6 and Fig."""
7 import tree_def
8
9 class FruitTree(tree_def.Tree):
10     """Instantiate: FruitTree(size_in_feet)
11     """
12     fruit = "forbidden fruit"
13
14     def Print(self):
15         print(self.__class__.__name__, end=' ')
16         tree_def.Tree.Print(self)
17         print(f"Eat my {self.fruit}.")
18
19 class Apple(FruitTree):
20     fruit = "apples"
21
22 class Banana(FruitTree):
23     fruit = "bananas"
24
25 class Fig(FruitTree):
26     fruit = "figs"
27
28 def main():
29     trees = Apple(20), Banana(12), Fig(8)
30     for tree in trees:
31         tree.Print()
32
33 if __name__ == "__main__":
34     main()
35
```

```
$ lab17_3.py
Apple tree, 20.0 feet tall
Eat my apples.
Banana tree, 12.0 feet tall
Eat my bananas.
Fig tree, 8.0 feet tall
Eat my figs.
$
```

lab17_4.py

```
1 #!/usr/bin/env python3
2 """lab17_4.py  Implement this inheritance tree:
3
4     Employee
5         name
6
7     SalariedEmployee    --> Inherits from Employee
8         Has a yearly salary.
9
10    ContractEmployee   --> Inherits from Employee
11        Has an hourly rate
12 """
13
14 import sys
15
16 class Employee:
17     """Employee class, should only be instantiated in
18     a subclass"""
19
20     def __init__(self, name, pay_rate):
21         self.name = name
22         self.pay_rate = float(pay_rate)
23
24     def GiveRaise(self, percent):
25         """percent is the percent raise, where 100 doubles
26         the pay rate."""
27
28         percent /= 100.0
29         self.pay_rate *= 1 + percent
30
31     def PrintName(self):
32         print(self.name, end=' ')
33
34 class SalariedEmployee(Employee):
35     """pay_rate is the yearly salary. A pay period is
36     1 week."""
37
38     def CalculatePay(self, weeks):
39         return self.pay_rate * weeks/52.
40
41 class ContractEmployee(Employee):
42     """pay_rate is hourly pay. A pay period is 1 hour."""
43
44     def CalculatePay(self, hours):
45         return self.pay_rate * hours
46
```

```
47 def main():
48     joe = SalariedEmployee("Joe", 52000)
49     joe.PrintName()
50     print(f"here's ${joe.CalculatePay(1):.2f} for you. ")
51     joe.GiveRaise(2)
52     joe.PrintName()
53     print(f"here's ${joe.CalculatePay(2):.2f} for you. ")
54
55     susan = ContractEmployee("Susan", 100)
56     susan.PrintName()
57     print(f"here's ${susan.CalculatePay(80):.2f} for you. ")
58     susan.GiveRaise(2)
59     susan.PrintName()
60     print(f"here's ${susan.CalculatePay(80):.2f} for you. ")
61
62     fred = Employee("Fred", 100)
63     try:
64         fred.CalculatePay(20) # Crash!
65     except AttributeError: # No CalculatePay for Employee
66         pass
67
68 if __name__ == "__main__":
69     main() # Output given in specification
```

©Marilyn Davis, 2021-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 18 Magic

- Multiple Inheritance
- Useful attributes
- Making Magic with *Special* Methods
- Privacy

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0

greeter7_def.py

```
1 #!/usr/bin/env python3
2 """Here we have a NamedGreeterGuru Class, and a
3 GuruNamedGreeter Class, overriding the Bye()
4 method in the Guru class, demonstrating left-
5 right, depth-first method resolution.
6 """
7 import random
8
9 class Guru:
10     # And here is a class attribute, accessable anywhere
11     # by Guru.sayings
12     sayings = ("There is no solution, for there is no"
13                "\nproblem.                               Marcel Duchamp",
14                "If you could get rid of yourself just once,"
15                "\nThe secret of secrets would open to you."
16                "\n                                         Jalaluddin Rumi",
17                "Nothing is so simple that it cannot be"
18                "\nmisunderstood.           Freeman Teague",
19                "Trying to define yourself is like trying"
20                "\nto bite your own teeth.      Alan Watts")
21
22     def Bye(self):
23         print("Good Bye.  And remember:")
24         self.Pontificate()
25
26     def Pontificate(self):
27         print(random.choice(Guru.sayings))
28
29 class Greeter:
30
31     def Greet(self):
32         print("Hello World")
33
34     def Bye(self):
35         print("Bye now.")
36
37 class NamedGreeter(Greeter):
38
39     def __init__(self, name):
40         self.name = name
41
42     def Greet(self):
43         Greeter.Greet(self)
44         print(f"I'm {self.name}")
45
46
```

```
47 # Multiple Inheritance
48
49 class GuruNamedGreeter(Guru, NamedGreeter):
50     pass # A "mixed-in" class adds no new functionality
51
52 class NamedGreeterGuru(NamedGreeter, Guru):
53     pass
54
55 def main():
56     rocky = GuruNamedGreeter("Rocky")
57     rocky.Greet()
58     rocky.Pontificate()
59     rocky.Bye()
60
61     moose = NamedGreeterGuru("Moose")
62     moose.Greet()
63     moose.Pontificate()
64     moose.Bye()
65     print("\nAccessing the class attribute:")
66     print(random.choice(Guru.sayings))
67
68 if __name__ == "__main__":
69     main()
```

```
$ greeter7_def.py
Hello World
I'm Rocky
Nothing is so simple that it cannot be
misunderstood.           Freeman Teague
Good Bye. And remember:
If you could get rid of yourself just once,
The secret of secrets would open to you.
                                Jalaluddin Rumi
Hello World
I'm Moose
Nothing is so simple that it cannot be
misunderstood.           Freeman Teague
Bye now.
Accessing the class attribute:
If you could get rid of yourself just once,
The secret of secrets would open to you.
                                Jalaluddin Rumi
$
```

Notes on discovering all about an object:

`an_object == another_object` --> True

They are the same builtin type
and have the same values for
all nested objects.

`an_object is another_object` --> True

|
|
"is" is a keyword! `id(an_object) == id(another_object)`
 They occupy the same spot in memory.
 `id()` is a builtin that is rarely
 used.

`isinstance(an_object, class-or-type-or-tuple)` --> True

If class-or-type-or-tuple is:

class -- if `an_object` is of the class or any subclass of it
type -- if `an_object` is the type
tuple -- if `an_object` is instance of any of the elements in the
tuple. The tuple elements can be classes and/or types.

`issubclass(C, B)` --> True

C is a subclass of any of the
classes in the tuple. B can be a
tuple of classes.

©Marilyn Davis, Aug 2020

printable_stack_def.py

```
1 #!/usr/bin/env python3
2 """printable_stack_def.py Extending our stack, providing a
3 "special method", __str__, which is called whenever:
4   1. f"{printable_stack_object}"
5   2. "{}".format(printable_stack_object)
6   3. str(printable_stack_object)
7   4. print(printable_stack_object) """
8
9 import sys, os
10 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
11 import Lab17_OOP.lab17_1 as stack_def # copy of lab exercise
12
13 class PrintableStack(stack_def.Stack):
14     """This class will reveal itself, and the result looks
15     like a stack.
16     """
17
18     def __str__(self):
19         try:
20             min_width = max([len(thing) for thing in self.things])
21         except ValueError: # self.things was empty
22             min_width = 4
23             center = " []"
24         else:
25             center = "|\\n|".join([thing.center(min_width)
26                                   for thing in self.things])
27         top = '+' + '-' * min_width + "\n|"
28         bottom = "|\\n " + '-' * min_width
29         return top + center + bottom
30
31 def main():
32     print("Printing a Stack is not pleasant.")
33     plain_box = stack_def.Stack()
34     print(plain_box)
35     print("\nPrintableStack shows its stack")
36     pbox = PrintableStack()
37     print(pbox)
38     for food in ["bread", "mayo", "cheese"]:
39         print("PrintableStack pushing", food)
40         pbox.Push(food)
41         print(pbox)
42
43
44     for i in range(3):
```

```
45         print("PrintableStack popping", pbox.Pop())
46         print(pbox)
47
48 if __name__ == "__main__":
49     main()
50
```

```
$ printable_stack_def.py
Printing a Stack is not pleasant.
<Lab17_00P.lab17_1.Stack object at 0x10d1dd198>
```

```
PrintableStack shows its stack
```

```
----
```

```
| [] |
```

```
----
```

```
PrintableStack pushing bread
```

```
----
```

```
|bread|
```

```
-----
```

```
PrintableStack pushing mayo
```

```
-----
```

```
|bread|
```

```
| mayo|
```

```
-----
```

```
PrintableStack pushing cheese
```

```
-----
```

```
|bread|
```

```
| mayo|
```

```
-----
```

```
PrintableStack popping cheese
```

```
-----
```

```
|bread|
```

```
| mayo|
```

```
-----
```

```
PrintableStack popping mayo
```

```
-----
```

```
|bread|
```

```
-----
```

```
PrintableStack popping bread
```

```
-----
```

```
| [] |
```

```
-----
```

```
$
```

©Marilyn Davis, 2007-2020

welcomer_def.py

```
1 #!/usr/bin/env python3
2 """
3 Another inheritance example, using the previous examples
4 by importing the old code. This one implements __call__,
5 __str__, and __del__ and a class attribute."""
6
7 import sys, os
8 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
9
10 import Lab17_OOP.lab17_4 as employee_def
11 import Lab17_OOP.greeter5_def as greeter_def
12
13 class Welcomer(greeter_def.NamedGreeter,
14                 employee_def.SalariedEmployee):
15     """Inherits from Salaried Employee"""
16     welcomers = 0
17
18     def __init__(self, name, pay_rate):
19         Welcomer.welcomers += 1
20         employee_def.SalariedEmployee.__init__(self, name,
21                                               pay_rate)
22
23     def __call__(self, something):
24         print(f'{something} yourself!')
25
26     def __del__(self):
27         Welcomer.welcomers -= 1
28         print(f'{self} says "Oh no!"')
29
30     def __str__(self):
31         return self.name
32
33 def main():
34     Joe = Welcomer("Joe", 20000) # style-guide anxiety here!
35     Joe.Greet()
36     print(Joe, f"here's ${Joe.CalculatePay(80):.2f} for you. ")
37     print()
38     marsha = Welcomer("Marsha", 19500)
39     marsha("Get to work")
40     print()
41     print(marsha, f"here's ${marsha.CalculatePay(80):.2f} for you.
41 ")
42     print()
43     print(f"There are {Welcomer.welcomers} welcomers.")
44     Joe("Goodbye")
45     print("Deleting Joe")
```

```
46     del Joe
47     print(f"There are {Welcomer.welcomers} welcomers.")
48     print()
49     marsha.Greet()
50     print("marsha is going out of scope so runs through del.")
51
52 if __name__ == "__main__":
53     main()
```

```
$ welcomer_def.py
Hello World
I'm Joe
Joe here's $30769.23 for you.
```

Get to work yourself!

Marsha here's \$30000.00 for you.

```
There are 2 welcomers.
Goodbye yourself!
Deleting Joe
Joe says "Oh no!"
There are 1 welcomers.
```

```
Hello World
I'm Marsha
marsha is going out of scope so runs through del.
Marsha says "Oh no!"
$
```

©Marilyn Davis, 2007-2020

circle_def.py

```
1 #!/usr/bin/env python3
2 """A Circle class, achieved by overriding __getitem__
3 which provides the behavior for indexing, i.e., [].
4 This also provides the correct cyclical behavior whenever
5 an iterator is used, i.e., for, enumerate() and sorted().
6 reversed() needs __reversed__ defined.
7 """
8
9 class Circle:
10
11     def __init__(self, data, times):
12         """Put the "data" in a circle that goes around
13         "times" times."""
14         self.data = data
15         self.times = times
16
17     def __getitem__(self, i):
18         """circle[i] --> Circle.__getitem__(circle, i)."""
19         l_self = len(self)
20         if i >= self.times * l_self:
21             raise IndexError(f"Circle object goes around {self.tim
21 es} times")
22         return self.data[i % l_self]
23
24     def __len__(self):
25         return len(self.data)
26
27 def main():
28     circle = Circle("around", 3)
29
30     print("Works with circle[i], for i > len(circle) too:")
31     for i in range(3 * len(circle) + 1):
32         try:
33             print(f"circle[{i:2d}] = {circle[i]}")
34         except IndexError as info:
35             print(info)
36             break
37
38     print("Works with sorted:")
39     print(sorted(circle))
40
41     print("Works for loops:")
42     small_circle = Circle("XO", 2)
43     for i, elementi in enumerate(small_circle):
44         print(f"small_circle[{i}] = {elementi}")
```

```

46
47     print("Works for nested loops:")
48     for i, elementi in enumerate(small_circle):
49         for j, elementj in enumerate(small_circle):
50             print(f"{i:3d}:{j:3d} -> {elementi}{elementj}")
51
52 if __name__ == "__main__":
53     main()
54

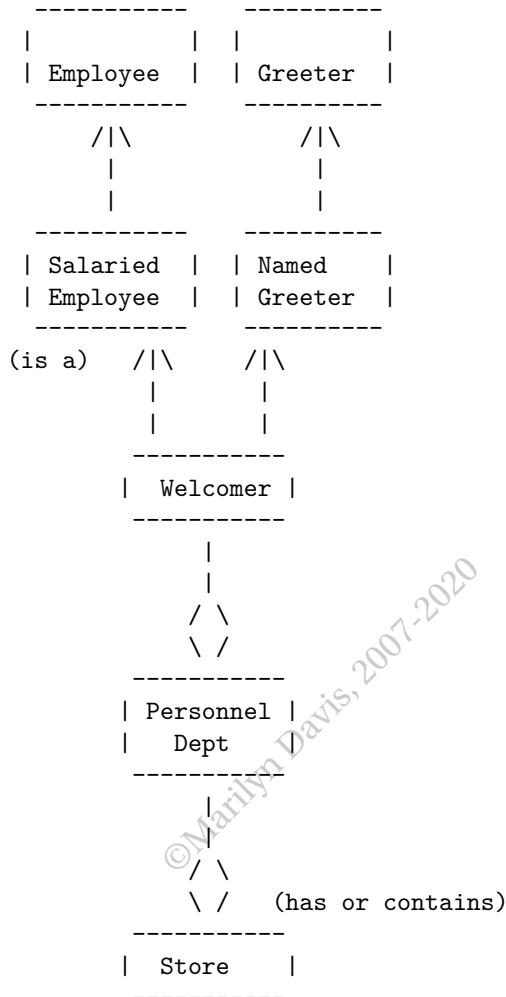
```

```

$ circle_def.py
Works with circle[i], for i > len(circle) too:
circle[ 0] = a
circle[ 1] = r
circle[ 2] = o
circle[ 3] = u
circle[ 4] = n
circle[ 5] = d
circle[ 6] = a
circle[ 7] = r
circle[ 8] = o
circle[ 9] = u
circle[10] = n
circle[11] = d
circle[12] = a
circle[13] = r
circle[14] = o
circle[15] = u
circle[16] = n
circle[17] = d
Circle object goes around 3 times
Works with sorted:
['a', 'a', 'a', 'd', 'd', 'd', 'n', 'n', 'n', 'o', 'o', 'o', 'r', 'r', 'r', 'u', 'u', 'u']
Works for loops:
small_circle[0] = X
small_circle[1] = O
small_circle[2] = X
small_circle[3] = O
Works for nested loops:
0: 0 -> XX
0: 1 -> XO
0: 2 -> XX
0: 3 -> XO
1: 0 -> OX
1: 1 -> OO
1: 2 -> OX          (Output continued)
1: 3 -> OO          3: 0 -> OX
2: 0 -> XX          3: 1 -> OO
2: 1 -> XO          3: 2 -> OX
2: 2 -> XX          3: 3 -> OO
2: 3 -> XO          $

```

Simplified Class Diagram for a Store, using
UML - Unified Modeling Language



store_def.py

