Every Boilermaker Engineer Codes: 101
Entry-Level Programming in Python

Lecture 06

Dr. John H. Cole
<jhcole@purdue.edu>

PURDUE
UNIVERSITY®

COLLEGE OF ENGINEERING

Spring 2021

Part I

Working with Modules

About Modules

- What is a module?
  - a set of stored variables, functions, and data types
  - not built into the Python interpreter (e.g. not print or range)
  - modules must be imported to be used in your code
  - turtle is a module for simple 2d graphics
  - any .py file can be imported as a module
- Why use modules?
  - group related functions together
  - helps keep code organized
  - eases code reuse across different programs

Standard Library vs Third Party Modules

- Python standard library
  - included with Python by default
  - extensive set of solutions to common programming problems
  - examples: `random`, `string`, `math`
- third party modules
  - not included with Python by default (install with `pip`)
  - public repo at Python Package Index or PyPI (pypi.org)
  - over 250,000 projects available
  - examples: `numpy`, `matplotlib`

Basic Import

- syntax: `import <module_name>`
- module must be in the Python search path
- modules have attributes (either variables or functions)
- refer to module's attributes using dot notation (e.g. `module_name.attribute`)

Editor - spam.py

```
1 a = 'Spam'
2 def b():
3     print(a)
```

Terminal

```
$ python
>>> import spam
>>> spam.a
'Spam'
>>> spam.b()
Spam
```

Multiple Import

- can import as many modules as needed
- separate module names with commas or use separate lines
- module names separate attributes into namespaces

Terminal

```
$ python
>>> import spam, eggs
>>> spam.a
'Spam'
>>> eggs.a
'eggs'
```

Editor - eggs.py

```
1 a = 'eggs'
2 def b():
3     print(a)
```

Terminal

```
$ python
>>> import spam
>>> import eggs
>>> spam.a
'Spam'
>>> eggs.a
'eggs'
```

Nested Import

- imports can be nested

Editor - foo.py

```
1 import spam
2 import eggs
3 a = 'foo'
4 def b():
5     print(a)
```

Terminal

```
$ python
>>> import foo
>>> foo.a
'foo'
>>> foo.spam.a
'Spam'
>>> foo.eggs.a
'eggs'
```

Import as

- add the as keyword to use an alias instead of the module name

Editor - spam.py

```
1 a = 'Spam'
2 def b():
3     print(a)
```

Terminal

```
$ python
>>> import spam as s
>>> s.a
'Spam'
>>> s.b()
Spam
```

Multiple Import with as

- use separate lines
- or separate multiple modules with commas

Terminal

```
$ python
>>> import spam as s
>>> import eggs as e
>>> s.a
'Spam'
>>> e.a
'eggs'
```

Terminal

```
$ python
>>> import spam as s, eggs as e
>>> s.a
'Spam'
>>> e.a
'eggs'
>>>
```

Partial Import

- import part of a module with the `from` clause
- can use an `as` clause with a `from` clause
- do not use dot notation

Terminal

```
$ python
>>> from spam import b
>>> b()
Spam
>>>
```

Terminal

```
$ python
>>> from spam import b as c
>>> c()
Spam
>>>
```

Multiple Partial Import

- separate multiple parts with commas
- or use separate lines
- do not use dot notation

Terminal

```
$ python
>>> from spam import a, b
>>> a
'Spam'
>>> b()
Spam
>>>
```

Terminal

```
$ python
>>> from spam import a
>>> from spam import b
>>> a
'Spam'
>>> b()
Spam
```

Import Wildcard

- use * to import everything in the module
- do not use dot notation

Terminal

```
$ python
>>> from spam import *
>>> a
'Spam'
>>> b()
Spam
>>>
```

Import Namespace Conflicts

- last import wins

Terminal

```
$ python
>>> from spam import *
>>> from eggs import *
>>> a
'eggs'
>>> b()
eggs
>>>
```

Import Summary

Terminal

```
>>> import spam
>>> spam.b()
Spam
>>> import spam as s
>>> s.b()
Spam
```

