Python: Functions

Thomas Schwarz, SJ
Marquette University

History

- Early computer programming was difficult

  - Not only because interacting with the computer was difficult

    - Data was entered by setting switches, using punched tapes or cards, electromagnetic tapes, etc

  - But also interaction was at the machine level

  - Earliest instructions were in binary

History

- Assembler were invented to translate human readable instructions into machine language

- Only later were "higher level programming languages" developed such as Fortran (for FORmula TRANslator) and Cobol (COmmon Business Language)

History

- Some tasks were also repetitive

  - Such as calculating the sine of a number

  - The necessity to calculate sine gave rise to the first procedure

    - The procedure expect its input at a certain location

    - It writes it output at another certain location

    - It consists of a block of lines of code

    - Procedure calling works like this:

      - The caller loads the input locations with the data

      - It also stores the address of the next instruction at a well-known location, the *return address*

      - Program control jumps to the beginning of the procedure

      - The procedure executes and loads its results in the output locations

      - The procedure then jumps to the *return address*

      - The caller finds the result at a certain location

History

- Besides the capability to re-use code, sub-procedures were also an important tool to break a complicated task into smaller pieces

- This is called modularization

- It's been a main-stay in software engineering ever since

Python Functions

- Python almost completely uses the abstraction of a function

- A function is called from the caller, given none or a number of *arguments* (aka parameters)

- The function returns to the caller

  - Giving a return value (a *fruitful* function)

  - Or just returning

Python Function

- Calling fruitless functions

  - We already have used a fruitless function, namely `print`

    - `print` is special, it can have any number of arguments

  - Example: `print("The value is", 3.145)`

    - Two arguments

      - String `"The value is"`

      - Floating point `3.145`

Python Functions

- We can use built-in fruitful functions

  - `abs` returns the absolute value

    ```
    >>> abs(-4)
    4
    ```

  - We can import the module math in order to have access to many mathematical functions

    - A complete list is in the Python Docs.

      - Here we just print out the values of some functions.
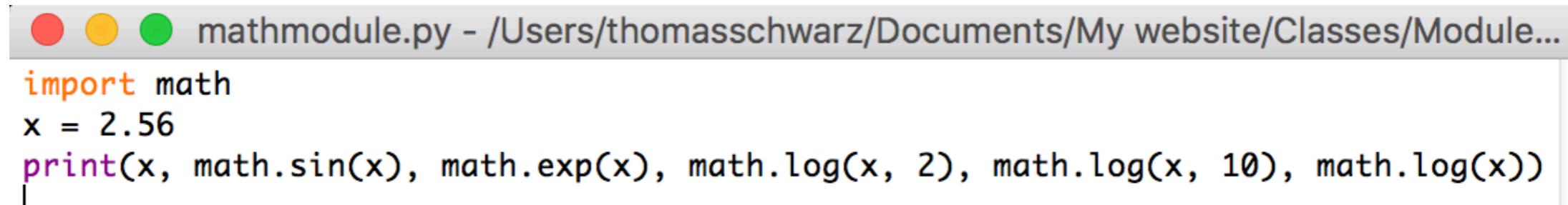
```
mathmodule.py - /Users/thomasschwarz/Documents/My website/Classes/Module...

import math
x = 2.56
print(x, math.sin(x), math.exp(x), math.log(x, 2), math.log(x, 10), math.log(x))
```

Python Functions

- Creating functions

```
def function_name ( parameter_list ) :
```

Indent ←——————→ Statement Block

- Uses key word def

- Followed by the name of the function (usually lower-letter)

- In parentheses, a list of arguments (aka parameters) separated by comma

- Followed by colon

Python Functions

- Example for a fruitless function

  - Function that prints out n asterisks, then a blank line, then n asterisks

    ```
    def asterisks(n):
        print(n*"*")
        print()
        print(n*"*")
    ```

  - There is a single argument, n

    - Note that n does not have a specified type.

    - Since in the body of the function, we multiply with n, it better be an integer.

Python Functions

- Example for a fruitless function

  - Function that prints out *n* asterisks, then a blank line, then *n* asterisks

    ```
    def asterisks(n):
        print(n*"*")
        print()
        print(n*"*")
    ```

  - Three statements follow in the function block.

  - The function execution finishes, when we fall out of the block

  - If we want to be explicit, we can add a final line to the function block with the single statement `return`

Python Function

- Example for a fruitful function

  - A function that given x and y, calculates the expression

  $$\frac{|x - y|}{x^2 + y^2}$$

  - The function needs to arguments and needs to return a value

Python Functions

$$x, y \rightarrow \frac{|x - y|}{x^2 + y^2}$$

```
def fun(x, y):
    enumerator = abs(x-y)
    denominator = x*x+y*y
    return enumerator/denominator
```

- There are now two arguments, separated by a comma

- The *body* of the function calculates the result

- The result is returned with the return-statement.

- An exception will be thrown if we call the function with values 0 and 0 since we then divide by zero in the calculation of the function.

Python Functions

- We can have more than one return statement

  - An implementation of the maximum of two numbers function

    ```
    def my_max(x,y):
        if x<y:
            return y
        return x
    ```

  - I do not need to put the last line in an else, since if x<y, then I already jumped out of the execution of the function body.