CptS 111, Fall 2023
Lect. #19, Mar. 29, 2023
Class Notes

Today's Agenda:

1. Using `with`
2. Modules

Ch. 7 (cont.)

1. Using the `with` Statement

We've learned how to open and close files and also how to read from them and write or print to them. Next, we add a bit of sophistication using the `with` statement. The beauty of using `with` is that it automatically closes a file so that you don't have to.

The template for a `with` statement is:

```
with open('<file>', 'r' or 'w' or 'a') as <file_name>:
    <with_body>
```

After the code inside the `with` body is completed, Python will close any open files.

Let's consider an example.

In [1]:
```
# Using the with statement

with open(input('Enter input file name: '), 'r') as file_in:
    for line in file_in:
        print(line, end='')
```

```
Enter input file name: quote.txt

"Many people are desperately looking for some wise advice which will
recommend that they do what they want to do."
--Unknown
```

Note that we can nest `input()` within the `open()` function.

As mentioned, after the code within the `with` body has been executed, Python closes the file. We can show this by running the same code again (which we couldn't do if the file hadn't been closed).

In [2]:
```
# Using the with statement again to prove that the file was closed

with open(input('Enter input file name: '), 'r') as file_in:
    for line in file_in:
        print(line, end='')
```

```
Enter input file name: quote.txt

"Many people are desperately looking for some wise advice which will
recommend that they do what they want to do."
--Unknown
```

Of course, you can also use `with` to print to a file. Let's add something to our quote.

In [3]:
```
# Use the with statement to print to the quote.txt file and then reopen a

with open('quote.txt', 'a') as file_out:
    print('  Philosopher', file=file_out)

with open('quote.txt') as file_in:
    for line in file_in:
        print(line, end='')
```

```
"Many people are desperately looking for some wise advice which will
recommend that they do what they want to do."
--Unknown
  Philosopher
```

2. Modules

We learned a little bit about modules in Ch. 3, and the `tkinter` module was used in PA #4. Today we're going to cover modules a bit more in depth.

Recall that for specialized functions, we import modules. Also recall that a **module** is just code that's stored in a .py file for use in another module or in a Python **script**, i.e., a program we've written in, e.g., an IDLE Editor window. In fact, we can use scripts we've written ourselves as modules. Each module contains functions designed for a specific purpose.

Python comes with a set of standard modules that is quite extensive, but in addition, there are thousands and thousands of modules available that have been written by both professionals and enthusiasts. Some of the standard modules included with Python are:

```
math: math functions                    # Lab #2, Lab #10
cmath: complex math functions
statistics: statistics functions
random: random numbers                  # Lab #10
turtle: graphics
os: operating system
time: time access and conversions
mailbox: mailbox manipulation
calendar: calendar functions            # Lab #10
tkinter: uses tk tools to create GUIs   # PA #4
```

In addition, many open source modules available for use freely by anyone can be found at:

Python Package Index (PyPI) repository (https://pypi.org/)

We'll be using several of these, including `numpy` (Lab #12), `matplotlib` (Lab #12), and `pygame` (PA #7).

Let's consider the ways we can import modules.

A. Basic Import Statement

Recall (from Ch. 3) that the simplest import statement requires the use of **dot notation** by which we mean that *when we use a function in the module, the name of the module must be given,*

In [4]:
```
# Consider the os module; use dot notation to access getcwd()

import os
os.getcwd()
```

Out[4]:
```
'/Users/shira/teaching/cs111/spr23/lectures'
```

In [5]:
```
# Use chdir() to change directories.

os.chdir('/users/shira/teaching/cs111/spr23/labs')
os.getcwd()
```