Terminal

```
>>> from spam import b
>>> b()
Spam
>>> from spam import b as c
>>> c()
Spam
>>> from spam import *
>>> b()
Spam
```

Uses of Random Values

- games (e.g. lottery numbers, dice rolls, terrain generation)
- art (e.g. computer graphics)
- computer simulation (e.g. Monte Carlo, Markov chains)
- science (e.g. randomized controlled trials)
- cryptography (e.g. key generation, passwords)
- government/politics (e.g. jury selection, polling)

True Random Numbers vs Pseudo-Random Numbers

- true random numbers
  - generated via a physical process (thermal noise, quantum phenomena, etc.)
  - completely unpredictable
  - cryptographically secure
  - slow, only a few random bits per second
- pseudo-random numbers
  - generated via an algorithm
  - initialized with a seed value
  - the same seed always produces the same sequence
  - fast, gigabytes per second available

The random Module

- part of the Python standard library
- includes functions for working with random numbers
- uses a pseudo-random number generator
- not suitable for security or cryptographic
  - use `secrets` instead

random()

- take no arguments
- returns a floating point number
- the next number in a pseudo-random sequence
- value is between 0.0 and 1.0 (i.e. $\{x|[0,1)\}$)

Terminal

```
$ python
>>> import random as r
>>> r.random()
0.8062579873336703
>>> r.random()
0.1383028474695975 3
```

- for a random value in $[10, 15)$

Terminal

```
>>> 10 + (15-10)*r.random()
13.545871543686616
>>> 10 + (15-10)*r.random()
10.652981862833572
>>>
```

uniform(a, b)

- returns a floating point number between a and b
  - $\{x|[a,b); a \leq b\}$
  - $\{x|(b,a]; b < a\}$
- implements `a + (b-a)*random()`
- b might be included (depends on rounding)

Terminal

```
$ python
>>> import random as r
>>> r.uniform(10,15)
13.365979769810295
>>> r.uniform(17.5,13.5)
15.321537558939141
```

Terminal

```
>>> r.uniform(2.1,3.3)
2.6856635188009754
>>> r.uniform(2.1,3.3)
2.398291319290941
>>> r.uniform(2.1,3.3)
3.2195585990837
```

seed(a)

- uses OS provided random seed by default
  - produces a different sequence every time program runs
- setting the seed enables repeatable random sequences

Terminal

```
$ python
>>> import random as r
>>> r.seed(3)
>>> r.random()
0.2379646270918913 7
>>> r.random()
0.5442292252959519
>>> r.random()
0.3699551665480792 5
```

Terminal

```
$ python
>>> import random as r
>>> r.seed(3)
>>> r.random()
0.2379646270918913 7
>>> r.random()
0.5442292252959519
>>> r.random()
0.3699551665480792 5
```

randrange(start, stop, step)

- accepts one, two, or three positional arguments
  - arguments are the same as `range()` function
  - `start` set to 0 by default
  - `step` set to 1 by default
  - do not use keyword arguments
- returns random integer from `range(start, stop, step)`

Terminal

```
>>> import random as r
>>> r.randrange(2)
1
>>> r.randrange(2)
1
>>> r.randrange(2)
0
```

Terminal

```
>>> r.randrange(1,101)
98
>>> r.randrange(1,101)
6
>>> r.randrange(1,101)
49
>>>
```

More randrange() Examples

Select a random value from:
`[10, 20, 30, 40, 50]`

Terminal

```
>>> r.randrange(10,51,10)
30
>>> r.randrange(10,51,10)
50
>>> r.randrange(10,51,10)
20
>>> r.randrange(10,51,10)
10
>>> r.randrange(10,51,10)
30
```

Select a random value from:
`[10, 8, 6, 4]`

Terminal

```
>>> r.randrange(10,3,-2)
10
>>> r.randrange(10,3,-2)
8
>>> r.randrange(10,3,-2)
4
>>> r.randrange(10,3,-2)
6
>>> r.randrange(10,3,-2)
6
```

randint(a, b)

- returns an integer between a and b ($\{x|[a, b]; a \leq b\}$)
- a and b *are* included
- implements `randrange(a, b+1)`

