Note

In the Notes, any code will be in blue and any output in red.

Statements

A *statement* is a unit of code that the Python interpreter can execute. We have seen two kinds of statements: print being an expression statement and assignment.

When run a statement, the interpreter executes it and displays the result, if there is one.

A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute. For example, the script

```
print(1)

x = 2

print(x)
```

produces the output:-

```
1

2
```

The assignment statement produces no output.

Operators and operands

Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called *operands*.

The operators +, -, *, /, and ** perform addition, subtraction, multiplication, division, and exponentiation, as in the following examples:

```
20+32

hour-1

hour*60+minute

minute/60 5**2

(5+9)*(15-7)
```

In Python 3.x, the result of this division is a floating-point result:

```
minute = 59

minute/60

0.9833333333333333
```

Note the // operator will divide and round down so

```
minute//60
```

```
0
```

Expressions

An *expression* is a combination of values, variables, and operators. A value all by itself is considered an expression, and so is a variable, so the following are all legal expressions (assuming that the variable x has been assigned a value):

```
17 x
```

```
x + 17
```

If you run an expression, the interpreter *evaluates* it and displays the result:

```
1 + 1
```

```
2
```

But in a script, an expression all by itself doesn't do anything! This is a common source of confusion for beginners.

Order of operations

When more than one operator appears in an expression, the order of evaluation depends on the *rules of precedence*. For mathematical operators, Python follows mathematical convention. The acronym *PEMDAS* is a useful way to remember the rules:

- *P*arentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 2 * (3-1) is 4, and (1+1)**(5-2) is 8. You can also use parentheses to make an expression easier to read, as in (minute * 100) / 60, even if it doesn't change the result.
- *E*xponentiation has the next highest precedence, so 2**1+1 is 3, not 4, and 3*1**3 is 3, not 27.
- *M*ultiplication and *D*ivision have the same precedence, which is higher than *A*ddition and *S*ubtraction, which also have the same precedence. So 2*3-1 is 5, not 4, and 6+4/2 is 8, not 5.
- Operators with the same precedence are evaluated from left to right. So the expression 5-3-1 is 1, not 3, because the 5-3 happens first and then 1 is subtracted from 2.

When in doubt, always put parentheses in your expressions to make sure the computations are performed in the order you intend.

Modulus operator

The *modulus operator* works on integers and yields the remainder when the first operand is divided by the second. In Python, the modulus operator is a percent sign (%). The syntax is the same as for other operators:

```
quotient = 7 // 3

print(quotient)
```

```
2
```

```
remainder = 7 % 3

print(remainder)
```

```
1
```

So 7 divided by 3 is 2 with 1 left over.

The modulus operator turns out to be surprisingly useful. For example, you can check whether one number is divisible by another: if x % y is zero, then x is divisible by y.

You can also extract the right-most digit or digits from a number. For example, x % 10 yields the right-most digit of x (in base 10). Similarly, x % 100 yields the last two digits.

String operations

The + operator works with strings, but it is not addition in the mathematical sense. Instead, it performs *concatenation*, which means joining the strings by linking them end to end. For example:

```
first = 10

second = 15

print(first+second)
```

```
25
```

```
first = '100'

second = '150'

print(first + second)
```

```
100150
```

The * operator also works with strings by multiplying the content of a string by an integer. For example:

```
first = 'Test '

second = 3

print(first * second)
```

```
Test Test Test
```