1 Function calls

A **function** is a named sequence of statements that performs a computation. Later, we will learn how to define our own functions. For now, we learn how to **call** functions that were written by another programmer.

Function calls have the following form:

function_name(arguments)

How python executes a function call:

1. Evaluate the arguments.
2. Call the function, passing in the argument values.
3. Perform the computation of the function and return the resulting value.

Terminology:

- **argument**: a value given to a function
- **pass**: to provide to a function
- **call**: to ask python to evaluate a function
- **return**: to pass back a value (the result of the function's evaluation)

In this example, the function `pow` is *called*. The *arguments* `3` and `4` are *passed* to `pow`. The function `pow` computes three raised to the power of four and *returns* the value 81.

```
>>> pow(3, 4)
81
```

1.1 Built-in functions

```
>>> int(2.3)   # makes a new int from argument, if possible
2
>>> float(2)   # makes a new float from argument, if possible
2.0
>>> x = int(4.3)
>>> type(x)    # returns the argument's type
<type 'int'>
>>> max(-30, 12, 4.0, 6.75)  # returns largest argument
12
>>> min(-30, 12, 4.0, 6.75)  # returns smallest argument
-30
>>> abs(-30)   # returns absolute value of argument
30
```

1.2 Help function

The help function takes one argument, such as the name of a function, and displays documentation.

```
>>> help(abs)
Help on built-in function abs in module __builtin__:

abs(...)
    abs(number) -> number
    Return the absolute value of the argument.
```

In the help documentation, arguments in brackets, such as `z` below, are optional.

```
>>> help(pow)
Help on built-in function pow in module __builtin__:

pow(...)
    pow(x, y[, z]) -> number
    With two arguments, equivalent to x**y.  With three arguments,
    equivalent to (x**y) % z, but may be more efficient (e.g. for longs).
```

Example:

```
>>> pow(2, 4)
16
>>> pow(2, 4, 3)
1
```

1.3 Functions in expressions and expressions in functions

An **expression** can include not only operators, values, and variables, but also function calls.

```
>>> max(3, 4) * 2 + abs(-3)
8
```

Thus, we can revise our definition of an expression: An **expression** is a combination of operators, values, variables and function calls that python can evaluate, resulting in a single value.

So, functions can appear in expressions. In addition, expressions can appear in the arguments to a function.

```
>>> max(3 ** 4, 4 ** 3)
81
>>> min(3 ** 4, 4 ** 3)
64
```

General principle: anywhere you can put a data value, you can put an expression. This is because python will evaluate the expression to produce a data value. The examples above illustrate this principle: one of the arguments to `max` is the expression `3 ** 4` which evaluates to the number 81.

1 Strings

A string is a sequence of characters. In python, strings belong to the data type `str`. Recall that a **type** is a set of values and operations that can be performed on those values. A string can be any combination of letters, digits, and/or punctuation marks. The operators are shown in this table:

Operator	Operation	Expression	Description	Result
+	concatenation	`'hey,'+ ' you'`	concatenate strings	`'hey, you'`
*	string copy (`str * int`)	`'hi!'* 3`	make three copies of `'hi!'`	`'hi!hi!hi!'`
	string copy (`int * str`)	`2 * 'hi!'`	make two copies of `'hi!'`	`'hi!hi!'`

All other mathematical operators such as -, /, and % *not* supported on strings and result in an error.

To make a string, enclose some characters in quotation marks, either single or double. Note: the quotes mark the begin and end of the string but are not considered part of the string.

```
>>> school_name = 'Colgate'
>>> team_name = "Raiders"
>>> school_name + team_name
'ColgateRaiders'
```

To include a `'` or `"` in the string, simply enclose the string in the opposite kind of quotation mark or use the escape character \. The escape sequence `\'` tells python that the quote is simply a quote and not the end of the string.

```
>>> cheer = "Go 'Gate!"
>>> another_cheer = 'Let\'s go \'Gate!'
>>> another_cheer + cheer
"Let's go 'Gate!Go 'Gate!"
>>> cheer * 3
"Go 'Gate!Go 'Gate!Go 'Gate!"
```

The `str` function can be used to make a string representation of a numerical value.

```
>>> shoe_size = 10.5
>>> "My shoe size is " + str(shoe_size)
'My shoe size is 10.5'
```

2 Writing programs

IDLE can be used in one of two modes:

(a) Interactive mode. Statements are typed directly into the shell and each one is evaluated immediately.

(b) Script mode. Program statements are written in a file and then the program is executed by selecting "Run module" from the "Run" menu.

The interactive mode is great for trying out new ideas. But when we write programs, even small ones, we usually use script mode.

2.1 Program output: print

When running a program in script mode, one cannot see the result of an expression unless you explicitly *print* it using the `print` statement.

```
print "Hello, world!"
```

The `print` statement can accept a comma-separated list of expressions. They will be evaluated and then printed all on one line separated by spaces.

```
name = 'Alice'
age = 19
print name, "is", age, "years old."
```

The code above prints `Alice is 19 years old`. The same thing can be accomplished using the concatenation operator. Notice below how spaces must be explicitly added to the strings and also that the `str` function is used to make `'19'` (a string) from `age` (an `int`):

```
name = 'Alice'
age = 19
print name + " is " + str(age) + " years old."
```

By default, the print statement also prints a newline character. Thus, a subsequent print statement appears on a new line. To print statements on the same line, end the print statement with a comma.

```
print "hello!"
print "bye!",              # <----  note the comma at the end
print "see ya!"
```

Will print this:

```
hello!
bye! see ya!
```

2.2 Program input: raw_input

The `raw_input` function can be used to get input from the user. The function is called with one argument, a string which acts as a user prompt. When the function is called, it automatically prints the prompt and then waits for the user to enter something. The user input is returned. Example:

```
name = raw_input("What is your name? ")
print "Hello, " + name + "!"
```

If the user types `Michael` in response to the prompt, the program will print `Hello, Michael!`

Important: `raw_input` always returns a string, even if the user types digits. Use `int` or `float` to make a numerical value from the string.