ECOR 1041

Computation and Programming

Introduction to Functions:
Built-In Functions, Importing Functions from Modules

Carleton University

References

- *Practical Programming*, 3rd ed.
    - Chapter 3, pp. 31 – 35
        - This is the introduction and first two sections of Chapter 3:
            - Functions That Python Provides
            - Memory Addresses: How Python Keeps Track of Values
    - Chapter 6, pp. 99 – 104
        - This is the introduction and first section of Chapter 6:
            - Importing Modules

Lecture Objectives

- Learn about computation using Python’s built-in functions and functions that are provided in modules

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
  - Function
  - Function call, call expression
  - Function argument
  - Module
  - `import` statement

Learning Outcomes

- Write expressions that call functions that are built in to Python
- Write expressions that call functions that are imported from a module

What is a Function (Mathematics)?

- Mathematicians use numerous symbols to denote different operations:
  - $\times$, $\div$, x^n, $\lfloor x \rfloor$, $\sin x$, $\sqrt{x}$, $x!$, $\int x\,dx$, …

What is a Function (Mathematics)?

- These symbols represent specific cases of a fundamental concept: the *function*
- A function f is a mapping from a set A (the *domain*) to a set B (the *codomain*) such that every element $a \in A$ is uniquely associated with an element $f(a) \in B$
- Example: a function of one variable $f : x \rightarrow f(x)$, where $f(x) =$ *an expression written in terms of* x

Function Application

- Consider $f : x \longrightarrow f(x)$, where $f(x) = x^2 + 4$
- In conventional mathematics, we denote the *application* of the function to its argument this way: *f*(*argument*)
- Example: $f(5)$
  - Argument 5 is substituted for *x* in the expression, then the expression is evaluated:

$$f(5) \Rightarrow 5^2 + 4 \Rightarrow 29$$

Python’s Built-in Functions

- Several functions are *built in* to Python
- Example: `abs` maps its argument (an `int` or a `float`) to the absolute value of that number

```
>>> abs
<built-in function abs>
```

Python Built-in Functions: Experiments

- In Python, function applications are known as *function calls* or *call expressions*
- Type some call expressions in the shell

```
>>> abs(4.2)
4.2
>>> abs(-6)
6
```

Python displays the value of the call expression, which is the value produced by the function

More Built-in Functions: Experiments

```
>>> min(-1 , 2)
-1
>>> max(1, -2, 3, -4)
3
>>> pow(5, 2)  # equivalent to 5 ** 2
25
>>> round(7.3)
7
```

The built-in `round` function

- Python’s `round()` function uses the rounding “half to even” strategy.
  - This means that x.5 is rounded to the nearest **even** integer.
    - Thus 1.5 rounds to 2, and so does 2.5.
    - Why?
      - It is a long story but it is less biased.

The built-in `print` function

- Python’s `print()` function is used to print a value or values, e.g.

```
>>> x = 5
>>> print(x)
5
```

Function Calls

- General form:

```
function_name(argument-list)
```

- Each argument is an expression; i.e., we are not limited to literal values and variables
- Example

```
>>> x = 5
>>> y = -9
>>> abs(x + y)
```

Function Calls: Steps

- When a function is called, Python:
  1. Evaluates all arguments, left to right.
  2. Passes the memory addresses of these values (objects) to the function.
  3. Executes the function. This produces a value.

Function Calls Produce Values

- A function call (call expression) produces a value, so function calls can be used in expressions; e.g.,

```
>>> abs(-2) + abs(-4.7)
6.7
```

- Here, the values produced by `max` and `abs` are used as the arguments of the call to `pow`

```
>>> pow(max(3, 4), abs(-2)) # calcs 4 ** 2
16
```

Modules: `math` Module

- A *module* is a file containing a collection of (usually) closely-related functions and variables
- Python's `math` module provides many math functions, plus variables for a few well-known values; e.g., **π**, *e*
- To use the functions and variables that are defined in a module we must first *import* it

```
>>> import math
```

`math` Module: Getting Help

- Use the built-in `help` function to learn what is in the `math` module

```
>>> import math
>>> help(math)
...  # Lots of output!
```

`math` Module: Getting Help

```
>>> help(math.sqrt)
Help on built-in function sqrt in module math:
sqrt(x, /)
  Return the square root of x.
```

- This could be more informative.
  - Is the argument an `int`? A `float`? Is either type ok?
  - What is the type of the values produced by `sqrt`?

`math` Module: Calling `sqrt`

```
>>> sqrt(25)
builtins.NameError: name 'sqrt' is not defined
>>> math.sqrt(25)
5.0
>>> math.sqrt(-25)
builtins.ValueError: math domain error
```

Use the dot operator to specify that we are calling the `sqrt` function that is inside the `math` module

`import` Statement

- `import math`
  - Imports everything keeping the math “namespace” so we refer to items in the library as `math.sqrt()`, `math.pi`, etc.
  - Ensure you understand this statement
- `import math as m` (may use any unique name instead of `m`)
  - Same as the above, but we now use `m.sqrt()`, `m.pi`, etc.
  - Slightly less typing

`import` Statement (continued)

- `from math import sqrt`
  - Imports only `sqrt`, and you refer to it as `sqrt()`, not `math.sqrt()`.
  - `sqrt` is imported into the `global` namespace (instead of being in `math`’s namespace)
- `from math import pi as PI` (may use any unique name instead of `PI`)
  - Imports only `pi`, and you refer to it as `PI`, not `pi` or `math.pi`.
  - Advantage: now it looks like a constant (though we can still change it, but we should not!).

`import` Statement (continued)

- `from math import *`
  - Imports everything from math into the global namespace
  - no "`math.`" needed
  - Advantage: less typing
  - Disadvantage: potential name clashes

Recap of Learning Outcomes

Learning Outcomes (Vocabulary)

- Know the meaning of these words and phrases
    - Function
    - Function call, call expression
    - Function argument
    - Module
    - `import` statement

Learning Outcomes

- Write expressions that call functions that are built in to Python
- Write expressions that call functions that are imported from a module