```
1 #!/usr/bin/env python3
2 """
3
4 Here we implement the diagram, demonstrating a "contains a"
5 or "has a" relationship between classes, and a pseudo-private
6 attribute and method. Inheritance is a "is a" relationship.
7
8 Below, in PersonnelDept.__init__, notice that there is
9 self.__welcomers. This is a "private" attribute because it
10 has 2 leading underscores and no more than 1 trailing
11 underscore. It gets mangled to _PersonnelDept__welcomers
12 (_ClassName + attribute_name) so that, from outside the class,
13 __welcomers does not exist.
14 """
15 import welcomer_def
16
17 class Store:
18     __number_of_stores = 0
19
20     def __init__(self, name):
21         Store.__number_of_stores += 1
22         print(f"Store number {self.__number_of_stores} created.")
23         self.name = name
24         self.hr = PersonnelDept(self.name) # Store *has a*
25                                         # PersonnelDept
26
27 class PersonnelDept:
28     def __init__(self, name):
29         self.__welcomers = []
30         self.store_name = name
31
32     def Hire(self, name):
33         new_guy = welcomer_def.Welcomer(name, 30000)
34         self.__welcomers += [new_guy]
35         print(self.store_name, (
36             f"welcomes {new_guy}, our new welcomer."))
37         return new_guy
38
39     def __Find(self, name):
40         for (i, worker) in enumerate(self.__welcomers):
41             if worker.name == name:
42                 return i
43         return -1
```

```
47     def Fire(self, name):
48         index = self.__Find(name)
49         if index == -1:
50             return name, "doesn't work here."
51         x = self.__welcomers.pop(index)
52         print(f"{x}, you are terminated.")
53         print("Thank you and good luck.")
54
55 def main():
56     flormart = Store("FlorMart")
57     jane = flormart.hr.Hire("Jane")
58     jane.Greet()
59     print(jane, f"here's ${jane.CalculatePay(2):.2f} for you. ")
60     print("""Calling Jane("You're in trouble")""")
61     print(jane, "replies:")
62     jane("You're in trouble")
63     flormart.hr.Fire("Jane")
64
65 if __name__ == "__main__":
66     main()
```

```
$ store_def.py
Store number 1 created.
FlorMart welcomes Jane, our new welcomer.
Hello World
I'm Jane
Jane here's $1153.85 for you.
Calling Jane("You're in trouble")
Jane replies:
You're in trouble yourself!
Jane, you are terminated.
Thank you and good luck.
Jane says "Oh no!"
$ python3 -i store_def.py
[same output deleted, then:]
>>> gw = Store("Goodwill")
Store number 2 created.
>>> gw.hr.__welcomers
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'PersonnelDept' object has no attribute '__welcomers'
>>> gw.__number_of_stores
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Store' object has no attribute '__number_of_stores'
>>> gw._Store__number_of_stores
2
>>>
```

Lab 18 – Exercises:



1. If you have a file named lab18_1.py:

```
class X:  
    def __init__(self):  
        self.x = 1  
    def Which(self):  
        print("X")  
  
class A(X):  
    def __init__(self):  
        X.__init__(self)  
        self.y = 2  
  
class Y:  
    def __init__(self):  
        self.z = 3  
    def Which(self):  
        print("Y")  
  
class B(Y):  
    def __init__(self):  
        Y.__init__(self)  
        self.x = 4  
  
class AB(A, B):  
    pass  
  
ab = AB()
```

Guess the results if you were to run `python -i lab18_1.py` and then ask the interpreter to evaluate each of the following:

a _____ ab.x

b _____ ab.y

c _____ ab.z

d _____ ab.Which()

Please don't take the time to type in the code. It's fine to get the wrong answer. Just make your best quick guess. We'll discuss it.

2. Modify Labs/Lab17_OOP/tree_def.py so the classes have a `__str__` method instead of `Print`. Test with `print`, `f"tree"`, and `str(tree)`.
3. Optional project. Make a `tt tree.py` program. Running it on your `Lab_13_Portable_Python/cats` directory should show this information:

```
../Lab12_Portable_Python/cats/
|---cats.txt
|---more_cats.txt
|---../Lab12_Portable_Python/cats/deep_cats/
|   |---cats.txt
|   |---more_cats.txt
|   |---../Lab12_Portable_Python/cats/deep_cats/deeper_cats/
|   |   |---cats.txt
|   |   |---more_cats.txt
-----
3 directories
6 files
-----
```

4. (Optional big project but fun.) Make a `Clock` class.

It can be initialized like this: `t = Clock(2, 30)` to make a time = 2 hours and 30 minutes.

Values should be manipulated so that `minutes < 60` and `hours < 13`.

Override some of these:

- `str()` by providing `__str__()`.
- `+` by providing `__add__()`. When the interpreter sees:
`c1 = Clock(2, 30)`
`c2 = Clock(1, 15)`
`c3 = c1 + c2`

if you have provided `__add__()` for your `Clock` class, it will call:

`Clock.__add__(c1, c2)`

You want `c3` to also be an object of your `Clock` class.

- `-` by providing `__sub__()`.
- `repr()` by providing `__repr__()`. Test that your `__repr__` returns a string that is an evaluable Python expression.

Time and energy permitting, allow other styles of initialization:

- `Clock()` defaults to the current time.
- `Clock((2, 30))` a tuple
- `Clock("2:30")` a string
- `Clock({'hr':2, 'min':30})` a dictionary
- `Clock(min=30, hr=2)` keywords

©Marilyn Davis, 2007-2020

```
$ python -i lab18_1.py
>>> ab.x
1
>>> ab.y
2
>>> ab.z
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: AB instance has no attribute 'z'
>>> ab.Which()
X
```

©Marilyn Davis, 2007-2020

lab18_2.py

```
1 #!/usr/bin/env python3
2 """
3 lab18_2.py Tree and Forest classes with __str__.
4 """
5 import random
6 import sys
7
8 class Tree:
9     """Instantiate: Tree(20) to make a tree 20 ft tall.
10    """
11    def __init__(self, height):
12        self.height = height
13
14    def __str__(self):
15        return f"tree, {self.height:.1f} feet tall"
16
17 class Forest:
18     """Instantiate: Forest(size="medium")
19     if size == "large", it will have 8 trees.
20         == "medium", it will have 5 trees.
21         == "small", it will have 2 trees.
22     """
23    def __init__(self, size="medium"):
24        if size not in ("small", "medium", "large"):
25            raise ValueError(f"""Intantiate with: Forest([size="me
25 dium"]) where size can be "small", "medium", or "large", not "{size
25 e}"."""")
26        self.size = size
27        self.number_of_trees = 8 if self.size == "large" else 5 if
28        self.size == "medium" else 2
29        self.trees = [Tree(random.randrange(1, 200))
30                      for count in range(self.number_of_trees)]
31
32    def __str__(self):
33        say = f"{self.size} forest with {self.number_of_trees} tre
32 es:"
34        say += ", ".join([str(t) for t in self.trees])
35
36 def main():
37     forest = Forest("small")
38     print("print output:", forest, sep='\n')
39     print(f"f-string:\n{forest}")
40     print("str.format:\n{}".format(forest))
41     print("Format replacement:\n%{}" % forest)
42     print("And with str:\n" + str(forest))
```

```
43
44 if __name__ == "__main__":
45     main()

$ lab18_2.py
print output:
small forest with 2 trees:tree, 25.0 feet tall, tree, 96.0 feet tall
f-string:
small forest with 2 trees:tree, 25.0 feet tall, tree, 96.0 feet tall
str.format:
small forest with 2 trees:tree, 25.0 feet tall, tree, 96.0 feet tall
Format replacement:
small forest with 2 trees:tree, 25.0 feet tall, tree, 96.0 feet tall
And with str:
small forest with 2 trees:tree, 25.0 feet tall, tree, 96.0 feet tall
$
```

©Marilyn Davis, 2007-2020

lab18_3.py

```
1 #!/usr/bin/env python3
2 """lab18_3.py tree command in python"""
3 import os, sys
4
5 class Node:
6     too_many_slashes = None
7     def __init__(self, dir_name):
8         self.dir_name = dir_name
9         self.slashes = self.dir_name.count(os.sep)
10        self.root_dir = False
11        if self.too_many_slashes == None: # first directory
12            Node.too_many_slashes = self.slashes - 1
13            self.slashes = 0
14            self.root_dir = True
15        def __str__(self):
16            return " | " * self.slashes + "---" if not self.root_dir
17        else ''
18
19 class FileNode(Node):
20     first_dir = True
21     def __init__(self, dir_node, file_name):
22         Node.__init__(self, dir_node.dir_name)
23         self.path = os.path.join(dir_node.dir_name, file_name)
24         self.name = file_name
25         if FileNode.first_dir:
26             FileNode.first_dir = False
27         self.slashes -= self.too_many_slashes
28
29     def __str__(self):
30         return Node.__str__(self) + self.name
31
32 class DirNode(Node):
33     def __init__(self, dir_name):
34         Node.__init__(self, dir_name)
35         self.file_nodes = []
36         self.slashes -= self.too_many_slashes + 1
37
38     def AddFileNode(self, file_node):
39         self.file_nodes += [file_node]
40
41     def __str__(self):
42         return_str = Node.__str__(self) + self.dir_name + os.sep
43         files_part = '\n'.join(str(f) for f in self.file_nodes)
44         if self.file_nodes:
45             return return_str + '\n' + files_part
46         return return_str
```

```
46
47 class Tree:
48     def __init__(self, start_at):
49         self.dir_nodes = []
50     def IsGood(file_name):
51         BAD_ENDERS = '#', '~', "pyc"
52         BAD_STARTERS = '#', '.'
53         for ender in BAD_ENDERS:
54             if file_name.endswith(ender):
55                 return False
56         for starter in BAD_STARTERS:
57             if file_name.startswith(starter):
58                 return False
59         return True
60
61     for (this_dir, dir_names, file_names) in os.walk(start_at)
61 :
62         new_dir = DirNode(this_dir)
63         self.dir_nodes += [new_dir]
64         for file_name in sorted(file_names):
65             if not IsGood(file_name):
66                 continue
67             new_dir.AddFileNode(FileNode(new_dir, file_name))
68
69     def __str__(self):
70         return_str = '\n'.join(str(n) for n in sorted(self.dir_nod
70 es, key=lambda d:d.dir_name))
71         return f"""
72 {return_str}
73 -----
74 {len(self.dir_nodes):4} directories
75 {sum([len(d.file_nodes)for d in self.dir_nodes]):4} files
76 -----
77 """
78
79
80 def main():
81     if len(sys.argv) == 2:
82         dir_name = sys.argv[1]
83     else:
84         dir_name = input("dir name or nothing for current dir: ")
85
86     if not dir_name:
87         dir_name = '.'
88
89     tree = Tree(dir_name)
90     print(tree)
91
```

```
92
93 if __name__ == "__main__":
94     main()
95
$ lab18_3.py
dir name or nothing for current dir: ../Lab12_Portable_Python/cats
./Lab12_Portable_Python/cats/
|---cats.txt
|---more_cats.txt
|---../Lab12_Portable_Python/cats/deep_cats/
|   |---cats.txt
|   |---more_cats.txt
|   |---../Lab12_Portable_Python/cats/deep_cats/deeper_cats/
|       |---cats.txt
|       |---more_cats.txt
-----
3 directories
6 files
-----
$
```

©Marilyn Davis, 2007-2020

lab18_4.py

```
1 #!/usr/bin/env python3
2 """lab18_4.py  A Clock class"""
3
4 import time
5
6 class Clock:
7     """Clock() for now, or Clock(hours, minutes) or Clock(minutes)
8     or Clock("1:20") or Clock(dict) where dict has keys
9     "hours" and "minutes"""""
10
11     def __init__(self, *args, **dict_args):
12         no_args = len(args)
13         if dict_args:
14             self.__ResolveDictArgs(dict_args)
15         elif no_args <= 1:
16             if no_args == 0:
17                 # making args[0] Now
18                 args = [time.ctime()[11:16]]
19             if isinstance(args[0], str):
20                 self.hours, self.minutes = args[0].split(':')
21             elif isinstance(args[0], dict):
22                 self.__ResolveDictArgs(args[0])
23             else: # sequence or single value
24                 try:
25                     self.hours, self.minutes = args[0]
26                 except TypeError:
27                     self.hours = 0
28                     self.minutes = args[0]
29             elif no_args == 2:
30                 self.hours, self.minutes = args
31         else:
32             raise TypeError(Clock.__doc__)
33         self.__Normalize()
34
35     def __add__(self, other):
36         return Clock(self.hours + other.hours,
37                      self.minutes + other.minutes)
38
39     def __cmp__(self, other):
40         return cmp(int(self), int(other))
41
42     def __eq__(self, other):
43         if cmp(self, other) == 0:
44             return True
45         return False
46
```

```
47     def __int__(self):
48         return self.MinutesSince12()
49
50     def MinutesSince12(self):
51         return (self.hours % 12) * 60 + self.minutes
52
53     def __neg__(self):
54         return Clock(-self.hours, -self.minutes)
55
56     def __Normalize(self):
57         """Assumes that self.minutes and self.hours are floatable
58         and makes the values be ints on a clock.
59         """
60         self.minutes = float(self.minutes) + (float(self.hours)
61                               - int(self.hours)) * 60
62         self.minutes = int(round(self.minutes))
63         self.hours = int(self.hours) + self.minutes//60
64         self.minutes %= 60
65         self.hours = 1 + (self.hours - 1) % 12
66
67     def __repr__(self):
68         return self.__class__.__name__ + f"""({self})"""
69
70     def __ResolveDictArgs(self, dict_args):
71         try:
72             self.minutes = dict_args["minutes"]
73             self.hours = dict_args["hours"]
74         except KeyError:
75             raise TypeError(Clock.__doc__)
76
77     def __str__(self):
78         return f"{self.hours:2d}:{self.minutes:02d}"
79
80     def __sub__(self, other):
81         return Clock(hours=self.hours - other.hours,
82                      minutes=self.minutes - other.minutes)
83
84 def main():
85     Clock()
86     c1 = Clock(12, 59)
87     for hours in range(-2, 25, 2):
88         for minutes in range(-10, 10):
89             c2 = Clock(hours, minutes)
90             assert eval(repr(c2)) == c2, "eval, repr error"
91             # repr depends on str so
92             # both are tested
93             cmp_value = int(c2)
94             assert Clock(int(c2)) == c2, f"int not right: {c2}"
```

