
Python Programming
Math and operators

Math and operators

- Basic math in Python.
- The math module.
- The random module.

Arithmetic operators

+	Addition
	5 + 2 returns 7
-	Subtraction
	5 -2 returns 3
*	Multiplication
	5 * 2 returns 10
/	Division
	5 / 2 returns 2.5
%	Modulus
	5 % 2 returns 1
**	Power
	5**2 returns 25
//	Floor Division
	5 //2 returns 2

- The modulus operator (%) is used to find the remainder after division

```
5 % 2 = 1
```

- The floor division operator (//) is the same as regular division, but rounded down

```
10 // 3 = 3
```

```
11 // 3 = 3
```

Exercise 8

- Floor and Modulus
  - 5-10 min

Assignment operators

=	Basic assignment
	a = 2
+=	One step addition and assignment
	a += 2 same as a = a + 2
-=	One step subtraction and assignment
	a -= 2 same as a = a - 2
*=	One step multiplication and assignment
	a *= 2 same as a = a * 2
/=	One step division and assignment
	a /= 2 same as a = a / 2

- Notice division returns a float

```
a = 10
a /= 2
print(a)
5.0
```

- Notice that a changes from an integer to a float when division is performed on it.

- You can use parentheses to change the order of operations a

```
6 + 3 / 3 = 7
(6 + 3) / 3 = 3
```

Built-in math functions

- **int(x)**
- int(x) returns x converted to an integer.
- rounding down (like floor) for positive numbers and rounding up for negative numbers.
- strings can only be an integer
- **float(x)**
- returns x converted to a float
- **abs(x)**
- returns the absolute value of x as an integer or a float.

Built-in math functions

- **min() and max()**
- returns the smallest and largest value of the passed-in arguments

```
>>> min(3.14, -1.5, -300)
-300
>>> max(2, 1, 3)
3
```

- **pow(base, exp[, mod])**
- is the equivalent of using the power operator (**)

```
>>> pow(4, 2)
16
>>> 4**2
16
```

- pow() can take a third argument: mod. pow(base, exp, mod) is functionally equivalent to base**exp % mod

Square Brackets in Code Notation

- Square brackets in code notation indicate that the contained portion is optional. Consider the pow() function signature:

```
pow(base, exp[, mod])
```

- This means that the mod parameter is optional.

Built-in math functions

- **round(number[, ndigits])**
- returns number as a number rounded to ndigits digits after the decimal. If ndigits is 0 or omitted then the function rounds to the nearest integer. If ndigits is -1 then it rounds to the nearest 10 (e.g., round(55, -1) returns 60).

```
>>> round(55, -1)
60

>>> round(3.14)
3

>>> round(-3.14)
-3

>>> round(3.14, 1) 3.1 >>> round(3.95, 1)
4.0
```

Built-in math functions

- **sum(iter[, start])**
- takes an iterable (e.g., a list) and adds up all of its elements. (list will be covered in the Iterables lesson)

The math module

- The math module is built in to Python
- To use it:

```
import math
```

- Common methods:
- **math.ceil(x)**
- x rounded up to the nearest whole number as an integer.

```
>>> math.ceil(5.4)
6

>>> math.ceil(-5.4)
-5
```

The math module

- **math.floor(x)**
- x rounded down to the nearest whole number as an integer.

```
>>> math.floor(5.6)
5

>>> math.floor(-5.6)
-6
```

- **math.trunc(x)**
- x with the fractional truncated, effectively rounding towards 0 to the nearest whole number as an integer.
- To get a full list of the math module’s methods, import math and then type help(math) in the Python shell or visit https://docs.python.org/3/library/math.html.

The math module

- **math.fabs(x)**
- absolute value of float x

```
>>> math.fabs(-5)
5.0
```

- math.factorial(x)
- factorial of x. This is often written as x!, but not in Python!

```
>>> math.factorial(3)
6
```

- **math.pow(x, y**
- x raised to the power y as a float. (Note the difference with the built-in pow())

```
>>> math.pow(5, 3)
125.0
```

The math module

- math.sqrt(x)

```
>>> math.sqrt(25)
5.0
```

- square root of x as a float.
- The math module also contains two constants: math.pi and math.e, for Pi and e as used in the natural logarithm.
- The math library offers many additional methods, including:
  - Logarithmic methods (e.g., math.log(x))
  - Trigonometric methods (e.g., math.sin(x))
  - Angular conversion methods (e.g., math.degrees(x))
  - Hyperbolic methods (e.g., math.sinh(x))

The random module

- The random module is also built into Python and includes methods for creating and selecting random values.
- To use it:

```
import random
```

- Common methods:

```
random.random()
```

- Random float between 0 and 1.

```
>>> random.random()
0.5715141345521301
```

- To get a full list of the random module's methods, import random and then type help(random) in the Python shell or visit https://docs.python.org/3/library/random.html.

The random module

- **random.randint(a, b)**
- Random integer between (and including) a and b.
- **random.randrange(b)**
- Random integer between 0 and b-1.
- **random.randrange(a, b)**
- Random integer between a and b-1.
- **random.randrange(a, b, step)**
- Random integer at step between (and including) a and b-1.
- **random.uniform(a, b)**
- Random float between (and including) a a

The random module

- **random.choice(seq) and random.shuffle(seq)**
- random.choice(seq) returns a random element in the sequence seq.
- random.shuffle(seq) shuffles the sequence seq in place. (Sequences are covered in an upcoming lesson.)

Seeding

- **random.seed(a)** is used to initialize the random number generator. The value of a will determine how random numbers are selected. The following code illustrates this:

```
>>> import random
>>> random.seed(1)
>>> random.randint(1, 100)
18
>>> random.randint(1, 100)
73
>>> random.randint(1, 100)
98
# reseed
>>> random.seed(1)
>>> random.randint(1, 100)
18
```

- Notice that the random numbers generated depend on the seed. If you run this same code locally, you should get the same random integers. This can be useful for testing

Exercise 9 and 10

- How Many Pizzas Do We Need?
  - 15-25 min
- Dice Rolling
  - 15 – 25 min