Out[5]:
```
'/Users/shira/teaching/cs111/spr23/labs'
```

In [6]:
```
# Note that some os functions are non-void

cwd = os.listdir()
print(cwd)
```

```
['lab1.pdf', 'lab3_t4.py', 'lab3.pdf', 'lab2.pdf', 'lab2_t1.txt', 'lab9
_t3.py', 'lab6.pdf', 'lab7.pdf', 'lab5_t4.py', 'lab6_t3a.py', 'lab7_t4.
py', 'lab5.pdf', 'lab4.pdf', 'lab4_t1.txt', 'lab4.tex', 'lab8_t1.txt',
'lab5.tex', 'lab4_t2.txt', 'lab7.tex', 'lab8_t2.txt', 'lab2_t4.py', 'la
b6.tex', 'lab2.tex', 'poem.txt', 'lab6_t4.py', 'lab3.tex', 'lab1.tex',
'lab8_t3.py', 'lab6_t1.txt', 'lab4_t4.py', 'lab3_t1.txt', 'lab1_t4.tx
t', 'lab4_t3.py', 'lab3_t2.txt', 'lab2_t2.py', 'lab6_t2.py', 'lab1_t3.t
xt', 'lab2_t3.py', 'slice3.eps', 'lab8.tex', 'lab9.tex', 'lab7_t3.py',
'lab9.pdf', 'lab8.pdf', 'lab5_t2.txt', 'lab9_t2.txt', 'lab6_t3b.py', 'l
ab5_t1.txt', 'lab9_t1.txt', 'genome.fna', 'lab5_t3.py', 'lab7_t1.txt',
'lab7_t2.py', 'lab3_t3.py']
```

B. Import Statement with Alias

We might want to use dot notation so we know that a function has been imported (and so that we don't have to worry about choosing names of functions that might be in a module), but we want to shorten the module name if we're going to use a lot of its functions to lessen the amount of typing required. Recall that we learned how to do this previously.

```
import <module> as <name>
```

Consider, for example, the `random` module and three of its functions:

```
randrange(): Generates a random number in a manner similar to the
             range() function; can have three arguments, and
             doesn't include the stop value
randint(): Generates a random integer including the start and sto
p
           values
random(): Generates a random number in the interval [0, 1) (')' m
eans doesn't include)
```

In [7]:
```
# Use an alias for a module; must use dot notation with alias

import random as r
print(dir(r))      # dir() gives contents of the random module
```

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONS
T', 'SystemRandom', 'TWOPI', '_ONE', '_Sequence', '_Set', '__all__', '_
_builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__nam
e__', '__package__', '__spec__', '_accumulate', '_acos', '_bisect', '_c
eil', '_cos', '_e', '_exp', '_floor', '_index', '_inst', '_isfinite',
'_log', '_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt',
'_test', '_test_generator', '_urandom', '_warn', 'betavariate', 'choic
e', 'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits',
'getstate', 'lognormvariate', 'normalvariate', 'paretovariate', 'randby
tes', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate',
'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariat
e']
```

In [8]:
```
# randrange() is similar in concept to the range() function; it never
# includes the stop value

for i in range(40):
    print(r.randrange(0, 10), end=' ')
```

```
8 9 0 5 8 2 5 7 5 4 5 6 4 6 6 0 3 0 9 0 5 4 7 8 3 6 0 0 9 6 3 9 7 8 7 3
4 6 4 6
```

In [9]:
```
# randint() includes the stop value; it can be used, e.g., to simulate
# rolling a single die

for i in range(40):
    print(r.randint(1, 6), end=' ')
print()
for i in range(40):
    print(r.randint(1, 6), end=' ')
```

```
6 3 4 4 2 2 4 4 1 3 1 3 3 6 1 3 6 3 1 4 3 3 4 1 4 5 5 3 2 2 2 4 2 2 2 1
2 5 5 6
5 3 3 4 3 5 1 4 5 5 3 6 3 3 3 4 6 4 4 5 5 3 5 3 5 3 6 2 6 4 5 2 5 5 5 6
2 1 5 4
```

In [10]:
```
# random() has no argument; generates a random fraction between 0 and 1,

for i in range(40):
    print(r.random(), end=' ')
```

```
0.32637068188596796 0.2533502710777319 0.054596312269493574 0.313167945
4563272 0.7142346113567271 0.04117931954844245 0.15807761536430476 0.85
2318014542321 0.7823545900093751 0.2451402289182647 0.5430090512509743
0.2580244335689569 0.2424685171562686 0.6380893017574043 0.215044048114
63917 0.9923206781007338 0.8251547756169457 0.08946277741558095 0.46688
78017650516 0.9903373616216117 0.9368469680965679 0.9520455586897936 0.
33228395639387565 0.12390763912704161 0.24666398711328374 0.96086822073
51931 0.6285783131781428 0.11785044993990268 0.12356705060091555 0.5109
38117090516 0.8443654255567075 0.16190911956061738 0.11625790341424003
0.518612856757441 0.7398343166951384 0.5908466875139393 0.4031745022206
772 0.32472242248640626 0.3403044241352542 0.8372261952957882
```

C. Import Statements Avoiding Dot Notation

Sometimes it's convenient to completely avoid the use of dot notation. We then just use the name of the function in the module.