```
95         c_sum = c1 + c2
96         c_diff = c1 - c2
97         c3 = c_sum + c_diff # should be 2 * c1
98         c4 = Clock(2* c1.hours, 2 * c1.minutes)
99         assert c3 == c4, f"+/-/= error: {c1} +/- {c2}"
100        c5 = -c2
101        assert c_diff == c1 + c5
102        hours, minutes = 2, 30
103        clocks = [Clock(hours, minutes)]
104        clocks += [Clock((hours, minutes))]
105        clocks += [Clock(f"{hours}:{minutes:2d}")]
106        clocks += [Clock({"hours": hours, "minutes": minutes})]
107        clocks += [Clock(hours=hours, minutes=minutes)]
108        for each in clocks[1:]:
109            assert clocks[0] == each
110        try:
111            Clock(1, 2, 3)
112        except TypeError:
113            pass
114        else:
115            print("Clock(1, 2, 3) failed to raise an error")
116
117 if __name__ == "__main__":
118     main()
119
```

```
$ lab18_4.py
$
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 19 Extending Builtins

- Extending Builtin Classes
- Iterators
- Diamond Inheritance
- Encapsulation
- Static Methods (Optional)
- Class Methods (Optional)

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0

list_stack_def.py

```
1 #!/usr/bin/env python3
2 """A stack again, this time using the built-in "list" type.
3
4 A stack is just a list with a "push" method, since the list
5 already has a "pop". When it is-a "list" it inherits all
6 the builtin facilities of the list.
7 """
8
9 class Stack(list):
10
11     def push(self, thing):
12         # list.append(self, thing) works here.
13         # Also, the following works this time because
14         # this class does not have an "append" already.
15         self.append(thing)
16
17 if __name__ == "__main__":
18     stack = Stack()
19     stack.push("Gone With The Wind")
20     stack.push("Maltese Falcon")
21     stack.push("Fifth Element")
22     print("The stack has a rather nice __str__ already:")
23     print(stack)
24     print('The stack has all the list facilities, plus the "push":')
25
26     print(dir(stack))
27     print("Sorting then popping:")
28     stack.sort()
29     print(stack.pop())
```

```
$ list_stack_def.py
The stack has a rather nice __str__ already:
['Gone With The Wind', 'Maltese Falcon', 'Fifth Element']
The stack has all the list facilities, plus the "push":
['__add__', '__class__', [... __all_list__magic__ > deleted]
'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert',
'pop', 'push', 'remove', 'reverse', 'sort']
Sorting then popping:
Maltese Falcon
$
```

list_circle_def.py

```
1 #!/usr/bin/env python3
2 """A Circle class, derived from the builtin list class.
3
4 All the facilities of a list are available for free,
5 for using or overriding.
6 """
7
8 class Circle(list):
9
10     def __init__(self, data, times):
11         list.__init__(self, data)
12         self.times = times
13
14     def __getitem__(self, i):
15         """circle[i] --> Circle.__getitem__(circle, i)."""
16         l_self = len(self)
17         if i >= self.times * l_self:
18             raise IndexError(f"Circle object goes around {self.tim
19 es} times")
20             return list.__getitem__(self, i % l_self)
21
22     def __iter__(self):
23         """Because we are inheriting from list, and it has
24         its own __iter__, we need to override it to get all
25         the functionality we had before.
26         """
27         for i in range(self.times * len(self)):
28             yield self[i]
29
30 def main():
31     """Same main as lab18_Magic.circle_def
32     """
33     circle = Circle("around", 3)
34
35     print("Works with circle[i], for i > len(circle) too:")
36     for i in range(3 * len(circle) + 1):
37         try:
38             print(f"circle[{i:2d}] = {circle[i]}")
39         except IndexError as info:
40             print(info)
41             break
42
43     print("Works with sorted:")
44     print(sorted(circle))
45
46     print("Works for loops:")
```

```

46     small_circle = Circle("X0", 2)
47     for i, elementi in enumerate(small_circle):
48         print(f"small_circle[{i}] = {elementi}")
49
50     print("Works for nested loops:")
51     for i, elementi in enumerate(small_circle):
52         for j, elementj in enumerate(small_circle):
53             print(f"{i:3d}:{j:3d} -> {elementi}{elementj}")
54 def TestList():
55     circle = Circle("tic", 3)
56
57     print("--- \nTesting list functions:", circle)
58     circle += 'k'
59     print([x for x in circle])
60     circle.sort()
61     print(circle)
62
63 if __name__ == "__main__":
64     main()
65     TestList()
66

```

```

$ list_circle_def.py
[deleted output from main(): same as before.]
---
Testing list functions: ['t', 'i', 'c']
['t', 'i', 'c', 'k', 't', 'i', 'c', 'k', 't', 'i', 'c', 'k']
['c', 'i', 'k', 't']
$

# If we didn't define __iter__ here, the output would have been:

# Testing list functions: ['t', 'i', 'c']
# ['t', 'i', 'c', 'k']
# ['c', 'i', 'k', 't']

```

5in Note: Using super, and in general avoiding hardcoding the literal name of classes, makes your code more robust against change:

```

super(object)
Typical use to call a cooperative superclass method:
class C(B):
    def meth(self, arg):
        super(C, self).meth(arg)

```

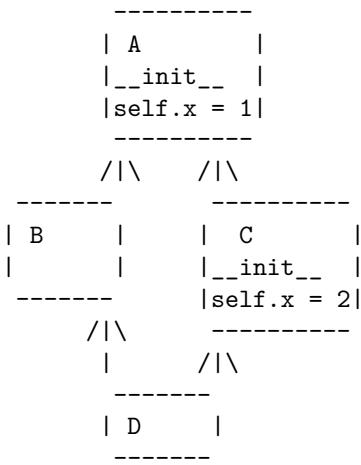
Note that super returns an object, so you don't need to put `self` in the argument list when you call to the immediate superclass' method.

circle_iter.py

```
1 #!/usr/bin/env python3
2 """(Optional) Here is the other way to make an iterator,
3 from scratch."""
4
5 import new_style_circle_def
6
7 class IterCircle(new_style_circle_def.Circle):
8
9     def __iter__(self):
10         """Returns an object that has a next() method that
11         raises a StopIteration error when it has exhausted
12         all the data.
13         """
14         class CircleIter:
15             def __init__(inner_self):
16                 inner_self.starting_data = self[:]
17                 inner_self.data_to_pop = self[:]
18                 inner_self.times = self.times
19
20             def __next__(inner_self):
21                 while True:
22                     try:
23                         return inner_self.data_to_pop.pop(0)
24                     except IndexError:
25                         pass
26                     inner_self.times = inner_self.times - 1
27                     if inner_self.times == 0:
28                         raise StopIteration
29                     inner_self.data_to_pop = inner_self.starting_d
30                     ata[:]
31
32             return CircleIter()
33 """
34 Same output as new_style_circle_def.py with same testing.
35 """
36 def main():
```

Same main as lab18_Magic.circle_def

Method Resolution: A Deeper Look



All cases of multiple inheritance are *diamond inheritance*.

`D().x = 1`

All classes inherit from Python's builtin super-est class **object**, completing the diamond.

`D().x = 2`

For this, **C3** rules are used, which are very complicated but usually this approximation is right:

1. List all the classes visited in the LRDF (left-right-depth-first) algorithm.

`[D, B, A, C, A]`

2. If there are duplicates, eliminate all but the last:

`[D, B, C, A]`

3. This rule does not produce correct C3 ordering when "two given base classes occur in a different order in the inheritance list of two different derived classes, and those derived classes are both inherited by yet another class".

To get the real **mro** (method resolution order) for your multiply-inheriting classes, use the class-level **mro()** function:

```
$ python3 -i diamond.py
2
>>> D.mro()
D.mro()
[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>,
 <class '__main__.A'>, <type 'object'>]
>>>
```

att.py

```
1 #!/usr/bin/env python3
2 """Classes are blueprints for name spaces. Attributes
3 can be added on an object by object basis. Feature or
4 flaw?"""
5
6 class NameSpace:
7
8     def __init__(self):
9         self.attribute = 10
10
11    def __str__():
12        return str(self.__dict__)
13
14 def main():
15     george = NameSpace()
16     george.age = "44"
17     george.job = "coder"
18
19     dinner = NameSpace()
20     dinner.maindish = "stew"
21     dinner.dessert = "pie"
22     dinner.attribute = "101"
23
24     print(george)
25     print(dinner)
26
27 if __name__ == "__main__":
28     main()
```

```
$ att.py
{'attribute': 10, 'age': '44', 'job': 'coder'}
{'attribute': 10, 'dessert': 'pie', 'attribute': '101', 'maindish': 'stew'}
$
```

att2.py

```
1 #!/usr/bin/env python3
2 """att2.py
3 You can override assigning and referencing attributes.
4
5 referencing:
6
7     object.x -> calls __getattr__ (if provided)
8
9     but only if the x attribute does not exist.
10
11 assignment:
12
13     object.x = 3 -> calls __setattr__ (if provided)
14 """
15 class Secret:
16     """The secret can only be set on initialization:
17     s = Secret("rose") and only the allowed attributes
18     can be set. """
19
20     __ALLOWED_ATTRIBUTES = ("members", "purpose")
21
22     def __init__(self, secret):
23         self.__dict__["_Secret__secret"] = secret
24         # Here we snuck around __setattr__ by adding directly
25         # to the __dict__, so that we could disallow the
26         # secret to be set after initialization. And, we had
27         # to do our own name mangling to keep it pseudo-secret.
28
29     def IsSecret(self, word):
30         return self.__secret == word
31
32     def __getattr__(self, attribute_name):
33         if attribute_name == "secret":
34             return "a secret!"
35         raise AttributeError(f"{self.__class__.__name__} instance
35 has no attribute '{attribute_name}'")
36
37     def __setattr__(self, attribute_name, value):
38         if attribute_name == "secret":
39             raise AttributeError(f"You can't change the {attribute
39 _name} to {value}")
40         if not attribute_name in self.__ALLOWED_ATTRIBUTES:
41             raise AttributeError(f"{attribute_name} is not an attr
41 ibute for class {self.__class__.__name__}")
42         self.__dict__[attribute_name] = value
43         # setattr(self, attribute_name) would loop forever!
```

```
44
45 def main():
46     club = Secret("snake")
47     print(f"club.secret is {club.secret}", end=' ')
48     print(f'club.IsSecret("snake") {club.IsSecret("snake")}')
49     try:
50         print(f"club.x is {club.x}")
51     except AttributeError as info:
52         print(f"\nError: {info}")
53     try:
54         print(f"club.members is {club.members}")
55     except AttributeError as info:
56         print(f"\nError: {info}")
57     club.members = 7
58     print(f"club.members is {club.members}")
59     try:
60         club.secret = "lizard"
61     except AttributeError as info:
62         print(f"Error: {info}")
63     try:
64         print("Setting club.x", end=' ')
65         club.x = "cucumber"
66     except AttributeError as info:
67         print(f"\nError: {info}")
68     print(dir(club))
69 if __name__ == "__main__":
70     main()
```

```
$ att2.py
club.secret is a secret! club.IsSecret("snake") True
club.x is
Error: Secret instance has no attribute 'x'
club.members is
Error: Secret instance has no attribute 'members'
club.members is 7
Error: You can't change the secret to lizard
Setting club.x
Error: x is not an attribute for class Secret
['IsSecret', '_Secret__ALLOWED_ATTRIBUTES', '_Secret__secret', '__doc__', '__getattr__', '__init__', '__module__', '__setattr__', 'members']
$
```

static.py

```
1 #!/usr/bin/env python3
2 """static.py (Optional) Class variables are supported and
3 work nicely, as expected, but there is no obvious way to
4 call a method unless you have an object."""
5
6 class Static:
7     number = 0
8
9     def __init__(self):
10         Static.number += 1
11         self.number = self.number
12         # or, more explicitly: self.number = Static.number
13
14
15     def __str__(self):
16         return f"{self.number} of {Static.number}"
17
18 def main():
19     objects = [Static() for i in range(3)]
20     print(", ".join([str(obj) for obj in objects]))
21
22 if __name__ == "__main__":
23     main()
```

```
$ static.py
1 of 3, 2 of 3, 3 of 3
$
```

static2.py

```

1 #!/usr/bin/env python3
2 """(Optional)
3 @staticmethod and @classmethod built-in decorators let you
4 invoke methods without having objects."""
5
6 import static
7
8 class Static2(static.Static):
9
10    @classmethod
11    def JumpUp(cls, number):          # cls will be the class
12        print(f"In classmethod(JumpUp), cls = {cls}")
13        static.Static.number += number
14
15    @staticmethod
16    def StartOver():                 # no self for a static method!
17        static.Static.number = 0
18
19 def main():
20    objects = [Static2() for i in range(3)]
21    print(f"""3 objects:
22 {", ".join([str(obj) for obj in objects])}""")
23    Static2.StartOver()
24    objects += [Static2() for i in range(3)]
25    print(f"""After StartOver() and 3 more objects:
26 {", ".join([str(obj) for obj in objects])}""")
27    Static2.JumpUp(100)
28    objects += [Static2() for i in range(3)]
29    print(f"""After JumpUp(100) and 3 more objects:
30 {", ".join([str(obj) for obj in objects])}""")
31
32 if __name__ == "__main__":
33     main()

```

```

$ static2.py
3 objects:
1 of 3, 2 of 3, 3 of 3
After StartOver() and 3 more objects:
1 of 3, 2 of 3, 3 of 3, 1 of 3, 2 of 3, 3 of 3
In classmethod(JumpUp), cls = <class '__main__.Static2'>
After JumpUp(100) and 3 more objects:
1 of 106, 2 of 106, 3 of 106, 1 of 106, 2 of 106, 3 of 106, 104 of 106, 105 of 106, 106
of 106
$
```

Lab 19 – Exercises:



1. Make a SortedDictionary class that inherits from the built-in dictionary, but the keys() method for your class returns a sorted list of keys. Also, provide an `__iter__` that iterates a sorted list of keys.

Make sure that any style of instantiation that you can use on a regular dictionary works on yours:

```
{1:'1', 2:'2'}, {}, ((1, '1'), (2, '2'))
```

(Optional) Allow someone instantiating your class to add a `description` attribute to an object, but no other attributes.

2. (Optional)

- Make a `Veggie` class and 3 specific vegetable classes that inherit from it. I chose `Asparagus`, `Corn` and `Squash`.
- Each of the sub-classes has only one class attribute, `bug_index`, which I initialized to be a small integer, less than 5; the sub-classes have no methods.
- The `Veggie` class has one class attribute, `weather`, which I initialized as "good". Other valid values are "ok" and "bad".
- Copy-and-paste the `PredictHarvest` method in
`Labs/Lab19_Extending_Builtins/predict_harvest_def.py`
into your `Veggie` class. To make this work, you'll also need the `weather_d` dictionary as a `Veggie` class attribute, also given.
- Provide a `UpdateBugIndex` method that is called by any sub-class of `Veggie` that updates the `bug_index` for that sub-class.
- Provide a `UpdateWeather` method that is called by any sub-class of `Veggie` or is called by `Veggie` itself, that changes the weather in the `Veggie` class.

3. (Optional) Use the list of cards we created in `Labs/Lab09_Functional_Programming/lab09_4` to make an iterating `Deck` class so that you can:

```
for card in Deck():
    print(card, end= ' ')
```

and the cards will appear shuffled.

Then, make a `GameDealer` (`no_players`, `no_cards`) class. Your `GameDealer` class will instantiate some player objects from a `Player` class that you also write for this exercise.

My test contains only 2 calls:

```
print(GameDealer(4, 5))
print(GameDealer(20, 3))
```

but you might do a better job of testing.