Terminal

```
$ python
>>> import random as r
>>> r.randint(10,100)
85
>>> r.randint(10,100)
15
>>>
```

Terminal

```
>>> r.randint(1,2)
2
>>> r.randint(1,2)
1
>>> r.randint(1,2)
1
>>>
```

Exercise 1

Guess the result

Terminal

```
$ python
>>> from random import *
>>> random.randint(3,10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'builtin_function_or_method' object
has no attribute 'randint'
```

Exercise 2

Guess the result

Terminal

```
$ python
>>> from random import *
>>> randint(15,10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.6/random.py", line 221, in
  randint
    return self.randrange(a, b+1)
  File "/usr/lib/python3.6/random.py", line 199, in
  randrange
    raise ValueError("empty range for randrange()
  (%d,%d, %d)" % (istart, istop, width))
ValueError: empty range for randrange() (15,11, -4)
```

choice(x)

- chose an item at random from sequence x

Terminal

```
$ python
>>> import random as r
>>> r.choice(range(5))
4
>>> l = ['a', 'b', 'c']
>>> r.choice(l)
'b'
```

choices(x, k = 1)

- chose k items at random from sequence x with replacement
- use keyword argument to set k
- k can be larger than the size of x

Terminal

```
>>> import random as r
>>> l = ['a', 'b', 'c']
>>> r.choices(l)
['a']
>>> r.choices(l, k = 5)
['c', 'c', 'a', 'a', 'c']
```

sample(x, k)

- chose k items at random from sequence x without replacement
- k must be less than or equal to the size of x

Terminal

```
>>> import random as r
>>> r.sample(range(100000), 5)
[90645, 12641, 72407, 2372, 30685]
>>> l = list(range(5))
>>> r.sample(l, 5)
[4, 2, 1, 0, 3]
```

shuffle(x)

- shuffle the sequence x in place

Terminal

```
>>> import random as r
>>> l = [0, 1, 2, 3, 4]
>>> r.shuffle(l)
>>> l
[0, 4, 1, 2, 3]
```

The math Module

- part of the Python standard library
- includes functions for performing mathematical calculations
- does not work with complex numbers
  - use `cmath` instead

tau, pi AND e

defines constants τ, π, and e to available precision

tau $\tau = 6.283185...$

pi $\pi = 3.141592...$

e $e = 2.718281...$

Terminal

```
$ python
>>> from math import *
>>> tau
6.283185307179586
>>> pi
3.141592653589793
>>> e
2.718281828459045
```

- tau is two pi ($\tau = 2\pi$)

ceil(x) AND floor(x)

`ceil(x)` returns the smallest integer greater than or equal to x

`floor(x)` returns the largest integer smaller than or equal to x

Terminal

```
>>> from math import *
>>> ceil(8)
8
>>> tau
6.283185307179586
>>> ceil(tau)
7
>>> ceil(-tau)
-6
```

Terminal

```
>>> from math import *
>>> floor(8)
8
>>> tau
6.283185307179586
>>> floor(tau)
6
>>> floor(-tau)
-7
```

factorial(x)

`factorial(x)` returns $x!$

$$x! = \prod_{n=1}^{x} n = 1 \times 2 \times 3 \cdots (x-2) \times (x-1) \times x$$

$$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6$$

Terminal

```
>>> from math import *
>>> factorial(1)
1
>>> factorial(2)
2
>>> factorial(3)
6
>>> factorial(4)
24
```

Terminal

```
>>> from math import *
>>> factorial(5)
120
>>> factorial(6)
720
>>> factorial(7)
5040
>>> len(str(factorial(30000)))
121288
```

pow(x, y), AND sqrt(x)

`pow(x, y)` returns x^y

`sqrt(x)` returns $\sqrt{x}$ for $x \geq 0$

Terminal

```
>>> from math import *
>>> pow(2,4)
16.0
>>> pow(e,pi)
23.140692632779263
>>> sqrt(25)
5.0
>>> sqrt(-1)
Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
ValueError: math domain error
```

exp(x) AND log(x, base)

