String: Text as a Value

- String are quoted characters
  - 'abc d' (Python prefers)
  - "abc d" (most languages)
- How to write quotes in quotes?
  - Delineate with "other quote"
  - **Example**: " ' " or ' " '
  - What if need both " and ' ?
- **Solution**: escape characters
  - Format: \ + letter
  - Special or invisible chars

Type: str

Char	Meaning
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	backslash

String are Indexed

- s = 'abc d'

```
 0 1 2 3 4
 a b c   d
```

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

```
 0 1 2 3 4 5 6 7 8
 H e l l o   a l l
```

- What is s[3:6]?

A: 'lo a'
B: 'lo'
C: 'lo '
D: 'o '
E: I do not know

String are Indexed

- s = 'abc d'

```
 0 1 2 3 4
 a b c   d
```

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

```
 0 1 2 3 4 5 6 7 8
 H e l l o   a l l
```

- What is s[:4]?

A: 'o all'
B: 'Hello'
C: 'Hell'
D: Error!
E: I do not know

Other Things We Can Do With Strings

- **Operation** in: s_1 in s_2
  - Tests if s_1 "a part of" s_2
  - Say s_1 a *substring* of s_2
  - Evaluates to a bool
- **Examples**:
  - s = 'abracadabra'
  - 'a' in s == True
  - 'cad' in s == True
  - 'foo' in s == False

- **Function** len: len(s)
  - Value is # of chars in s
  - Evaluates to an int
- **Examples**:
  - s = 'abracadabra'
  - len(s) == 11
  - len(s[1:5]) == 4
  - s[1:len(s)-1] == 'bracadabr'

Function Calls

- Python supports expressions with math-like functions
  - A function in an expression is a **function call**
  - Will explain the meaning of this later
- Function expressions have the form **fun**(x,y,…)

function name

argument

- **Examples** (math functions that work in Python):
  - round(2.34)
  - max(a+3,24)

Arguments can be any **expression**

Built-In Functions

- You have seen many functions already
  - Type casting functions: int(), float(), bool()
  - Dynamically type an expression: type()
  - Help function: help()
- Getting user input: raw_input()
- print <string> is **not** a function call
  - It is simply a statement (like assignment)
  - But it is in Python 3.x: print(<string>)

Arguments go in (), but name() refers to function in general

Method: A Special Type of Function

- Methods are unique (right now) to strings
- Like a function call with a "string in front"
  - Usage: ***string*.*method***(*x*,*y*…)
  - The string is an *implicit argument*
- Example: upper()
  - s = 'Hello World'
  - s.upper() == 'HELLO WORLD'
  - s[1:5].upper() == 'ELLO'
  - 'abc'.upper() == 'ABC'

Will see why we do it this way later in course

Examples of String Methods

- s_1.index(s_2)
  - Position of the first instance of s_2 in s_1
- s_1.count(s_2)
  - Number of times s_2 appears inside of s_1
- s.strip()
  - A copy of s with white-space removed at ends

- s = 'abracadabra'
- s.index('a') == 0
- s.index('rac') == 2
- s.count('a') == 5
- ' a b '.strip() == 'a b'

See Python Docs for more

Built-in Functions vs Modules

- The number of built-in functions is small
  - http://docs.python.org/2/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
  - Access the functions using module as a *prefix*

Example: Module math

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

To access math functions

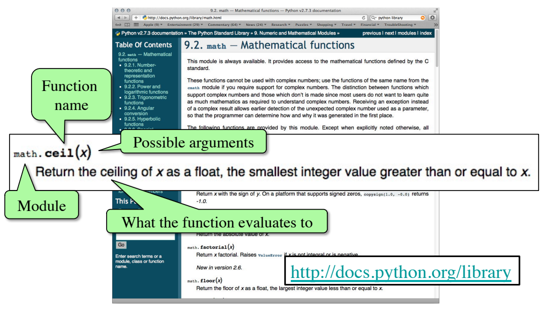
Functions require math prefix!

Module has variables too!

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 the Python Documentation



9.2. math — Mathematical functions — Python v2.7.3 documentation

http://docs.python.org/library/math.html

Python v2.7.3 documentation » The Python Standard Library » 9. Numeric and Mathematical Modules » previous | next | modules | index

Table Of Contents
- 9.2. math — Mathematical functions
  - 9.2.1. Number-theoretic and representation functions
  - 9.2.2. Power and logarithmic functions
  - 9.2.3. Trigonometric functions
  - 9.2.4. Angular conversion
  - 9.2.5. Hyperbolic functions

Go

Enter search terms or a module, class or function name.

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

math.**ceil**(*x*)
Return the ceiling of *x* as a float, the smallest integer value greater than or equal to *x*.

Return *x* with the sign of *y*. On a platform that supports signed zeros, copysign(1.0, -0.0) returns *-1.0*.

Return the absolute value of *x*.

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

New in version 2.6.

math.**floor**(*x*)
Return the floor of *x* as a float, the largest integer value less than or equal to *x*.

http://docs.python.org/library

Using the from Keyword

```
>>> import math
>>> math.pi
3.141592653589793
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import *
>>> cos(pi)
-1.0
```

Must prefix with module name

No prefix needed for variable pi

No prefix needed for anything in math

- Be careful using from!
- Namespaces are *safer*
  - Modules might conflict (functions w/ same name)
  - What if import both?
- **Example**: Turtles
  - Use in Assignment 4
  - 2 modules: turtle, tkturtle
  - Both have func. Turtle()