My last line of output is:

```
10 of Diamonds, Jack of Clubs, Sorry
```

which indicates that when the deck is exhausted, my `GameDealer` deals cards that say `Sorry`.

lab19_1.py

```
1 #!/usr/bin/env python3
2 """lab19_1.py A SortedDictionary class with only "description"
3 allowed as an attribute -- using __setattr__"""
4
5 class SortedDictionary(dict):
6
7     allowed_attributes = "description",
8
9     def keys(self):
10         return sorted(dict.keys(self))
11         # return sorted(super(type(self), self).keys())
12
13     def __iter__(self):
14         """If we don't define this, it will use the regular
15         dictionary __iter__ which does not call
16         SortedDictionary.keys()."""
17
18         for each in list(self.keys()):
19             yield each
20
21     def __setattr__(self, attribute_name, value):
22         if attribute_name not in SortedDictionary.allowed_attributes:
23             raise TypeError(f"Can't set attributes of class {self.__class__.__name__}")
24         self.__dict__[attribute_name] = value
25
26 def main():
27     # d is a tuple of constructs that will initialize
28     # a regular dictionary.
29     for initializer in (
30         {"Zero":0, "False":0, "None":0, "True":1},
31         {},
32         (("calling birds", 4), ("french hens", 3),
33          ("turtle doves", 2),
34          ("partridge in a pear tree", 1))):
35         regular_dict = dict(initializer)
36         sorted_dict = SortedDictionary(initializer)
37         print(f" regular_dict.keys(): {list(regular_dict.keys())}")
38
39         print(f" sorted_dict.keys(): {list(sorted_dict.keys())}")
40         print(f"""sorted_dict for-loop: {", ".join([
41             str(k) for k in sorted_dict])}""")
42
43         sorted_dict.description = "Fourth Day of Christmas"
44         print(f"sorted_dict.description = {sorted_dict.description}")
```

```
44     try:
45         regular_dict.description = "Fourth Day of Christmas"
46     except AttributeError:
47         pass
48     else:
49         print("Unexpected behavior!")
50     sorted_dict.x = 3
51
52 if __name__ == "__main__":
53     main()
54
```

```
$ lab19_1.py
regular_dict.keys(): ['False', 'Zero', 'True', 'None']
sorted_dict.keys(): ['False', 'None', 'True', 'Zero']
sorted_dict for-loop: False, None, True, Zero
regular_dict.keys(): []
sorted_dict.keys(): []
sorted_dict for-loop:
regular_dict.keys(): ['turtle doves', 'french hens', 'partridge in a pear tree', 'calling birds']
sorted_dict.keys(): ['calling birds', 'french hens', 'partridge in a pear tree', 'turtle doves']
sorted_dict for-loop: calling birds, french hens, partridge in a pear tree, turtle doves
sorted_dict.description = Fourth Day of Christmas
Traceback (most recent call last):
  File "./lab19_1.py", line 53, in <module>
    main()
  File "./lab19_1.py", line 50, in main
    sorted_dict.x = 3
  File "./lab19_1.py", line 23, in __setattr__
    raise TypeError, r
      f"can't set attributes of class {self.__class__.__name__}"
TypeError: can't set attributes of class SortedDictionary
$
```

lab19_2.py

```
1 #!/usr/bin/env python3
2 """
3 lab19_2.py (Optional) A Veggie class hierarchy, with a class
4 method and a staticmethod.
5 """
6
7 class Veggie:
8     """Instantiate: in subclass.
9     """
10
11    weather = "good"
12    weather_d = {"bad": 4, "ok": 7, "good": 10}
13
14    def PredictHarvest(self):
15        """Returns a sentence predicting the harvest for the
16            Veggie.
17        """
18        harvest_index = self.weather_d[self.weather] - self.bug_in-
19 dex
20        judgement = ("great" if harvest_index > 8
21                     else "ok" if harvest_index > 5
22                     else "disappointing" if harvest_index > 3
23                     else "zilch")
24        return f"{self} will be {judgement} this year."
25
26    @classmethod
27    def UpdateBugIndex(cls, bug_index):
28        """Changes the bug_index in the appropriate class.
29        """
30        if cls == Veggie:
31            raise ValueError(
32                "UpdateBugIndex must called from a subclass.")
33        cls.bug_index = bug_index
34
35    @staticmethod
36    def UpdateWeather(weather):
37        """Changes the weather in this class.
38        """
39        if weather not in Veggie.weather_d:
40            raise ValueError("weather must be "
41                             f"'{weather}' or '{'.join(Veggie.weather_d.keys(
42 ))}'")
43        Veggie.weather = weather
44
45    def __str__(self):
46        return self.__class__.__name__
```

```
45
46 class Asparagus(Veggie):
47     bug_index = 2
48
49 class Corn(Veggie):
50     bug_index = 1
51
52 class Squash(Veggie):
53     bug_index = 3
54
55 def main():
56     Veggie.UpdateWeather("good")
57     veggies = (Asparagus(), Corn(), Squash())
58     for veggie in veggies:
59         print(veggie.PredictHarvest())
60     asparagus, corn, squash = veggies
61     print("No bugs for asparagus.")
62     Asparagus.UpdateBugIndex(0)
63     for veggie in veggies:
64         print(veggie.PredictHarvest())
65     print('Weather degraded to "ok".')
66     Veggie.UpdateWeather("ok")
67     for veggie in veggies:
68         print(veggie.PredictHarvest())
69     try:
70         Veggie.UpdateWeather("hot")
71     except ValueError as msg:
72         print(msg)
73     else:
74         print("Unexpected behavior.")
75     try:
76         Veggie.UpdateBugIndex(7)
77     except ValueError as msg:
78         print(msg)
79     else:
80         print("Unexpected behavior.")
81
82 if __name__ == "__main__":
83     main()
84
```

```
$ lab19_2.py
Asparagus will be ok this year.
Corn will be great this year.
Squash will be ok this year.
No bugs for asparagus.
Asparagus will be great this year.
Corn will be great this year.
Squash will be ok this year.
```

```
Weather degraded to "ok".  
Asparagus will be ok this year.  
Corn will be ok this year.  
Squash will be disappointing this year.  
weather must be bad or good or ok.  
UpdateBugIndex must called from a subclass.  
$
```

©Marilyn Davis, 2007-2020

lab19_3.py

```
1 #!/usr/bin/env python3
2 """(Optional)An iterating Deck of cards.
3 """
4 import os, random, sys
5 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
6
7 import Lab09_Functional_Programming.lab09_5 as cards
8
9 class Deck:
10     """An iteratiing deck of cards that destroys each card
11     as it is taken with a next call - or as it is iterated
12     with a for-loop."""
13
14     def __init__(self):
15         self.__cards = cards.GetCards()
16         random.shuffle(self.__cards)
17
18     def __iter__(self):
19         return self
20
21     def __next__(self):
22         try:
23             return self.__cards.pop()
24         except IndexError:
25             raise StopIteration
26
27 class Player:
28     def __init__(self):
29         self.hand = []
30
31     def AcceptCard(self, card):
32         self.hand += [card]
33
34     def __str__(self):
35         return ", ".join(self.hand)
36
37 class GameDealer:
38     def __init__(self, no_players=4, no_cards=5):
39         self.no_cards = int(no_cards)
40         self.players = [Player() for p in range(int(no_players))]
41         self.deck = Deck()
42         self.DealHands()
43
44
45
46
```

```
47     def DealHands(self):
48         """Deals cards to the players."""
49         for card_no in range(self.no_cards):
50             for player in self.players:
51                 try:
52                     new_card = next(self.deck)
53                 except StopIteration:
54                     new_card = "Sorry"
55                 player.AcceptCard(new_card)
56
57     def __str__(self):
58         """Returns a string representation of the dealt
59         cards."""
60         return '\n'.join([str(p) for p in self.players])
61
62 def main():
63     for players, cards in ((4, 5), (20, 3)):
64         print(f"GameDealer({players}, {cards})")
65         print(GameDealer(players, cards))
66
67 if __name__ == "__main__":
68     main()
```

```
$ lab19_3.py
GameDealer(4, 5)
4 of Clubs, 6 of Hearts, 6 of Diamonds, Jack of Hearts, 2 of Hearts
Queen of Hearts, Ace of Spades, 5 of Clubs, 7 of Clubs, King of Diamonds
Joker, 3 of Diamonds, Ace of Diamonds, 4 of Hearts, 2 of Clubs
10 of Diamonds, 8 of Clubs, Ace of Clubs, King of Clubs, 9 of Clubs
GameDealer(20, 3)
6 of Hearts, King of Clubs, Jack of Spades
3 of Hearts, Joker, Queen of Spades
5 of Clubs, 7 of Diamonds, 6 of Diamonds
[12 hands deleted]
King of Spades, 9 of Hearts, Sorry
4 of Spades, 8 of Diamonds, Sorry
4 of Hearts, 6 of Clubs, Sorry
8 of Clubs, 10 of Diamonds, Sorry
7 of Spades, Queen of Hearts, Sorry
$
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

Online available until June 21

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 20 Developer Modules

- Raw File I/O
- raise an exception
- try/except/else/finally
- Context Manager Class
- Module: unittest (Optional)
- Module: optparse (Optional)

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

v.7.0

file1.py

```

1 #!/usr/bin/env python3
2 """Demonstrates reading a file line by line."""
3
4 def PrintFile(f_name):
5     with open(f_name) as open_file:
6         for line in open_file:
7             print(line, end=' ')
8
9 def main(f_name):
10    PrintFile(f_name)
11
12 if __name__ == "__main__":
13    main("ram_tzu.txt")

```

```

$ file1.py
Ram Tzu knows this:
When God wants you to do something,
you think it's your idea.
$
```

Raising your exceptions yourself.

It's easy: use the keyword ***raise***.

```

>>> raise RuntimeError("a problem happened")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: a problem happened
>>>
```

Usually you want to use a built-in exception.

To see all the built-in exceptions: l

```

>>> dir(__builtins__)
['ArithmetError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError', 'BytesWarning',
'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning',
'IndentationError', 'IndexError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
'NameError', 'None', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
'PendingDeprecationWarning', 'ReferenceError', 'RuntimeError', 'RuntimeWarning', 'StandardError',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True',
'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError',
'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', ...]
```

The `dir(__builtins__)` report starts with the Exception classes, all capitalized. Notice that `False`, `None` and `True`, are also in the capitalized list. The output next lists underscored items, and then all the builtin, lower-case, functions. The `__builtins__` module is always available with import.

file2.py

```
1 #!/usr/bin/env python3
2 """Demonstrates the "finally" clause.
3
4 Finally happens whether or not there's an exception and it
5 passes the exception up to the surrounding try/except.
6
7 * You cannot put an except and finally with the same try ...
8 unless you are running Python 2.5+.
9 + The finally clause happens, no matter what, even if there
10 is a return in the try clause.
11 """
12
13 def PrintFile(f_name):
14     file_obj = open(f_name)
15     try:
16         for line in file_obj:
17             print(line, end=' ')
18     finally:
19         file_obj.close()
20
21 def main(file_name="ram_tzu.txt"):
22     try:
23         PrintFile(file_name)
24     except IOError as info:
25         print(info)
26
27 if __name__ == "__main__":
28     import sys
29     try:
30         main(sys.argv[1])
31     except IndexError:
32         main()
33
```

```
$ file2.py
Ram Tzu knows this:
When God wants you to do something,
you think it's your idea.
$ file2.py xy
[Errno 2] No such file or directory: 'xy'
```

\$

file3.py

```
1 #!/usr/bin/env python3
2 """Demonstrates the "finally" clause -- and making it happen.
3 """
4 def PrintFile(file_name, fail_on_read=False):
5     try:
6         open_file = file(file_name) #file is an alias for open
7         try:
8             for line in open_file:
9                 print(line, end='')
10            if fail_on_read:
11                raise IOError("Failed while reading.")
12        finally:
13            print("Finally")
14            open_file.close()
15    except IOError as info:
16        print(info)
17
18 def main(file_name="ram_tzu.txt"):
19     print(f'\nCalling PrintFile("{file_name}")')
20     PrintFile(file_name)
21     print(f"\nCalling PrintFile("{file_name}", fail_on_read=True"
21 )"""
22     PrintFile(file_name, fail_on_read=True)
23     print("\nCalling PrintFile("absent_file")")
24     PrintFile("absent_file")
25
26 if __name__ == "__main__":
27     main()
```

Calling PrintFile("ram_tzu.txt")

Ram Tzu knows this:

When God wants you to do something,
you think it's your idea.

Finally

Calling PrintFile("ram_tzu.txt", fail_on_read=True)

Ram Tzu knows this:

Finally

Failed while reading.

Calling PrintFile("absent_file")

[Errno 2] No such file or directory: 'absent_file'

\$

file4.py

```
1 #!/usr/bin/env python3
2 """The finally can be attached to the try/except since
3 Python 2.5. But, the finally always happens last, and
4 doesn't re-raise.
5 """
6 def PrintFile(file_name, fail_on_read=False):
7     try:
8         file_obj = open(file_name)
9         for line in file_obj:
10             print(line, end=' ')
11             if fail_on_read:
12                 raise IOError("Failed while reading.")
13     except IOError as info:
14         print(info)
15     finally:
16         print("Finally")
17         try:
18             file_obj.close()
19         except UnboundLocalError:
20             pass
21
22 def main(file_name="ram_tzu.txt"):
23     """Same main as file3.py
24     """
25     print(f"\nCalling PrintFile({file_name})")
26     PrintFile(file_name)
27     print(f"\nCalling PrintFile({file_name}, fail_on_read=True")
27 )"""
28     PrintFile(file_name, fail_on_read=True)
29     print("\nCalling PrintFile('absent_file')")
30     PrintFile("absent_file")
31
32 if __name__ == "__main__":
33     main()
```

Calling PrintFile("ram_tzu.txt")

Ram Tzu knows this:

When God wants you to do something,
you think it's your idea.

Finally

Calling PrintFile("ram_tzu.txt", fail_on_read=True)

Ram Tzu knows this:

Failed while reading. Different order

Finally

Calling PrintFile("absent_file")

[Errno 2] No such file or directory: 'absent_file'

Finally Another finally

context.py

```

1 #!/usr/bin/env python3
2 """Making a context manager."""
3
4 class OpenClose:
5
6     def __init__(self, file_name, mode="r"):
7         self.file_name = file_name
8         self.mode = mode
9
10    def __enter__(self):
11        self.obj = open(self.file_name, self.mode)
12        return self.obj
13
14    def __exit__(self, exc_type, exc_val, exc_tb):
15        print("Like finally: in __exit__", exc_type)
16        self.obj.close()
17
18 def PrintFile(file_name, fail_on_read=False):
19     try:
20         with OpenClose(file_name) as file_object:
21             for line in file_object:
22                 print(line, end=' ')
23                 if fail_on_read:
24                     raise IOError("Failed while reading.")
25     except IOError as info:
26         print(info)
27
28 def main(file_name="ram_tzu.txt"):
29     """Same main as file3.py
30     """
31     print(f"\nCalling PrintFile({file_name})")
32     PrintFile(file_name)
33     print(f"\nCalling PrintFile({file_name}, fail_on_read=True")
33 )"""
34     PrintFile(file_name, fail_on_read=True)
35     print("\nCalling PrintFile('absent_file')")
36     PrintFile("absent_file")
37
38 if __name__ == "__main__":
39     main()
40

```

\$ context.py

Calling PrintFile("ram_tzu.txt")
 Ram Tzu knows this:
 When God wants you to do something,
 you think it's your idea.