```
from <module> import <func1>, <func2>
```

Special case:

```
from <module> import *
```

* = wildcard - use of this means import all functions

Let's consider, for example, the `math` module. We'll import several functions and the constant `pi`.

In [11]:
```
# Avoid dot notation by importing functions by name

from math import factorial, sqrt, cos, sin, pi

zero = sin(pi)
print('sin of pi =', zero)
print()
x = factorial(99)
print('factorial of 99 =', x)
print()
y = sqrt(x)
print('sqrt of factorial of 99 =', y)  # Recall sqrt() always returns a f
print()
z = cos(y)
print('cosine of sqrt of factorial of 99 =', z)
```

```
sin of pi = 1.2246467991473532e-16

factorial of 99 = 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
sqrt of factorial of 99 = 9.660549437994929e+77

cosine of sqrt of factorial of 99 = -0.5824628384306424
```

Notice that we didn't get zero for `sin(pi)` when we should have. There are a number of factors involved in this (including the finite precision of floats), but what's important to note is that we have to think a little about answers we get when they're very small. The same is true when we use our calculators.

Next, let's use the wildcard `*` to import all the functions in the `statistics` module.

In [12]:
```
# Avoid dot notation by importing everything using the wildcard *

from statistics import *

avg = mean([1, 2, 3, 4, 5, 15, 20])
med = median([1, 2, 3, 4, 5, 15, 20])
print(f'mean = {avg:.2f}, median = {med}')
```

```
mean = 7.14, median = 4
```

D. Import Module Functions with Aliases

Previously we mentioned that we can assign aliases to modules. We can also assign aliases to module functions.

```
from <module> import <func1> as <name1>, <func2> as <name2>, ...
```

For example:

In [13]:
```
# Avoid dot notation by importing functions and using aliases for them

from statistics import mean as avg, median as med
print('mean:', avg([1, 2, 3, 4, 5, 15, 20]))
print('median:', med([1, 2, 3, 4, 5, 15, 20]))
```

```
mean: 7.142857142857143
median: 4
```

Care must be taken when using modules without dot notation because we can easily wipe out or change functions or variables without realizing it.

In [14]:
```
# Note that avoiding dot notation can cause problems

from math import pi
print('pi from math module:', pi)
pi = 3.141
print('pi after reassigning pi:', pi)
print()
import math as m
print('m.pi from math module:', m.pi)
pi = 3.141
print('pi after assigning pi:', pi)
print('m.pi from math module:', m.pi)
```

```
pi from math module: 3.141592653589793
pi after reassigning pi: 3.141

m.pi from math module: 3.141592653589793
pi after assigning pi: 3.141
m.pi from math module: 3.141592653589793
```

What import statement should we use? It depends on several factors:

- How many of the functions are we going to use?
- How big is the module?
- How long is our program?

If a module is small or we're going to use a lot of its functions, then we probably want to import the entire module.

When should we use dot notation? This is really a matter of style. I prefer `import <module> as <name>` because then I don't have to worry about wiping out a function or variable or changing a value, but it also lets me know immediately that I'm using a function I imported.

Note that it's good programming practice to put all import statements at the beginning of a program.

E. Importing Modules We've Written

In [15]:
```
# We can import modules we've written
# This module simulates the results of rolling a pair of dices and showin
# the resulst in the form of a histogram.

import dice_rolls as dr
dr.main()
```

```
Enter number of rolls: 450

Dice roll histogram:
2s:  ***********
3s:  ************************
4s:  ********************************
5s:  ************************************************************
6s:  ********************************************************
7s:  ************************************************************************
*
8s:  ************************************************************************
***********
9s:  ****************************************
10s: ***********************************
11s: *******************************
12s: **************
Enter number of rolls: 475

Dice roll histogram:
2s:  ************
3s:  ************************
4s:  ******************************************
5s:  ***************************************
6s:  ************************************************************************
*
7s:  ************************************************************************
*********************************
8s:  ************************************************************
9s:  ********************************************************
10s: *********************************
11s: ****************************
12s: ***************
```

In [ ]:
```
# This program uses the turtle graphics module

import random_walk as rw
rw.main()                # Call main() to use the turtle module
```

```
Enter number of moves: 75

Enter number of moves:
```