2.4 Numeric Types

Integers

Integers are the most familiar of all the standard types. Any whole number is an integer. Older computer languages, like C and Java, have limits on the size of integer values due to the way they represent numbers internally. These limits are large, around 2 billion, but occasionally we need numbers that are larger than than these limits and this can create bugs in programs. Python has no limit on the size of an integer value. For example, here is the number 100-factorial (100!), which is the product of all of the integers between 1 and 100: 93326215443944152681699238856266700490715968264381621468592963895217 59999322991560894146397615651828625369792082722375825118521091686400 00000000000000000000000. This is a legitimate integer value in Python.

There are 7 standard arithmetic operators for integers:

Symbol	Meaning	Example	Result
$+$	Addition	$3+5$	8
$-$	Unary: negation Binary: subtraction	$-(3+4)$ $7-3$	-7 4
$*$	Multiplication	$4*7$	28
/	Exact division	11/3 12/3	3.666667 4.0
//	Whole division	11//3 12//3	3 4
**	Exponentiation	$2**4$ $10**-1$	16 0.1
%	Remainder	11%3 12%3	2 0

Most of these need no explanation. It may surprise you that there are two different division operators, but both are useful. The "exact division" operator / gives the quotient of its two operands as a floating point number. The "whole division" operator gives the whole number of times its second operand can be divided into its first operand. You have computed remainders before in mathematics classes, but mathematicians don't have a standard symbol for the remainder operation; programming languages do.

The remainder operator `%` is used more than you might expect. Suppose numbers `x` and `d` are integers. If `x % d` is 0, then `d` divides evenly into `x`. This has many applications. For example, we can tell if a number `x` is even or odd by checking `x % 2`; if this is 0 then `x` is even and if it is 1 then `x` is odd. Similarly, if `day1` and `day2` are two different dates of the same month, then they fall on the same day of the week if `day1%7` is the same as `day2%7`. For example, I am writing this on July 8, 2015, which is a Monday. July 15, July 22 and July 29 are also Mondays: `8%7`, `15%7`, `22%7` and `29%7` are all 1.

There is a rule in most programming languages, that an operation between two integers results in an integer, while an operation between an integer and a float is a float. This holds for all of Python's operators except for exact division / and exponentiation where the second operand, the exponent, is negative. In early versions of Python `a**b` wasn't defined when `b` was negative, but negative exponents are sufficiently convenient that the language designers finally relented and allowed non-integer results.

Here is a table of comparison operators for integers. Each of these returns a Boolean (True or False) value:

Symbol	**Meaning**	**Example**	**Result**
$<$	Less than	$3 < 5$	True
		$5 < 3$	False
$>$	Greater than	$3 > 5$	False
		$5 > 3$	True
$<=$	Less than or equal to	$3 <= 5$	True
		$5 <= 5$	True
		$5 <= 3$	False
$>=$	Greater than or equal to	$3 >= 5$	False
		$5 >= 5$	True
		$5 >= 3$	True
$==$	Equal to (comparison, not assignment)	$3 == 5$	False
		$5 == 5$	True
$! =$	Not equal to	$3! = 5$	True
		$5! = 5$	False

Floating Point numbers

Floats are numbers that are not necessarily integers. They have a more complex internal representation inside the computer (in binary) than integers do, and there is not necessarily a perfect correspondence between decimal values and floating point representations. For example, if you ask Python to compute 1.0/10, the result is not 0.1, but rather 0.10000000000000001. Of course, this is close enough for most computational purposes, but it leads to some unexpected results. If you ask the system whether 1.0/10 is 0.1, the answer is "No". If general, it isn't a good idea to directly compare two floating point values. You can ask whether they are close (say within 0.00000001), but don't test them for equality.

Floats have the same arithmetic operators and the same comparison operators as integers:

Symbol	Meaning	Example	Result
+	Addition	3.2+5	8.2
-	Unary: negation Binary: subtraction	-(3.4+4.5) 7.0-3.9	-7.9 3.1
*	Multiplication	4.2*5	21.0
/	Exact Division	12.6/3 11.0/3	4.2 3.6666666666
//	Whole Division	12.6//3 11.0//3	4.0 3.0
**	Exponentiation	2**3.6	12.125732532083186
%	Remainder	11.0%3 10.5%3.3	2.0 0.6
<	Less than	3.2 < 5.6	True
>	Greater than	3.2 > 5.6	False
<=	Less than or equal to	3.2 <= 5.6	True
>=	Greater than or equal to	3.2 >= 5.6	False
==	Equal to (comparison, not assignment)	3.2 == 5.6	False
! =	Not equal to	3.2! = 5.6	True

When dividing decimal numbers we dont usually think of getting a "remainder", but the definition for floats is the same as it is for integers: `a%b` is what remains after we remove the largest possible integer multiple of `b` from `a`. One way we might compute this for positive values of `a` and `b` is to repeatedly subtract `b` from `a` until the result is smaller than `b`; that result is the remainder.