```
Like finally: in __exit__ None

Calling PrintFile("ram_tzu.txt", fail_on_read=True)
Ram Tzu knows this:
Like finally: in __exit__ <class 'OSError'>
Failed while reading.

Calling PrintFile("absent_file")
[Errno 2] No such file or directory: 'absent_file'
$
```

context2.py

```
1 #!/usr/bin/env python3
2 """Using the OpenClose context manager class. """
3
4 import context
5
6 def WriteFile(file_name, text):
7     try:
8         with context.OpenClose(file_name, "w") as file_object:
9             file_object.write(text)
10    except IOError as info:
11        print(info)
12
13 if __name__ == "__main__":
14     WriteFile("sometimes", "Sometimes you win.\n")
15
```

```
$ context2.py
Like finally: in __exit__ None
$ cat sometimes
Sometimes you win.
$
```

pyunit.py

```
1 #!/usr/bin/env python3
2 """(Optional) This example of unittest is taken from
3 http://www.python.org/doc/lib/module-unittest.html."""
4 import random
5 import unittest
6
7 class TestSequenceFunctions(unittest.TestCase):
8
9     def setUp(self):
10         self.seq = list(range(10))
11
12     def testShuffle(self):
13         # make sure the shuffled sequence does not
14         # lose any elements
15         random.shuffle(self.seq)
16         self.seq.sort()
17         self.assertEqual(self.seq, list(range(10)))
18
19     def testChoice(self):
20         element = random.choice(self.seq)
21         self.assertTrue(element in self.seq)
22
23     def testSample(self):
24         self.assertRaises(ValueError, random.sample,
25                           self.seq, 20)
26         for element in random.sample(self.seq, 5):
27             self.assertTrue(element in self.seq)
28
29 if __name__ == "__main__":
30     unittest.main()
```

```
$ pyunit.py
```

```
...
```

```
Ran 3 tests in 0.006s
```

```
OK  # <-- Note: Idle will crash after you see OK. Your code is fine!
$
```

Each of the `assertSomething` methods also invites a `msg` parameter:

```
17     self.assertEqual(self.seq, range(3), msg="elements lost")
```

This, on failure gives the message:

```
..F
=====
FAIL: testShuffle (__main__.TestSequenceFunctions)
-----
Traceback (most recent call last):
  File "./pyunit.py", line 17, in testShuffle
    self.assertEqual(self.seq, range(3), msg="elements lost")
AssertionError: elements lost

-----
Ran 3 tests in 0.001s

FAILED (failures=1)
```

Sticking with the default `msg`:

```
self.assertEqual(self.seq, range(3))
```

gives a lot of information:

```
./pyunit.py
..F
=====
FAIL: testShuffle (__main__.TestSequenceFunctions)
-----
Traceback (most recent call last):
  File "./pyunit_msg_default.py", line 17, in testShuffle
    self.assertEqual(self.seq, range(3))
AssertionError: Lists differ: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] != [0, 1, 2]

First list contains 7 additional elements.
First extra element 3:
3

- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
+ [0, 1, 2]

-----
Ran 3 tests in 0.001s

FAILED (failures=1)
```

protect_pdf.py

```
1 #!/usr/bin/env python3
2 """(Optional) protect_pdf.py [-h for complete help] input_pdf_file
3 -p password [-o output_pdf_file] [-v for verbose]
4
5 Puts password protection on the renamed input_pdf_file.
6
7 If output_pdf_file is not given, it manufactures it as:
8 pdf_file = ".pdf" + '_protected' + ".pdf"
9
10 Demonstrates the optparse module for parsing the command line
11 options and putting them into suitably-named identifiers in a
12 namespace.
13 """
14 import optparse
15 import subprocess
16
17 def ProtectPDF(options):
18     """options is a namespace with these attributes:
19
20         input_pdf_file = file to be password protected
21         user_pw = password
22         output_pdf_file = password protected version
23         verbose = True or False
24
25 Note, the unprotected input_pdf_file still exists.
26 """
27     if not options.output_pdf_file:
28         front, back = options.input_pdf_file.rsplit('.', 1)
29         options.output_pdf_file = f"{front}_secured.{back}"
30
31     command_sequence = ["pdftk", options.input_pdf_file,
32                         "output", options.output_pdf_file,
33                         "owner_pw", "FriedFlavors",
34                         "user_pw", options.password, "allow",
35                         "Printing", "DegradedPrinting",
36                         "ScreenReaders"]
37
38     if options.verbose:
39         print("Calling:", " ".join(command_sequence))
40
41     command_process = subprocess.Popen(command_sequence,
42                                         stderr=subprocess.PIPE)
43     some_output = False
44     for line in command_process.stderr:
45         some_output = True
46         line = str(line, encoding="ascii")
47         print(line, end='')
```

```
47     if options.verbose and not some_output: # success!
48         print("Done.", end=' ')
49         print(f"{options.output_pdf_file} is password-protected wi-
50 th the password={options.password}")
51
51 def CollectCommand(parser):
52     """Here we have the parser harvest the command line and
53     we check that we have an appropriate arguments.
54     """
55     (options, args) = parser.parse_args()
56     if len(args) != 1:
57         parser.error("A pdf file name is needed.")
58     options.input_pdf_file = args[0]
59     if not options.password:
60         parser.error("A password is needed.")
61     return options
62
63 def main():
64     parser = SetUpParsing()
65     options = CollectCommand(parser)
66     ProtectPDF(options)
67
68 def SetUpParsing():
69     """Calls add_option repeatedly, once for every unix-style
70     option we need. """
71
72     parser = optparse.OptionParser("%prog [-h for complete help] i-
72 nput_pdf_file -p password [-o output_pdf_file] [-v for verbose]")
73
74     parser.add_option("-o", "--output_pdf_file", dest="output_pdf_
74 file", default=None, help='OPTIONAL: Write the password-protected
74 pdf file to output_pdf_file, if given. Default is input_pdf_file
74 with "secured" inserted before the ".pdf".')
75     parser.add_option("-p", "--password", dest="password", help="p-
75 assword for protecting the input_pdf_file")
76     parser.add_option("-v", "--verbose", dest="verbose", action="s-
76 tore_true", default=False, help="OPTIONAL: Details of the processi-
76 ng are written to stdout.")
77     return parser
78
79 if __name__ == "__main__":
80     main()
```

```
$ protect_pdf.py
Usage: protect_pdf.py [-h for complete help] input_pdf_file -p password [-o output_pdf_file]
[-v for verbose]

protect_pdf.py: error: A pdf file name is needed.
$ protect_pdf.py --help
Usage: protect_pdf.py [-h for complete help] input_pdf_file -p password [-o output_pdf_file]
[-v for verbose]

Options:
-h, --help            show this help message and exit
-o OUTPUT_PDF_FILE, --output_pdf_file=OUTPUT_PDF_FILE
                      OPTIONAL: Write the password-protected pdf file to
                      output_pdf_file, if given. Default is input_pdf_file
                      with "secured" inserted before the ".pdf".
-p PASSWORD, --password=PASSWORD
                      password for protecting the input_pdf_file
-v, --verbose         OPTIONAL: Details of the processing are written to
                      stdout.

$ protect_pdf.py syllabus.pdf
Usage: protect_pdf.py [-h for complete help] input_pdf_file -p password [-o output_pdf_file]
[-v for verbose]

protect_pdf.py: error: A password is needed.
$ protect_pdf.py syllabus.pdf -p gopher -v
Calling: pdftk syllabus.pdf output syllabus_secured.pdf owner_pw FriedFlavors user_pw gopher
allow Printing DegradedPrinting ScreenReaders
Done. syllabus_secured.pdf is password-protected with the password=gopher
$ protect_pdf.py syllabus.pdf -p gopher
$ protect_pdf.py nonexistent.pdf -p gopher -v
Calling: pdftk nonexistent.pdf output nonexistent_secured.pdf owner_pw FriedFlavors user_pw
gopher allow Printing DegradedPrinting ScreenReaders
Error: Unable to find file.
Error: Failed to open PDF file:
nonexistent.pdf
Errors encountered. No output created.
Done. Input errors, so no output created.
$ protect_pdf.py syllabus.pdf -o syllabus_gopher.pdf -p gopher
$ ls *.pdf
syllabus.pdf           syllabus_gopher.pdf          syllabus_secured.pdf
$
```

Lab 20 – Exercises:



Important Note: If you develop a unittest under Idle, it will, after successfully running your test, generate an error when it quits. This error has nothing to do with your code.

1. (Optional) Develop a test class for the `Game_Dealer` class you developed. Or, use my solution:
`Labs/Lab19_Extending_Builtins/lab19_3.py`
2. (Optional) Create a command-line program to deal card games, also using `lab19_3.py`.
 - `lab20_2.py` – deals 4 hands of 5 cards each, the default.
 - `lab20_2.py -p 6 -c 3` – deals 6 hands (for 6 players) of 3 cards each
3. (Optional) Make a `FileLogger` context manager class that seems to behave the same as the `OpenClose` context manager class, but it secretly keeps a log of every file it opens and closes, and the time that it was opened or closed.

©Marilyn Davis, 2007-2020

©Marilyn Davis, 2007-2020

lab20_1.py

```
1 #!/usr/bin/env python3
2 """(Optional) Test for GameDealer class."""
3
4 import unittest
5
6 import os, sys
7 from functools import reduce
8 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], ".."))
9
10 import Lab19_Extending_Builtins.lab19_3 as game_dealer
11
12 WHOLE_DECK = sorted(game_dealer.Deck())
13
14 class ReportingDealer(game_dealer.GameDealer):
15     """GameDealer only had methods that output strings,
16     so here we provide a list version for testing.
17     """
18     def Report(self):
19         """For testing."""
20         return [p.hand for p in self.players]
21
22 class TestPlayCards(unittest.TestCase):
23
24     def testSmall(self):
25         little = ReportingDealer(1, 1).Report()
26         self.assertEqual(len(little), 1)
27         self.assertEqual(len(little[0]), 1)
28         self.assertTrue(little[0][0] in WHOLE_DECK)
29
30     def testZilch(self):
31         self.assertEqual([], ReportingDealer(0, 1).Report())
32         self.assertEqual([[]], ReportingDealer(1, 0).Report())
33         self.assertEqual([], ReportingDealer(0, 0).Report())
34
35     def testWholeDealer(self):
36         all_hands = ReportingDealer(9, 6).Report()
37         for hand in all_hands:
38             self.assertEqual(len(hand), 6)
39         self.assertEqual(len(all_hands), 9)
40         all_hands_collapsed = sorted(
41             reduce(lambda x, y: x + y, all_hands))
42         self.assertEqual(all_hands_collapsed, WHOLE_DECK)
43
44
45
46
```

```
47
48     def testTooMany(self):
49         too_many = ReportingDealer(11, 5).Report()
50         too_many_collapsed = reduce(lambda x, y: x + y,
51                                      too_many)
52         self.assertTrue("Sorry" in too_many_collapsed)
53         too_many_collapsed.remove("Sorry")
54         too_many_collapsed.sort()
55         self.assertEqual(too_many_collapsed, WHOLE_DECK)
56
57     def testWayTooMany(self):
58         way_too_many = ReportingDealer(11, 6).Report()
59         way_too_many_collapsed = reduce(lambda x, y: x + y,
60                                         way_too_many)
61         self.assertEqual(len(way_too_many_collapsed), 66)
62         self.assertEqual(way_too_many_collapsed.count("Sorry"),
63                         12)
64         for i in range(12):
65             way_too_many_collapsed.remove("Sorry")
66         way_too_many_collapsed.sort()
67         self.assertEqual(way_too_many_collapsed, WHOLE_DECK)
68
69 if __name__ == "__main__":
70     unittest.main()
71
```

```
$ lab20_1.py
```

```
....
```

```
Ran 5 tests in 0.005s
```

```
OK
```

```
$
```

lab20_2.py

```

1 #!/usr/bin/env python3
2 """lab20_2.py (Optional) -- deals card hands:
3 lab20_2.py -- deals 4 hands of 5 cards
4 lab20_2.py -p 6 -c 3 -- deals 6 hands of 3 cards
5 """
6 import os, sys
7 sys.path.insert(0, os.path.join(os.path.split(__file__)[0], '..'))
8 import Lab19_Extending_Builtins.lab19_3 as game_dealer
9
10 def main():
11     import optparse
12     parser = optparse.OptionParser(
13         "%prog [-p number_of_players=4] [-c number_of_cards=5]")
14     parser.add_option("-p", "--players", dest="no_players",
15                       help="number of players", default=4)
16     parser.add_option("-c", "--cards", dest="no_cards",
17                       help="number of cards per hand",
18                       default=5)
19     (options, args) = parser.parse_args()
20     if len(args) > 0:
21         parser.error(f"I don't recognize {' '.join(args)}")
22     print(game_dealer.GameDealer(options.no_players,
23                                   options.no_cards))
24
25 if __name__ == "__main__":
26     main()

```

```

$ lab20_2.py
© Marilyn Davis, 2020
Joker, 6 of Diamonds, Ace of Hearts, 4 of Clubs, 2 of Clubs
9 of Clubs, King of Spades, 4 of Hearts, King of Diamonds, 7 of Clubs
10 of Hearts, 5 of Diamonds, Queen of Diamonds, 2 of Hearts, 3 of Diamonds
8 of Hearts, 4 of Diamonds, 9 of Diamonds, 10 of Clubs, 3 of Hearts
$ lab20_2.py -x
Usage: lab20_2.py [-p number_of_players=4] [-c number_of_cards=5]

```

```

lab20_2.py: error: no such option: -x
$ lab20_2.py -help
Usage: lab20_2.py [-p number_of_players=4] [-c number_of_cards=5]

```

Options:

-h, --help	show this help message and exit
-p NO_PLAYERS, --players=NO_PLAYERS	number of players
-c NO_CARDS, --cards=NO_CARDS	number of cards per hand

```

$ lab20_2.py -p 6 -c 3
Jack of Spades, 10 of Diamonds, Ace of Clubs
9 of Clubs, 4 of Spades, Joker
9 of Diamonds, Jack of Diamonds, 10 of Spades
9 of Hearts, 7 of Spades, 3 of Diamonds

```

```
7 of Hearts, 7 of Diamonds, King of Diamonds
Ace of Hearts, 10 of Hearts, 8 of Hearts
$ lab20_2.py -p 11
9 of Spades, King of Clubs, 5 of Spades, 6 of Hearts, Queen of Clubs
10 of Spades, 2 of Hearts, 9 of Diamonds, 3 of Clubs, Jack of Hearts
10 of Clubs, 6 of Clubs, Queen of Diamonds, 3 of Hearts, Jack of Spades
5 of Hearts, King of Spades, King of Hearts, Jack of Clubs, 10 of Hearts
8 of Hearts, Ace of Hearts, 8 of Spades, 7 of Spades, 9 of Clubs
Queen of Hearts, 5 of Diamonds, Joker, 7 of Diamonds, 8 of Diamonds
Ace of Spades, 5 of Clubs, 2 of Diamonds, 4 of Clubs, 4 of Spades
Jack of Diamonds, 2 of Clubs, 10 of Diamonds, 6 of Diamonds, 9 of Hearts
Ace of Clubs, 8 of Clubs, Joker, 7 of Clubs, 4 of Hearts
Ace of Diamonds, 3 of Diamonds, 6 of Spades, 2 of Spades, 7 of Hearts
4 of Diamonds, King of Diamonds, 3 of Spades, Queen of Spades, Sorry
$
```

©Marilyn Davis, 2007-2020

lab20_3.py

