

Functions

A function is a named sequence of statements that performs a desired operation.

When you brush your teeth in the morning, you

- 1) open the cabinet,
- 2) grab your brush and paste,
- 3) put some paste on your brush,
- 4) open your mouth,
- 5) push the brush across your teeth.

Think of this sequence of steps as a **function**.

Now when a mom tells a kid to brush his teeth, she doesn't want to explain all these steps each time. When the mom says, "brush your teeth", she is making a **function call**.

'Brush your teeth' is an **abstraction** for all the many steps involved.

Programming computers is hard because of the precision and detail required. You have to tell the computer 'steps' at a very detailed level.

The top software engineers often don't write a lot of code directly. They just design giant pieces of software, working at the structure chart level (or UML level for object-oriented programs).

The goal is to divide and conquer, to break the problem down into parts small enough that it easy to program. You also want to create reusable components that can be used in many different software applications.

Calling Functions

In an earlier tutorial, we called some built-in functions of Python, such as 'type' and 'len'

The 'len' function returns the length of a list:

```
while i<len(list1)
```

Functions in programming are like math functions: you provide the name of the function, e.g., 'len', followed by an open parenthesis, and then some parameters (e.g., list1) and then a closed parenthesis.

The function returns a value to the caller. In the example above, the return value is part of a condition. Other times, you'll want to "catch" the return value in a variable, e.g.,

```
length = len(list1)
```

Built-in functions like 'len' can be called from any python program or the interpreter. But there are also thousands of Python functions that are not 'built-in' but are still quite useful. There are many Python libraries to use, and you can also create libraries of your own.

Importing Modules

Within the interactive interpreter or a program file, you can import existing python code. We call collections of reusable code a 'code library' or a 'module'.

For Mastermind, you'll need the 'random' module.

For this tutorial, we'll import the math module. From the interactive interpreter prompt, type the following:

```
>>> import math
>>> math.sqrt(16)
```

The import statement says you want to use math functions. But when you import a module, like 'math', you must qualify any function calls you make to functions with "math." For the above example, if you typed in just 'sqrt(16)' you'd get an error. 'math.sqrt(16)' is understandable to Python because it knows that sqrt is within the math module.

Try the following math functions

pow (x,y) -- returns x to the yth power, so pow(2,3) is 8
acos(x) - returns cosine of x

A description of all the functions in the math module is at:

<http://docs.python.org/lib/module-math.html>

Writing Your Own Functions

Here is an example of a function:

```
def square(number):  
    result=number*number  
    return result
```

A function is basically a list of commands. By assigning a name to a bunch of commands, we create an abstraction that can be used in a larger program.

Take, for example, computing the total of all elements in a list. Code to perform this operation might require several commands. If we write a function called 'totalList' we can then call it whenever we need to total a list (instead of copy-paste).

Parameters help make a function more flexible and reusable. This idea of making your code as general as possible is a key in software engineering.

The caller of the function must supply the correct number of arguments in the function call. For instance, to call the 'square' function above, the caller would have to send in one parameters, e.g.,

```
result= square(4)
```

Calling totalList would be done with totalList(list)

The data sent to a function are called the 'actual parameters'.

Though Python doesn't require (or allow) the programmer to specify the types of parameters, if the caller sends an inappropriate type, the function won't work.

Return values

A function can return a value to the calling code with the 'return' statement. This is often the last statement in the function.

The calling code often catches the return value in a variable, e.g.,

```
    val = square(2)
or
    list=[3,5,2]
    total = totalList(list)
```

But as you saw with the 'len' function, sometimes the function call is part of another construct, like a condition.

Structure of a Python Program

With functions, one can break a program into simple parts. A typical Python program will have several functions and a 'main' body of code, as below:

```
def func1(param1,param2):
    #...

def func2(param1):
    #...

#... more functions

# main code just starts with no indentation
#...
```

Worksheet

0. Using Python's interactive interpreter, try out the math module. Call at least three methods to see how they work.
1. In a file mymath.py, write a function 'cube(x)' which returns the cube of the parameter x. The file's 'main' code should call 'cube(x)' a couple of times and print the results.
2. In mymath.py, import the math module and rewrite the cube function so that it calls the pow function to actually do its work.
3. Write a function 'getGuess' that asks the user to input four colors and returns the four colors in a list. Call your function from the 'main' program and print out the list.