The *print()* function works with floats just as it does with integers. The easiest way to control the number of decimal places printed with a float is to use string formatting, which we describe in the next section. Similarly, input works with floats just as it does with integers. Finally, the comparison operators for floats are the same as those for integers.

Converting between types

Sometimes you have an integer variable that you want to change to a float, or vice versa. There are several ways to achieve this. In Python you can use type names like functions to convert values to these types. For example, if `x` has an integer value, such as 23, and you want to get the floating point version of this, 23.0, you can get it as **float**(x). Similarly, we can get the integer version of a floating point number `x` with **int**(x). This always rounds `x` towards 0: **int**(34.9) is 34, and **int**(−34.9) is -34.

The Math library

The following table lists many functions that are available in the Math library for Python. To use any of these you need to put the following line at the top of your program:

```
from math import *
```

If you omit this line you will get an error message saying that the functions are not defined.

Alternatively, you can include at the top of your program

```
import math
```

and prefix any item that you use from the library with "math.", as in

```
area = math.pi*(radius**2)
```

The idea here is that you can either import all of the names from the library into your program or you can just import a link that allows you to refer to objects in the library though the <library-name> dot notation.

Symbol	**Meaning**	**Example**	**Result**
ceil(x)	The ceiling of x: the smallest whole number (still a float) greater than or equal to x.	ceil(34.2) ceil(-34.2)	35.0 -34.0
floor(x)	The floor of x: the largest whole number (still a float) less than or equal to x.	floor(34.2) floor(-34.2)	34.0 -35.0
fabs(x)	The absolute value of x: if $x > 0$ this is x, and if $x < 0$ this is x.	fabs(34.2) fabs(-34.2)	34.2 34.2
exp(x)	This returns where e is the base of the natural logarithm.	exp(2)	7.38906
log(x)	This is the natural logarithm; some math books call it $ln(x)$.	log(15)	2.70805
log(x, base)	This is the logarithm of x with the given base.	log(32, 2)	5.0
log10(x)	This is log(x, 10).	log10(100)	2.0
sqrt(x)	The square root of x.	sqrt(144)	12.0
sin(x)	This is the sine of angle x, where x must be measured in radians. If you want to use degree measure, use sin(radians(x)).	sin(30) sin(radians(30))	-0.988032 0.5
cos(x)	This is the cosine of angle x, where x must be measured in radians. If you want to use degree measure, use cos(radians(x)).	cos(60) cos(radians(60))	-0.952413 0.5
tan(x)	This is the tangent of angle x, where x must be measured in radians. If you want to use degree measure, use tan(radians(x)).	tan(45) tan(radians(45))	1.619775 1.0
asin(x)	This is the angle, in radian measure, that has x as its sine.	asin(0.5) degrees(asin(0.5))	0.523599 30.0
acos(x)	This is the angle, in radian measure, that has x as its cosine.	acos(0.5) degrees(acos(0.5))	1.047198 60.0
atan(x)	This is the angle, in radian measure, that has x as its tangent.	atan(1.0) 0.785398 degrees(atan(1.0))	 45.0
degrees(x)	This takes angle x, measured in radians, and returns the number of degrees in x.	degrees(pi/4)	45.0
radians(x)	This takes angle x, measured in degrees, and returns the number of radians in x.	radians(45)	0.785398
hypot(x, y)	This returns which is the length of the hypotenuse of a right triangle whose other sides have lengths x and y.	hypot(3, 4)	5.0
pi	pi is the ratio of the circumference of a circle to its diameter.	pi	3.14159
e	e is the base of the natural logarithm.	e	2.71828

The Random library

Random numbers are very useful for games, simulations, and other sorts of programs that we will write. Python has a very powerful, easy-to-use library of random number utilities. To use any of these, you need to put one of the following line at the top of your program:

```
from random import *
import random
```

With the former line you can use the random functions directly; with the latter line you need to prefix them with "random.", as in `random.randint(1, 6)`. The following is a small selection of the functions that are available in this library. All of these functions return floating point values.

Symbol	Meaning	Example	Result
random()	This returns a random floating poing number between 0 and 1.	random()	0.23145
uniform(a, b)	This returns a random floating poing number between a and b.	uniform(3, 5)	4.21368
randint(a, b)	This returns a random integer between a and b, possibly equal to either a or b.	randint(0, 1) randint(0, 1)	0 1
seed(x)	This resets the starting point of the random number generator based on the value of x, which could be an integer, a floating point value or a string. If you want number to be random but repeatable, set the seed to the same value each time. This doesnt return anything.	seed("bob")	No output.