```
1 #!/usr/bin/env python3
2 """Provides a FileLogger(file_name) context manager class, that lo-
3 gs the opening and closing of the file_name.
4 """
5 #!/usr/bin/env python
6 """Provides FileLogger(file_name) context manager class.
7 """
8 import context, datetime
9
10 class FileLogger(context.OpenClose):
11     log_file_name = "file.log"
12
13     def __enter__(self):
14         self.logger_object = open(FileLogger.log_file_name, 'a')
15         self.logger_object.write(f"""Opening "{self.file_name}" in
15 mode '{self.mode}' on {datetime.datetime.now()}\n""")
16         return context.OpenClose.__enter__(self)
17
18     def __exit__(self,*args):
19         self.logger_object.write(f'Closing "{self.file_name}" on {
19 datetime.datetime.now()}\n\n')
20         self.logger_object.close()
21         context.OpenClose.__exit__(self, *args)
22
23 def main():
24     for loop in range(3):
25         with FileLogger("words.txt", "a") as file_obj:
26             file_obj.write("Writing some words at {}.\n".format(
27                 datetime.datetime.now()))
28         print("words.text:")
29         print(open("words.txt").read())
30         print(FileLogger.log_file_name + ":")
31         print(open(FileLogger.log_file_name).read())
32
33 main()
```

```
$ lab20_3.py
Like finally: in __exit__ None
Like finally: in __exit__ None
Like finally: in __exit__ None
words.text:
Writing some words at 2019-01-16 15:21:32.842240.
Writing some words at 2019-01-16 15:21:32.842939.
Writing some words at 2019-01-16 15:21:32.843458.

file.log:
Opening "words.txt" in mode 'a' on 2019-01-16 15:21:32.842006
```

```
Closing "words.txt" on 2019-01-16 15:21:32.842258
Opening "words.txt" in mode 'a' on 2019-01-16 15:21:32.842819
Closing "words.txt" on 2019-01-16 15:21:32.842953
$
```

©Marilyn Davis, 2007-2020

PYTHON

LAB BOOK

Python For Programmers
UCSC-Extension CMPR.X416.(37)

June 8 - 11, 2020

These written materials are a prop to motivate the lectures in the course:

Python For Programmers

UCSC-Extension CMPR.X416.(37)

They are not intended to be studied without the lectures and are likely to confuse you rather than help you to learn Python if you attempt to study this Lab Book without the associated lectures.

Lab 21 Wrap Up

- Exceptions:
- traceback module
- raise exceptions
- Inventing exceptions
- Namespaces
- Nests
- Pitfalls
- Finding Modules and Help

©2007-2020 by Marilyn Davis, Ph.D.
All rights reserved.

Exceptions

```
>>> help('exceptions')
```

gives you lots of info about exceptions. For example, exceptions are classes, in a hierarchy:

```
Exception
|
+-- SystemExit
+-- StopIteration
+-- StandardError
|   +-- KeyboardInterrupt
|   +-- ImportError
|   +-- EnvironmentError
|   |   +-- IOError
|   |   +-- OSError
|   |   +-- WindowsError
|   +-- EOFError
|   +-- RuntimeError
|   |   +-- NotImplementedError
|   +-- NameError
|   |   +-- UnboundLocalError
|   +-- AttributeError
|   +-- SyntaxError
|   |   +-- IndentationError
|   |   +-- TabError
|   +-- TypeError
|   +-- AssertionError
|   +-- LookupError
|   |   +-- IndexError
|   |   +-- KeyError
|   +-- ArithmeticError
|   |   +-- OverflowError
|   |   +-- ZeroDivisionError
|   |   +-- FloatingPointError
|   +-- ValueError
|   |   +-- UnicodeError
|   +-- ReferenceError
|   +-- SystemError
|   +-- MemoryError
+---Warning
    +-- UserWarning
    +-- DeprecationWarning
    +-- SyntaxWarning
    +-- OverflowWarning
    +-- RuntimeWarning
```

This:

```
try:  
    something  
except ArithmeticError:  
    pass
```

catches all 3 arithmetic errors:

`OverflowError`, `ZeroDivisionError`, and `FloatingPointError`.

The `help('exceptions')` also shows this about each Exception class and subclass:

```
class ArithmeticError(StandardError)  
| Base class for arithmetic errors.  
|  
| Method resolution order:  
|     ArithmeticError  
|     StandardError  
|     Exception  
|  
| Methods inherited from Exception:  
|  
|     __getitem__(...)  
|  
|     __init__(...)  
|  
|     __str__(...)
```

When you collect an exception:

```
try:  
    something  
except ValueError as info:  
    print(info)
```

The `info` is actually an instance of the `ValueError` class, or which ever exception type happened. When there is this code:

```
print(info)
```

calls `str(info)`, as it always does, and the `Exception` class's `__str__` gets called.

So, you cannot:

```
print("This happened: " + info)
```

but you can:

```
print(f"This happened: {info}")
```

or

```
print("This happened: " + str(info))
```

The syntax for catching multiple exceptions can be:

```
try:  
    something  
except ExceptOne as exception_object:  
    pass  
except ExceptTwo as exception_object:  
    pass
```

-or-

```
try:  
    something  
except (ExceptOne, ExceptTwo) as exception_object:  
    pass
```

You don't have to collect the exception_object:

```
try:  
    something  
except ExceptOne:  
    pass
```

You can add a generic `except Exception` at the end to collect all the exceptions that you didn't specifically name, or specifically name their parents in the Exception hierarchy, but be careful to track those errors so you can fix them. Don't lose control of your code! `else`

You can always add an `else`. The `else` clause will happen when no exceptions were raised:

```
try:  
    something  
except (ExceptOne, ExceptTwo) as exception_object:  
    pass  
else:  
    something_else
```

If you are running Python 2.5 or later, you can add a `finally`, which will absolutely happen, if the `try` block succeeds or if it fails, or even if it contains a `return` or `sys.exit()`.

```
try:  
    something  
except (ExceptOne, ExceptTwo) as exception_object:  
    pass  
else:  
    something_else  
finally:  
    something_that_absolutely_will_happen
```

However, for earlier Python versions, you are allowed either 'except' or 'finally', not both with one 'try'. So you need to use this form:

```
try:  
    try:  
        something  
    finally:  
        something_that_absolutely_will_happen  
except (ExceptOne, ExceptTwo) as exception_object:  
    pass  
else:  
    something_else
```

for the same effect.

```
assert_.py  
1 #!/usr/bin/env python3  
2 """The "assert" statement is useful while debugging. It goes  
3 away under any optimization."""  
4  
5 def main():  
6     number = float(input("Give me a positive number: "))  
7  
8     assert number > 0, f"Input is not positive: {number}"  
9  
10    print(f"Good. {number} is positive.")  
11  
12 if __name__ == "__main__":  
13     main()  
14  
  
$ assert_.py  
Give me a positive number: 3.14  
Good. 3.14 is positive.  
$ assert_.py  
Give me a positive number: 0  
Traceback (most recent call last):  
  File "./assert_.py", line 13, in <module>  
    main()  
  File "./assert_.py", line 8, in main  
    assert number > 0, f"Input is not positive: {number}"  
AssertionError: Input is not positive: 0.0  
$
```

That was a bad idea!

assert statements are left out of your code when development is over, and any optimization is added. They are intended for a the developer, as a *sanity check*.

If you want to cause a traceback that will endure after optimization, then you want to *raise* an exception.

raise1.py

```
1 #!/usr/bin/env python3
2 """You can raise an exception anytime you take a notion."""
3
4 def GetPositiveNumber(prompt):
5     """Writes the prompt to stdout and collects the response.
6     Returns an int > 0 if one was given, ValueError otherwise.
7     """
8     said = input(prompt)
9     number = float(said)
10    if number > 0:
11        return number
12    raise ValueError("Number given must be positive.")
13
14 if __name__ == "__main__":
15     print(GetPositiveNumber("Positive number: "))
16
```

```
$ raise1.py
Positive number: -1
Traceback (most recent call last):
  File "./raise1.py", line 15, in <module>
    print(GetPositiveNumber("Positive number: "))
  File "./raise1.py", line 12, in GetPositiveNumber
    raise ValueError("Number given must be positive.")
ValueError: Number given must be positive.
```

```
$
```

©Marilyn Davis 2007-2020

raise2.py

```
1 #!/usr/bin/env python3
2 """And, you can re-raise an exception."""
3
4 import raise1
5
6 def main():
7     try:
8         number = raise1.GetPositiveNumber("Positive number: ")
9     except ValueError:
10        print("That was wrong!")
11        raise      # Raises last exception again
12
13 main()
```

```
$ raise2.py
Positive number: -2
That was wrong!
Traceback (most recent call last):
  File "./raise2.py", line 13, in <module>
    main()
  File "./raise2.py", line 8, in main
    number = raise1.GetPositiveNumber("Positive number: ")
  File "/Users/marilyn/Python/MM3/Labs/Lab21_Wrap_Up/raise1.py", line 12, in GetPositiveNumb
er
    raise ValueError("Number given must be positive.")
ValueError: Number given must be positive.
$
```

raise3.py

```
1 #!/usr/bin/env python3
2 """The argument you give to your raise can be anything,
3 a string is most common, but a sequence is possible."""
4
5
6 def GetPositiveNumber(prompt):
7     """Writes the prompt to stdout and collects the response.
8     Returns an int > 0 if one was given, ValueError otherwise,
9     giving back the non-positive number, if that was the
10    problem.
11    """
12    said = input(prompt)
13    number = float(said)
14    if number > 0:
15        return number
16    raise ValueError("Number given must be positive.", number)
17
18 def main():
19     try:
20         print(GetPositiveNumber("Positive number: "))
21     except ValueError as exception_object:
22         print(exception_object)
23
24 main()
25
```

```
$ raise3.py
Positive number:-1
('Number given must be positive.', -1.0)
$
```

exception_info.py

```
1 #!/usr/bin/env python3
2 """
3 Get info about your caught exception from sys and traceback
4 modules."""
5
6 import sys
7 import traceback
8
9 def Fun():
10     raise ArithmeticError("Fake message.")
11
12 if __name__ == "__main__":
13     try:
14         Fun()
15     except:
16         print("sys.exc_type =", sys.exc_info()[0])
17         print("sys.exc_value =", sys.exc_info()[1])
18         print("---")
19         print(traceback.format_exc(), end=' ')
20         print("---")
21
```

```
$ exception_info.py
sys.exc_type = <class 'ArithmeticError'>
sys.exc_value = Fake message.
---
Traceback (most recent call last):
  File "./exception_info.py", line 14, in <module>
    Fun()
  File "./exception_info.py", line 10, in Fun
    raise ArithmeticError("Fake message.")
ArithmetiError: Fake message.
---
$
```

myexcept1.py

```
1 #!/usr/bin/env python3
2 """You can invent your own exception, having it inherit from
3 some class in the Exception hierarchy."""
4
5 class NonPositiveNumber(ArithmetError):
6     """Defines a new exception class with the same behavior
7     as any builtin ArithmetError."""
8     pass
9
10 def GetPositiveNumber(prompt):
11     """Writes the prompt to stdout and collects the response.
12     Returns an int > 0 if one was given, raising:
13         ValueError - if a number wasn't given
14         NonPositiveNumber - if the number <= 0
15             and reporting the non-positive number, if that was
16             the problem.
17 """
18     said = input(prompt)
19     number = float(said)
20     if number > 0:
21         return number
22     raise NonPositiveNumber("Number given must be positive.",
23                             number)
24 def main():
25     for trial in range(3):
26         try:
27             print(GetPositiveNumber("Number please: "))
28         except NonPositiveNumber as exception_obj:
29             print("Caught NonPositiveNumber:", exception_obj)
30         except ValueError as exception_obj:
31             print("Caught ValueError:", exception_obj)
32
33 if __name__ == "__main__":
34     main()
```

```
$ myexcept1.py
Number please: -2
Caught NonPositiveNumber: ('Number given must be positive.', -2.0)
Number please: x
Caught ValueError: could not convert string to float: 'x'
Number please: 1
1.0
$
```

myexcept2.py

```
1 #!/usr/bin/env python3
2 """You can override and extend the functionality."""
3
4 class NonPositiveNumber(ArithmeticError):
5
6     times = 0
7
8     def __init__(self, *args):
9         """We call to the base class initializer
10        and add some functionality."""
11
12         ArithmeticError.__init__(self, *args)
13         NonPositiveNumber.times += 1
14
15     def __str__():
16         return f"You messed {NonPositiveNumber.times} {'time' if NonPositiveNumber.times==1 else 'times'}! {self.args}"
17
18 def GetPositiveNumber(prompt):
19     """Writes the prompt to stdout and collects the response.
20     Returns an int > 0 if one was given, raising:
21         ValueError - if a number wasn't given
22         NonPositiveNumber - if the number <= 0
23             and giving back the non-positive number, if that was
24             the problem.
25 """
26     said = input(prompt)
27     number = float(said)
28     if number > 0:
29         return number
30     raise NonPositiveNumber("Value not positive.", number)
31
32 def main():
33     for trial in range(5):
34         try:
35             print(GetPositiveNumber("Number please: "))
36         except NonPositiveNumber as exception_obj:
37             print("Caught NonPositiveNumber:", exception_obj)
38         except ValueError as exception_obj:
39             print("Caught ValueError:", exception_obj)
40
41 if __name__ == "__main__":
42     main()
43
```

```
$ myexcept2.py
Number please: -1
Caught NonPositiveNumber: You messed 1 time! ('Value not positive.', -1.0)
Number please: 1
1.0
Number please: x
Caught ValueError: could not convert string to float: 'x'
Number please: -1
Caught NonPositiveNumber: You messed 2 times! ('Value not positive.', -1.0)
Number please: -2
Caught NonPositiveNumber: You messed 3 times! ('Value not positive.', -2.0)
$
```

©Marilyn Davis, 2007-2020

manynames.py

```
1 #!/usr/bin/env python3
2 """manynames.py -- adapted from "Learning Python" by Mark Lutz
3 and David Ascher, published by O'Reilly. Demonstrates name-spaces
4 associated with classes, functions, and methods."""
5
6 x = 11                      # Module (global) name/attribute
7
8 class C:
9     x = 22                      # Class attribute
10    def M(self):
11        print(x)                # Sees the global x
12        self.x = 44              # instance attribute
13
14 def F():
15     x = 55                      # Local identifier in function
16
17 def G():
18     print(x)                  # Access module x (11)
19
20 if __name__ == "__main__":
21     obj = C()
22     obj.M()                    # 11: sees the global x
23     print(obj.x)               # 44: instance
24     print(C.x)                 # 22: class (obj.x if no x in instance)
25     print(x)                   # 11: module (manynames.x outside file)
26     G()                       # 11: sees the global x
27     try:
28         print(F.x)            # fails: only visible in function
29     except AttributeError as info:
30         print("F.x failed:", info)
```

```
$ manynames.py
11
44
22
11
11
F.x failed: 'function' object has no attribute 'x'
$
```

function_nest.py

