Module 3

Function Calls
Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
- Function calls have the form
  \[
  \text{name}(x,y,\ldots)
  \]
  - Arguments are themselves expressions
  - Arguments are separated by commas
Built-In Functions

- Python has several math functions
  - `round(2.34)`
  - `max(a+3,24)`

- You have seen many functions already
  - Type casting functions: `int()`, `float()`, `bool()`

- Documentation of all of these are online
  - [https://docs.python.org/3/library/functions.html](https://docs.python.org/3/library/functions.html)
  - Most of these are two advanced for us right now

Arguments can be any **expression**
Functions as Commands/Statements

• Most functions are expressions.
  ▪ You can use them in assignment statements
  ▪ Example: \( x = \text{round}(2.34) \)

• But some functions are commands.
  ▪ They instruct Python to do something
  ▪ Help function: \text{help}() ▪ Quit function: \text{quit}()

• How know which one? Read documentation.
Case Study: String Functions

• String processing is a major feature of Python
  ▪ Easier than in many other languages
  ▪ Will be the focus of first major assignment

• Also highlights the flexibility of functions
  ▪ Many string functions are expressions
  ▪ But some of the most important are commands

• Let’s examine three important functions
Function \texttt{len}

- **Used as an expression**
  - Value is \# of chars in \texttt{s}
  - Evaluates to an \texttt{int}

- **Examples:**
  - \texttt{s = 'Hello'}
  - \texttt{len(s) == 5}
  - \texttt{len('all') == 3}
  - \texttt{len(s+'all') == 8}

Used in many expressions
Function `print`

- **Used as a command**
  - Displays arguments on screen

- **Examples:**
  - `print('Hello')`
    - Hello
    - This is not a value!
  - `x = print('Hello')` is None
  - `print('Hello
World')`
    - Hello
    - World
    - Translates special characters

- `print` should be called by itself, not in an expression
One Last Function: `input`

```python
>>> input('Type something')
Type something: abc
'abc'

>>> input('Type something: ')  # Like print but it waits for typing
Type something: abc
'abc'

>>> x = input('Type something: ')  # Evaluates to what is typed
Type something: abc
'abc'

>>> x  # Can assign its value
'abc'
```
One Last Function: **input**

```python
>>> input('Type something')
Type something: abc
'abc'
```

Like print but it waits for typing.

```python
>>> input('Type something: ')
Type something: abc
'abc'
```

Will see the purpose function of this later.

```python
>>> x = input('Type something: ')  # x can be assigned
Type something: abc
'abc'
```

Can assign its value.
Built-in Functions vs Modules

- The number of built-in functions is small
  - [http://docs.python.org/3/library/functions.html](http://docs.python.org/3/library/functions.html)

- Missing a lot of functions you would expect
  - Example: cos(), sqrt()

- **Module**: file that contains Python code
  - A way for Python to provide optional functions
  - To access a module, the `import` command
Example: Module `math`

```python
>>> import math
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.cos(0)
1.0
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

To access math functions

Functions require math prefix!

Module has variables too!
Example: Module `math`