`exp(x)` returns e^x

- somewhat more accurate than `math.e**x` or `pow(math.e, x)`

`log(x)` returns $\ln(x)$

`log(x, base)` returns $\ln(x)/\ln(\text{base})$

Terminal

```
>>> from math import *
>>> exp(100)
2.6881171418161356e+43
>>> e**100
2.6881171418161212e+43
>>> pow(e, 100)
2.6881171418161212e+43
```

Terminal

```
>>> from math import *
>>> log(exp(100))
100.0
>>> log(10)
2.302585092994046
>>> log(100,10)
2.0
```

sin(x) AND cos(x)

`sin(x)` returns $\sin x$ for x in radians

`cos(x)` returns $\cos x$ for x in radians

Terminal

```
>>> from math import *
>>> sin(0)
0.0
>>> sin(pi/2)
1.0
>>> sin(tau/4)
1.0
```

Terminal

```
>>> from math import *
>>> cos(0)
1.0
>>> cos(pi/3)
0.5000000000000001
>>> cos(tau/6)
0.5000000000000001
```

asin(x) AND acos(x)

`asin(x)` returns the arcsin of x, in $[-\pi/2, \pi/2]$

`acos(x)` returns the arccosine of x, in $[0, \pi]$

Terminal

```
>>> from math import *
>>> asin(0)
0.0
>>> asin(1)
1.5707963267948966
>>> asin(1)/(pi/2)
1.0
>>> asin(-1)/(pi/2)
-1.0
```

Terminal

```
>>> from math import *
>>> acos(1)
0.0
>>> acos(0)
1.5707963267948966
>>> acos(-1)
3.141592653589793
>>> acos(-1)/pi
1.0
```

tan(x), atan(x) AND atan2(y, x)

`tan(x)` returns the tangent of x for x in radians

`atan(x)` returns the arctangent of x, in $(-\pi/2, \pi/2)$

`atan2(y, x)` returns the arctangent of $\frac{y}{x}$, in $[-\pi, \pi]$

Terminal

```
>>> tan(-pi/4)
-0.999999999999999
>>> tan(pi/4)
0.999999999999999
>>> atan(-1e16)/(pi/2)
-1.0
>>> atan(0)
0.0
>>> atan(1e10)/(pi/2)
1.0
```

Terminal

```
>>> atan2( 0.0,-1.0)/pi
1.0
>>> atan2( 1.0, 0.0)/pi
0.5
>>> atan2( 0.0, 1.0)/pi
0.0
>>> atan2(-1.0, 0.0)/pi
-0.5
>>> atan2(-0.0,-1.0)/pi
-1.0
```

degrees(x) AND radians(x)

`degrees(x)` returns x converted from radians to degrees

`radians(x)` returns x converted from degrees to radians

Terminal

```
>>> degrees(pi)
180.0
>>> radians(180)
3.141592653589793
```

Part II

Your Turn

Practice Exercise 1

Write a program that prints five rows of stars, where the number of stars in each row is chosen at random from between 1 and 10, including both 1 and 10.

Terminal

```
$ python rndstar.py
******
***
*****
*********
***
```

Editor - rndstar.py

```
1 from random import randint
2 for _ in range(5):
3     length = randint(1,10)
4     print('*'*length)
```

Practice Exercise 2

Write a program that rotates a turtle to a random angle. Repeat the rotation so the turtle turns to random orientation 20 times.

Editor - jitter.py

```
1  from random import uniform
2  from turtle import *
3  setup(420, 420)
4  shapesize(10, 10, 10)
5  shape('turtle')
6  color('black', 'green')
7  speed(1)
8  for _ in range(20):
9      setheading(uniform(0, 360))
10
11 done()
```

Practice Exercise 3

Write a program to choose a dessert from a list of desserts such as `desserts = ['pie', 'cake', 'flan']`

Terminal

```
$ python dessert.py
I want cake.
$ python dessert.py
I want pie.
$ python dessert.py
I want pie.
$ python dessert.py
I want flan.
```

Editor - dessert.py

```
1 from random import choice
2 desserts = ['pie', 'cake', 'flan']
3 dessert = choice(desserts)
4 print(f'I want {dessert}.')
```


Thanks for watching!