```
1 #!/usr/bin/env python3
2 """function_nest.py Adapted from "Learning Python"
3 by Mark Lutz & Davis Ascher"""
4
5 x = "global x"
6 y = "global y"
7
8 def F1():
9     def F2():
10         y = "F2's y"
11         def F3():
12             print("F3 sees this x =", x)
13             print("F3 sees this y =", y)
14         F3()
15     F2()
16     print("F1 sees this x =", x)
17
18 if __name__ == "__main__":
19     F1()
```

```
$ python3 -i function_nest.py
F3 sees this x = global x
F3 sees this y = F2's y
F1 sees this x = global x
>>> F1.F2.y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'function' object has no attribute 'F2'
>>>
```

class_nest.py

```
1 #!/usr/bin/env python3
2 """class_nest.py class nesting scopes in the other
3 direction."""
4 w = 10
5 class C1:
6     x = 99
7     class C2:
8         y = 100
9         class C3:
10            z = 101      # <-- Visible from outside
11            print(w)
12            print(z)
13            # print C1.x    <-- Can't see outer classes
14            # print y        or any outer identifiers
15            # during first read.
16 if __name__ == "__main__":
17     print("About to initialize:")
18     c1 = C1()
```

```
$ python3 -i class_nest.py
10   # ----- This output happened when the class was read into
101  # ----- the compiler, not when an instance was instantiated.
About to initialize:
>>> list(name for name in dir(c1) if not name.startswith("__"))
['C2', 'x']
>>> c1.C2.C3.z
101
>>> c1.C2.C3.z = ":~"
>>> c1.C2.C3.z
':~'
>>>
```

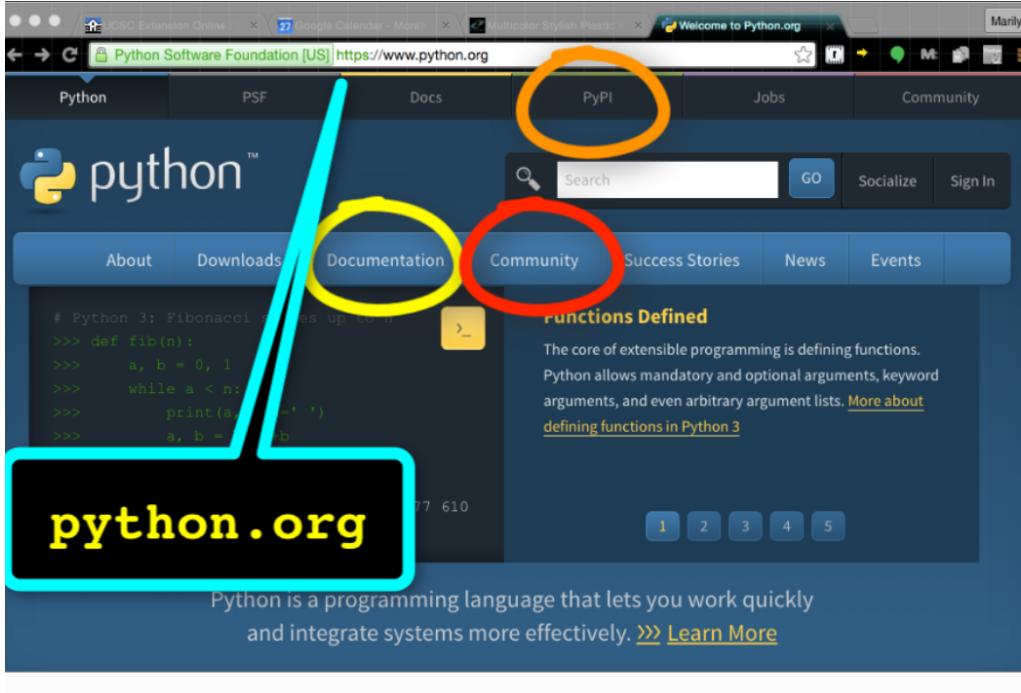
Finding Modules

- To find all the modules in your computer:

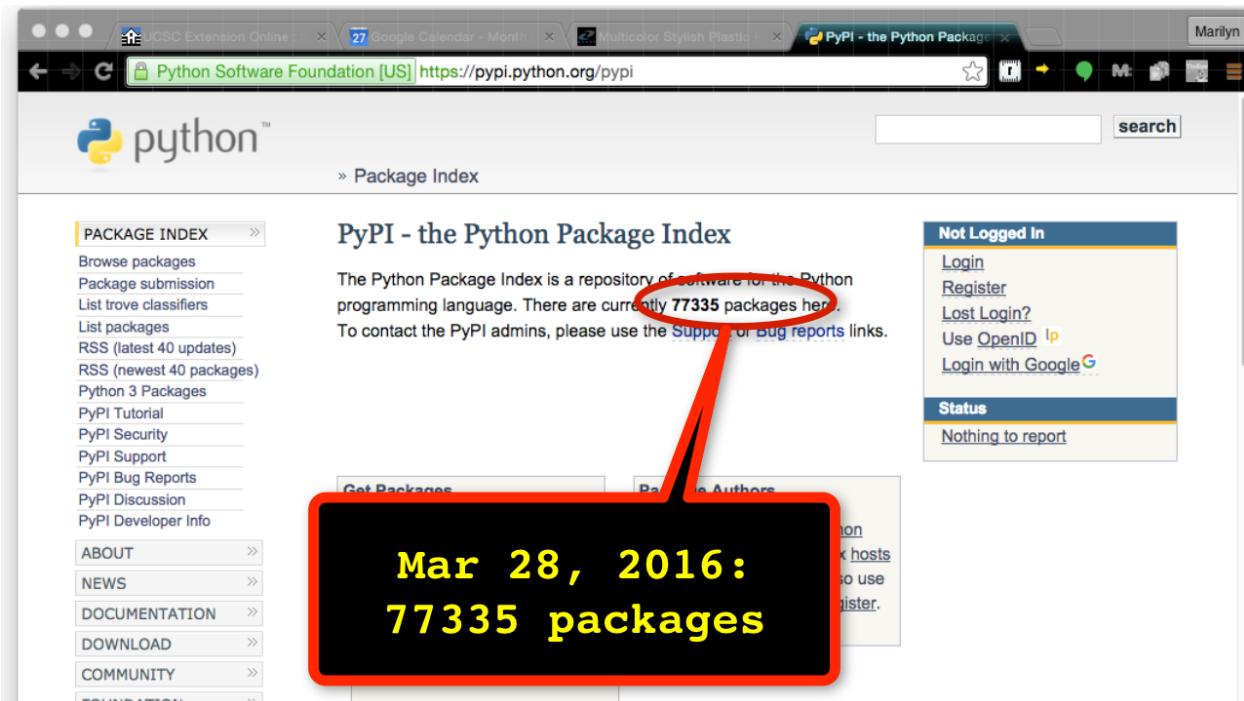
```
>>> help('modules')
Please wait a moment while I gather a list of all available modules...
```

```
BaseHTTPServer      atexit          imp              shelve
Bastion            audiodev        imputil         shlex
[etc.]
```

- Your best link for all things Python:

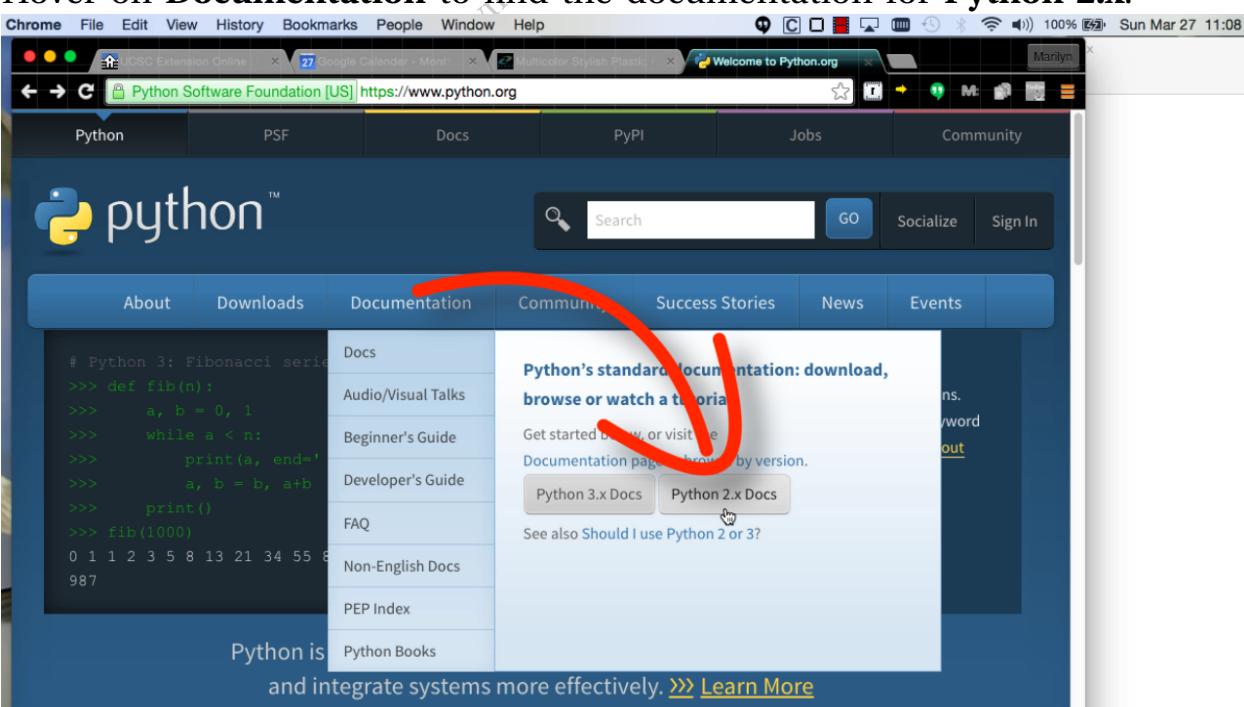


- To find many more libraries, you'll like the **PyPi** link off the **python.org** front page:

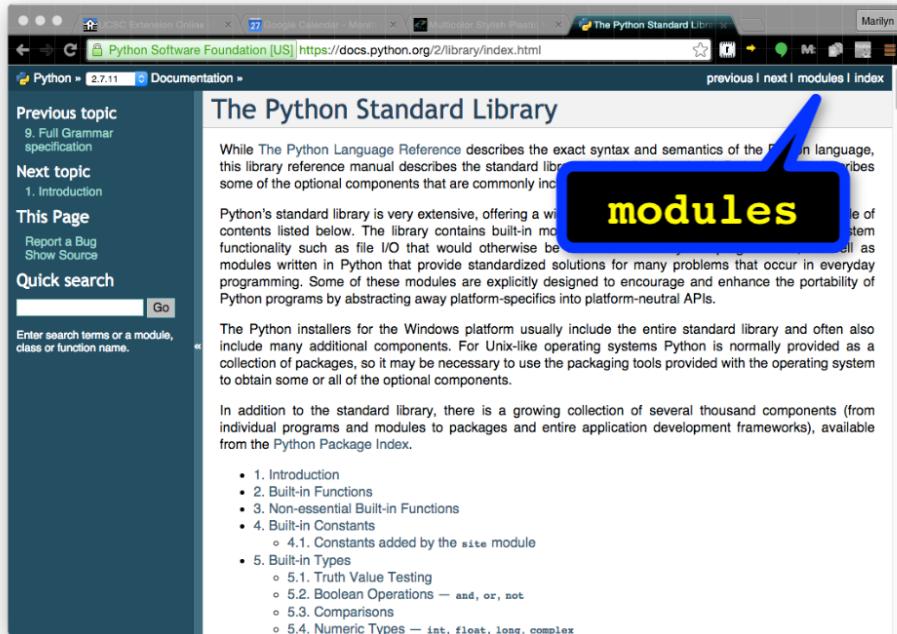


To get help:

- Hover on **Documentation** to find the documentation for Python 2.x.



- **Library Reference** is very useful, and correct:



- Notice the **modules** link near the upper right corner.

Finding Help

- Use Python's builtin `dir` and `help` facility:
- Get handy with internet searching for answers to your questions. Include the word *Python* with your search words.

- To get great human support, use the **Community** link on the front page to find the list of email lists maintained by python.org:

Tweets by @ThePSF

Python - The PSF @ThePSF One million children in the UK will start to receive their BBC micro:bits with @MicroPython today! pyfound.blogspot.com/2016/03/a-mill... 28 Mar

Python - The PSF @ThePSF A Million Children goo.gl/tb547vZ 22 Mar

Python - The PSF @ThePSF Postscript to Warehouse Post! goo.gl/fbxLNGSY 28 Jan

Python - The PSF @ThePSF Welcome to the Warehouse! goo.gl/fbzLENRE 28 Jan

Embed View on Twitter

Python >>> Community >>> Mailing Lists

Mailing Lists

Python Mailing Lists and Newsgroups

Here's an overview of the mail and news resources for python. A complete listing of python.org's public mailing lists is also available.

To request a new list, send e-mail to postmaster @ python.org. Please check first to make sure a similar list does not already exist.

list of lists

Mailing Lists python.org's public mailing lists document

comp.lang.python

comp.lang.python is a high-volume Usenet open (not moderated) newsgroup for general discussions and questions about Python. You can also access it as a mailing list through python-list.

Pretty much anything Python-related is fair game for discussion, and the group is even

- To get great help fast, join the **tutor** list where Python teachers in Europe will work patiently through the night, while you sleep, to answer your good question in the best and kindest way.

stdlib-sig	Python standard library development list
Sunpiggies	Python Users Group -- Flagstaff/Phoenix/Tucson, Arizona USA
TehPUG	Tehran Python User Group
Texas	Python community
Tkinter-discuss	tkinter builder
tkRAD-discuss	tkRAD builder
tkScenarist-help	tkScenarist builder
Tracker-discuss	tracker used for Python development
TriZPUG	Triangle (North Carolina) Python Users Group (formerly TriZPUG)
Tutor	Discussion for learning programming with Python
vml-papers-discuss	[no description available]
Web-SIG	SIG for Python support for the Web
Wheel-builders	Discussion of building and distributing Python packages, esp. extension packages, as
Winnipeg	Winnipeg Python Users Group
xml-sig	Python XML Special Interest Group



Lab 21 – Exercises:



Optional Reading

Read Guido's tutorial about exceptions. To find it, go to python.org, click Documentation on the left, and then *tutorial: 8. Errors and Exceptions*.

Exercises

1. Experience these brain-teasers in the lab so you're ready for work:

- (a) First think about this and make a prediction:

```
>>> a_list = ["loop"]
>>> a_list += a_list
>>> a_list
-----
>>> a_list.append(a_list)
>>> a_list
-----
```

Now, try it.

Python uses **...** to indicate an infinite pattern.

- (b) Here we have a global tea list (don't make global data!) three function definitions. Which, if any, of the functions will crash when called? Just take a guess:

```
tea = ['earl grey', 'camomile', 'chai']

def AppendTea():
    tea.append('green')

def AssignTea():
    global tea
    tea += ['blackberry']

def PlusAddTea():
    tea += ['peppermint']
```

(c) Try this:

```
>>> def Snake(rattle=[]):
...     rattle += ["hiss"]
...     print(rattle)
...
>>> Snake([100])
-----
>>> Snake()
-----
>>> Snake()
-----
>>> Snake()
```

2. Imagine you have a huge, and growing, dataset to process. You are going to get ready by using `data.txt`, a sample of the data. The sample data has two types of errors, missing values and entries that cannot be made into a float.

Write a function `CheckData(file_name, number_of_columns)`.

- Raise a `ValueError` when you find a line that doesn't have `number_of_columns` numbers.
- When there is a line of data that can't all be floats, a `ValueError` will raise.

In either case, the line number of the faulty line of data, the line of data itself, and the line in your code where the problem was discovered will be shown.

The idea is that, with each problem in the data, you need to look at the line of data and see if you want to alter it, or if you want to alter your code. For this exercise, just delete that line from the data, and run your program again.

`Labs/Lab21_Wrap_Up/data.txt` is your sample dataset. Also you have the same data in `Labs/Lab21_Wrap_Up/bad_data.txt` in case you want to renew `data.txt` easily.

My output:

```
$ verbbflab21_2.py
Renew the data? y
Traceback (most recent call last):
  File "./lab21_2.py", line 30, in <module>
    main()
  File "./lab21_2.py", line 27, in main
    print(CheckData(d_file, number_across))
  File "./lab21_2.py", line 14, in CheckData
    has number_of_numbers elements."")
ValueError: data.txt line # 957: -74.7      -0.6      21.8          -52.5      34.2      -78.9      -43.6
      has 7 elements.