```python
>>> import math
>>> cos(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.cos(0)
1.0
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
```

Other Modules

- **io**
  - Read/write from files
- **random**
  - Generate random numbers
  - Can pick any distribution
- **string**
  - Useful string functions
- **sys**
  - Information about your OS
Reading Documentation

- Being able to read docs is an important skill
  - It is impossible for you to memorize everything
  - If you need something, expected to look it up
- Reason why programmers have large monitors
  - Can have documentation open at all times
  - Does not get in the way of programming
- But reading documentation requires training
  - Information laid out in a very specific way
  - May not be obvious to a beginner
9.2. math — Mathematical functions

This module is always available. It provides access to the mathematical functions defined by the C standard.

These functions cannot be used with complex numbers; use the functions of the same name from the cmath module if you require support for complex numbers. The distinction between functions which support complex numbers and those which don’t is made since most users do not want to learn quite as much mathematics as required to understand complex numbers. Receiving an exception instead of a complex result allows earlier detection of the unexpected complex number used as a parameter, so that the programmer can determine how and why it was generated in the first place.

The following functions are provided by this module. Except when explicitly noted otherwise, all return values are floats.

9.2.1. Number-theoretic and representation functions

math.ceil(x)
Return the ceiling of x, the smallest integer greater than or equal to x. If x is not a float, delegates to x.__ceil__(), which should return an Integral value.

math.copysign(x, y)
Return a float with the magnitude (absolute value) of x but the sign of y. On platforms that support signed zeros, copysign(1.0, -0.0) returns -1.0.

math.fabs(x)
Return the absolute value of x.

math.factorial(x)
Return x factorial. Raises ValueError if x is not integral or is negative.

math.floor(x)
Return the floor of x, the largest integer less than or equal to x. If x is not a float, delegates to x.__floor__(), which should return an Integral value.

math.fmod(x, y)
Return xmod(y, x), as defined by the platform C library. Note: C standard is that fmod(x, y) be exactly (mathematically: to infinite precision) equal to x - x*y for some integer n such that the result has the same sign as x and magnitude less than abs(y). Python’s x % y returns a result with the sign of y instead, and may not be exactly computable for float arguments. For example, fmod(-10.0, 3.0) is -1.0, but the result of Python’s -10.0 % 3.0 is 2.0, which cannot be
Reading the Python Documentation

- **Function name**: `math.ceil(x)`
- **Possible arguments**: X
- **What the function evaluates to**: Return the ceiling of x, the smallest integer greater than or equal to x.

Example:
```
math.ceil(3.5)  # Returns 4.0
math.ceil(4.0)  # Returns 4.0
math.ceil(4.1)  # Returns 5.0
```
Alternative: help()

>>> import math
>>> help(math)
Help on module math:

NAME
  math

FUNCTIONS
  acos(...)
    acos(x)

Return the arc cosine (measured in radians) of x.

help can take an argument
Always available, but not as searchable
Hit space to page through
Using the `from` Keyword

```python
>>> import math
>>> math.pi
3.141592653589793

>>> from math import pi
>>> pi
3.141592653589793

>>> cos(pi)
ERROR

>>> from math import *
>>> cos(0)
1.0
>>> sin(0)
0.0

Must prefix with module name

No prefix needed for variable pi

ONLY imported pi

No prefix needed for **anything** in math
```

---

### Using the `from` Keyword

- **Example 1:**
  ```python
  >>> import math
  >>> math.pi
  3.141592653589793
  ```
  - **Note:** You must prefix with the module name `math`.

- **Example 2:**
  ```python
  >>> from math import pi
  >>> pi
  3.141592653589793
  ```
  - **Note:** No prefix needed for the variable `pi`.

- **Example 3:**
  ```python
  >>> cos(pi)
  ERROR
  ```
  - **Note:** Only the imported `pi` is available.

- **Example 4:**
  ```python
  >>> from math import *
  >>> cos(0)
  1.0
  >>> sin(0)
  0.0
  ```
  - **Note:** No prefix needed for anything in `math`. You can directly use functions like `cos` and `sin`.
Be careful using from!

- Using import is *safer*
  - Modules might conflict (functions w/ same name)
  - What if import both?

- **Example**: numpy
  - Has cos, sin too
  - Why? Performance (scientific computing)
  - But not always installed!
Renaming

>>> import math as m
>>> m.cos(0)
1.0

Can rename a module

>>> from math import cos as fred
>>> fred(0)
1.0

Can rename a function
Nested Modules

>>> import introcs.strings
>>> introcs.strings.strip(' abc ')
'abc'

>>> from introcs import strings
>>> strings.strip(' abc ')
'abc'

>>> from introcs.strings import strip
>>> strip(' abc ')

Importing introcs imports all modules that it contains