$lab21_2.py
Renew the data? n
data.txt, line # 978:  83.6      62.7      -7.5      -5.9      83.9      -84.8      --8.6      29.2
      is not all floats.
Traceback (most recent call last):
  File "./lab21_2.py", line 30, in <module>
    main()
  File "./lab21_2.py", line 27, in main
```

```
    print(CheckData(d_file, number_across))
File "./lab21_2.py", line 16, in CheckData
    [float(n) for n in number_list]
File "./lab21_2.py", line 16, in <listcomp>
    [float(n) for n in number_list]
ValueError: could not convert string to float: '--8.6'
$ lab21_2.py
Renew the data? n
data.txt has 998 lines and 7984 numbers.
$
```

3. Invent a new exception type: `BadData` that inherits from `ValueError`. Redo the last exercise and raise your `BadData` instead of `ValueError`.
4. (Optional) Write a function, `Get_XY(prompt)`, that prompts the user and returns a tuple of two floats. If the user answers some form of quit or just hits the enter key, or Ctrl-D or Ctrl-C Ctrl-C, the function returns `None`.

All errors are explained and the user is asked to *Please try again.*

Use your function to collect two sides of a right triangle and give the hypotenuse. Reminder:

$$x^2 + y^2 = \text{hypotenuse}^2$$

You'll need to import `math`.

©Marilyn Davis, 2007-2020

1. Brainteasers:

(a) Would anyone predict this?

```
>>> a_list = ["loop"]
>>> a_list += a_list
>>> a_list
['loop', 'loop']
>>> a_list.append(a_list)
>>> a_list
['loop', 'loop', [...]]
```

So `a_list` has itself in it, that `a_list` inside has `a_list` inside, has `a_list` inside, ... Python uses `...` to indicate this infinite pattern.

I can't explain the difference between `append` and `+=`.

(b) Did you expect this?

list_scopes.py

```
1 #!/usr/bin/env python3
2 """list_scopes.py - answers lab21_1.b Scope issue with lists.
3 """
4 tea = ["earl grey", "camomile", "chai"]
5
6 def AppendTea():
7     tea.append("green") # <- Good because we don't assign
8                         #       (bind) the name, we use the
9                         #       function of an existing name.
10 def AssignTea():
11     global tea          # <- But for assignment, we need
12     tea += ["blackberry"] #       global because it will try
13                         #       to bind the name to the
14 def PlusAddTea():
15     tea += ["peppermint"] # <- No good
16
17 def main():
18     AppendTea()
19     print(tea)
20     AssignTea()
21     print(tea)
22     PlusAddTea()
23 if __name__ == "__main__":
24     main()
```

```
$ list_scopes.py
['earl grey', 'camomile', 'chai', 'green']
['earl grey', 'camomile', 'chai', 'green', 'blackberry']
Traceback (most recent call last):
  File "./list_scopes.py", line 24, in <module>
    main()
  File "./list_scopes.py", line 22, in main
    PlusAddTea()
  File "./list_scopes.py", line 15, in PlusAddTea
    tea += ['peppermint'] # <- No good
UnboundLocalError: local variable 'tea' referenced before assignment
```

PlusAddTea will crash! tea.append is a reference to tea and the tea object does the appending.

- (c) This is an important surprise:

```
>>> def Snake(rattle=[]):
...     rattle += ["hiss"]
...     print(rattle)
...
>>> Snake([100])
[100, 'hiss']
```

No surprises there but:

```
>>> Snake()
['hiss']
```

As expected.

```
>>> Snake()
['hiss', 'hiss']
>>> Snake()
['hiss', 'hiss', 'hiss']
```

Woah, since it used the mutable default in the last call, and then it altered it, it altered that default!

lab21_2.py

```
1 #!/usr/bin/env python3
2 """Reading data and pointing out the data flaws.
3 """
4
5 import shutil
6
7 def CheckData(d_file, number_across):
8     with open(d_file) as f_obj:
9         for i, line in enumerate(f_obj, 1):
10             number_list = line.split()
11             number_of_numbers = len(number_list)
12             if number_of_numbers != number_across:
13                 raise ValueError(f"""{d_file} line #{i:4}: {line}
14 has {number_of_numbers} elements.""")
15             try:
16                 [float(n) for n in number_list]
17             except ValueError:
18                 print(f"""{d_file}, line #{i:4}: {line} is not all
19 floats""")
20             raise
21
22     return f"{d_file} has {i} lines and {number_across * i} number
21 s."
23
24 def main(d_file="data.txt", number_across=8):
25     yn = input("Renew the data? ")
26     if yn and yn[0] in "yY":
27         shutil.copy("bad_data.txt", "data.txt")
28         print(CheckData(d_file, number_across))
29
30 if __name__ == "__main__":
31     main()
```

Output shown in specification.

lab21_3.py

```
1 #!/usr/bin/env python3
2 """Reading data and pointing out the data flaws.
3 """
4 import shutil
5
6 class BadData(ValueError):
7     pass
8
9 def CheckData(d_file, number_across):
10    with open(d_file) as f_obj:
11        for i, line in enumerate(f_obj, 1):
12            number_list = line.split()
13            number_of_numbers = len(number_list)
14            if number_of_numbers != number_across:
15                raise BadData(f"""{d_file} line #{i:4}: {line}
16 has {number_of_numbers} elements""")
17            try:
18                [float(n) for n in number_list]
19            except ValueError:
20                raise BadData(f"""{d_file}, line #{i:4}: {line}
21 is not all floats""")
22            raise
23
24    return f"{d_file} has {i} lines and {number_across * i} number
24 s."
25
26 def main(d_file="data.txt", number_across=8):
27    yn = input("Renew the data? ")
28    if yn and yn[0] in "yY":
29        shutil.copy("bad_data.txt", "data.txt")
30    print(CheckData(d_file, number_across))
31
32 if __name__ == "__main__":
33    main()
34
```

Output is similar to the last exercise.

lab21_4.py

```
1 #!/usr/bin/env python3
2 """lab21_4.py (Optional)-- collects 2 sides of a right
3 triangle and prints the hypotenuse."""
4
5 import math
6
7 def GetXY(prompt):
8     """Returns a tuple of 2 floats, or None if the user just
9     hits the enter key or enters q."""
10
11    while True:
12        try:
13            incoming = input(prompt)
14        except (KeyboardInterrupt, EOFError):
15            return None
16        if incoming == '' or incoming[0].lower() == 'q':
17            return None
18        try:
19            x, y = [float(x) for x in incoming.split(',')]
20        except (TypeError, NameError, ValueError, SyntaxError):
21            print("Two numbers are required.")
22        else:
23            return x, y
24
25 def Hypot(x, y):
26     """Returns sqrt(x**2 + y**2)"""
27     return math.hypot(x, y) # or sqrt(x * x + y * y)
28
29 def main():
30     while True:
31         xy = (GetXY("Give me 2 sides of a right triangle:"
32                     " (x, y)"))
33         if not xy:
34             print()
35             break
36         answer = Hypot(*xy)
37         print(f"Hypotenuse is {answer:.2f}")
38
39 if __name__ == "__main__":
40     main()
```

```
$ lab21_3.py
Give me 2 sides of a right triangle: (x, y) 4, 5
Hypotenuse is 6.40
Give me 2 sides of a right triangle: (x, y) a
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1,a
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1,2,3
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 2,,3
Two numbers are required.
Give me 2 sides of a right triangle: (x, y) 1,2
Hypotenuse is 2.24
Give me 2 sides of a right triangle: (x, y) X
Two numbers are required.
Give me 2 sides of a right triangle: (x, y)
$
```

©Marilyn Davis, 2007-2020

Index

%
 formatting strings, 193
% operator, 19
 arithmetic, 19
 dictionary replacement, 193
 formatting strings, 18, 193

* unpacking sequences, 49

/, // operators, 27

== operator, 236

@ decorator, 199
@classmethod, 269
@staticmethod, 269

bytes, 24, 180
chr, 24
ord, 24
seek, 126
stderr, 127
stdout, 127
str, 24
tell, 126

_ single underscore prepend, 149
__all__, 150
__future__, 27
__name__ == "__main__", 88-92

attribute control, 265-266
 __getattr__ and __setattr__, 266

boolians, 18
break, 5
builtin classes
 extending, 259

call-back function, 80
class
 iterator, 241
 extending list, 261
 attribute control, 265-266
 attributes, 212
 calling superclass, 219, 262, 271
 class attribute, 268

class method, 269
container, 215
containment, 214-215
extending builtins, 259
inheritance, 217-220, 233, 239, 264
initialization, 213, 249
method, *see* function
nested, 313
overriding
 __str__, 237
privacy, pseudo, 244
providing
 __call__ method, 239
 __getattr__, 266
 __getitem__, 241, 261
 __init__ method, 213
 __setattr__, 266
 self, 211
 static method, 269
 super, 262
 syntax, 211-220, 233
command line, 127
comment, 1
comprehension
 dictionary, 113
 list, 98
conditional, 5
container class, 215
containment, 214-215
context handler, 119, 279
context manager
 class, 284-285
copy
 copy module, 161
 deepcopy, 161
 file, 131
 independent
 dict, 161
 cProfile module, 181

declaration
 global, 55
 nonlocal, 152
decorator, 199
deepcopy, 161

dict, 109
 copying, 161
 iterating, 109, 115
 string replacement, 193
 dictionary, *see* dict
 comprehension, 113
 dir, 57
 division issue, 27
 dynamic code generation, 177-179, 183, 187

 else, 5
 attached to a loop, 1
 if/elif, 5
 try/except block, 17
 enumerate, 99
 eval, 177, 183
 eval with repr, 65
 exceptions, 282, **300-303**
 assert, 303
 finally, 302
 generic, 302
 handling, 17, 281-121, 283, 307
 hierarchy, 300
 inventing your own, 308-309
 multiple, 302
 raise-ing them yourself, 280, 304-306
 sys information, 307
 exceptions module, 280
 exec, 177, 183

 f-strings, 18-23
 False, 18
 file, 119-126, 279-285
 copy, 131
 notes, 125
 file
 seek, 126
 tell, 126
 filter, 99, 100
 finally, 281-283, 302
 before Python 2.5, 281-282
 since Python 2.5, 121, 283
 for, 7
 formatted strings, 18-27, **26**, 193
 from, 147-150
 import *, 147-150
 function
 nesting, 312
 protocols, 45-48
 default arguments, 38, 322
 keyword arguments, 38
 functional programming, 99
 functions
 nesting, 86

 generator, 197
 getattr, 179, 187
 global, 321
 global declaration, 55

 has-a class relationship, 215
 help, 57

 identifier
 visibility, 55, 152
 Idle, 3
 if/elif/else, 5
 import, 56
 *, 147-150
 as, 165
 from, 147-150
 from any directory, 164
 with dots (.), 164
 your own code, 88
 in
 for loop, 7
 testing membership, 56, 64
 indentation, 1
 inheritance, 217-220, 233
 from a different module, 239
 multiple, 233, 239
 diamond, 264
 input, 15-18
 input
 input, 15
 no response, 19
 user, 15
 integer division, 19
 integrated development environments, 3
 introspection, 57
 dir, 57
 from command line, 1
 help, 57
 is operator, 236
 isinstance, 236
 issubclass, 236
 iter, 198
 iterating, 241
 dict, 109, 115
 extending list, 261
 sequence, 9, 64

 key option of sorts, 80

 lambda, 99
 list, 62
 augmented assignment, 62
 comprehensions, 98
 concatenation, 62
 empty, 63

iterating, 64
repetition, 64
scope, 321
singleton, 63
slicing, 62
testing membership, 64
logger
 context handler, 297
logical operators, 7
loops
 break/else, 5
 for, 7
 while, 5

mangling, name, 244
map, 100
math module, 57
method, *see* function
methods
 overriding, 237
modulo, *see* % operator
multiple inheritance, 233
mutability, 50, 65, 147

name mangling, 244
namespaces, 55, 152, 265, 311-313
nesting functions, 86
nonlocal declaration, 152

open, 119-126, 122, 279-285
 notes, 125
 unicode, 123
operators, 7
 logical, 7
 modulo, *see* % operator
 relational, 7
optparse module, 288
os module, 136
 os.walk, 138
overriding, *see* class, overriding

packages, 164-166
piping processes, 180
pitfalls, 318
portability, 136
print, 4
 always a space, 15
 multiple objects, 15
 new line, 1
privacy, psuedo, 244
Pythonic thinking, 54

quotes, 4

raise, 282

 raise exceptions, 280
 random module, 56, 57
 recursing a directory, 138
 redirect print output, 127
 relational operators, 7
 repr, 65
 running a Python program, 1-3

 scope, 311-313
 list, 321
 self, 211
 sequence, 9, 56, 62-50
 augmented assignment, 62
 concatenation, 62
 empty object, 63
 iterating, 64
 repetition, 64
 singleton object, 63
 slicing, 62
 backwards, 62
 testing membership, 64
 set, 151-152
 setattr, 179, 187
 shutil module, 131
 slicing, 62
 backwards, 62
 sort/sorted, 71, 80
 key option, 80
 stdin/stdout/stderr, 127
 str, 62
 augmented assignment, 62
 concatenation, 15, 62
 delimiting, 4
 empty, 63
 iterating, 64
 overriding, 237
 repetition, 64
 singleton, 63
 slicing, 62
 testing membership, 64
 str.format, 18
 string, *see* str
 strings
 triple-quotted strings, 4
 subprocess module, 180
 sys module, 127
 sys.path, 164

 tempfile module, 132
 time module, 138
 traceback, 307
 triple-quotted strings, 4
 True, 18, 56
 try/except block
 else, 17

finally, 121, 281-283
tt next, 198
tuple, 9, 62
 augmented assignment, 62
 concatenation, 62
 empty, 63
 iterating, 9
 repetition, 64
 singleton, 63
 testing membership, 64

UML, 243
unicode, 24, 123
Unified Modeling Language, 215
unittest module, 286
unpacking sequences, 45
unwrapping
 swapping, 50

visibility
 identifier, 55, 152

walking a directory, 138
while loop, 18
with keyword, 119

yield, 197

zip, 99

©Marilyn Davis, 2007-2020