# What is the following really say? (Using less syntax.)

temp = 88.0

if temp > 95.0:
    print "It's HOT!"
elif temp > 70:
    print "Warm..."
elif temp > 45.0:
    print "cool..."
else:
    print "Brrr!"

# What if we add the following?
if temp <= 95:
    print "I wouldn't say it's hot."
if temp < 89.0:
    print "Nor would you."

# we could replace
#     temp = 88.0
# above with
#     temp = input("Enter temperature: ")
Warm...
I wouldn't say it's hot.
Nor would you.
>>> import random
>>> help random
SyntaxError: invalid syntax
>>> help(random)
Help on module random:

NAME
    random - Random variable generators.

FILE
    /Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/random.py

MODULE DOCS
    http://docs.python.org/library/random

DESCRIPTION
    integers
       ---------
       uniform within range

    sequences
       ---------
       pick random element
       pick random sample
       generate random permutation

    distributions on the real line:
       -----------------------------
       uniform
       triangular
       normal (Gaussian)
       lognormal
       negative exponential
       gamma
       beta
       pareto
       Weibull

    distributions on the circle (angles 0 to 2pi)
      ---------------------------------------------
       circular uniform
       von Mises

General notes on the underlying Mersenne Twister core generator:

* The period is 2**19937-1.
* It is one of the most extensively tested generators in existence.
* Without a direct way to compute N steps forward, the semantics of
  jumpahead(n) are weakened to simply jump to another distant state and rely
  on the large period to avoid overlapping sequences.
* The random() method is implemented in C, executes in a single Python step,
  and is, therefore, threadsafe.

CLASSES
    _random.Random(_builtin_.object)
    Random
class Random(_random.Random):

    Random number generator base class used by bound module functions.

    Used to instantiate instances of Random to get generators that don't
    share state. Especially useful for multi-threaded programs, creating
    a different instance of Random for each thread, and using the jumpahead(
    ) method to ensure that the generated sequences seen by each thread don't
    overlap.

    Class Random can also be subclassed if you want to use a different basic
    generator of your own devising: in that case, override the following
    methods: random(), seed(), getstate(), setstate() and jumpahead().
    Optionally, implement a getrandbits() method so that randrange() can cov
    arbitrarily large ranges.

    Method resolution order:
        Random
        _random.Random
        __builtin__.object

    Methods defined here:

    __getstate__(self)

    __init__(self, x=None)
        Initialize an instance.

        Optional argument x controls seeding, as for Random.seed().

    __reduce__(self)

    __setstate__(self, state)

    betavariate(self, alpha, beta)
        Beta distribution.

        Conditions on the parameters are alpha > 0 and beta > 0.
        Returned values range between 0 and 1.

    choice(self, seq)
        Choose a random element from a non-empty sequence.

    expovariate(self, lambda)
        Exponential distribution.

        lambda is 1.0 divided by the desired mean. It should be
        nonzero. (The parameter would be called "lambda", but that is
        a reserved word in Python.) Returned values range from 0 to
        positive infinity if lambda is positive, and from negative
        infinity to 0 if lambda is negative.

    gammavariate(self, alpha, beta)
        Gamma distribution. Not the gamma function!

        Conditions on the parameters are alpha > 0 and beta > 0.

        The probability distribution function is:

        x ** (alpha - 1) * math.exp(-x / beta)
... http://en.wikipedia.org/wiki/Triangular_distribution

uniform(self, a, b) method of Random instance
Get a random number in the range [a, b) or [a, b] depending on rounding.

vonnivesvariate(self, mu, kappa) method of Random instance
Circular data distribution.

mu is the mean angle, expressed in radians between 0 and 2*pi, and
kappa is the concentration parameter, which must be greater than or
equal to zero. If kappa is equal to zero, this distribution reduces to
a uniform random angle over the range 0 to 2*pi.

weibullvariate(self, alpha, beta) method of Random instance
Weibull distribution.  
alpha is the scale parameter and beta is the shape parameter.

DATA
__all__ = ['Random', 'seed', 'random', 'uniform', 'randint', 'choice',...

>>> help(random.choice)
Help on method choice in module random:

choice(self, seq) method of random.Random instance
Choose a random element from a non-empty sequence.

>>> random.choice(10, 30, 40, 50, 90, 111)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    random.choice(10, 30, 40, 50, 90, 111)
TypeError: choice() takes exactly 2 arguments (7 given)

>>> random.choice([10, 30, 40, 50, 90, 111])
90
>>> random.choice([10, 30, 40, 50, 90, 111])
40
>>> random.choice([10, 30, 40, 50, 90, 111])
90
>>> from random import *
>>> choice([10, 30, 40, 50, 90, 111])
30
>>> ...
user = input("'rock', 'paper' or 'scissors': ")

computer = '?'

if user == 'rock':
    if computer == 'rock':
        print "Tie!"
    elif computer == 'paper':
        print "I Win!"
    elif computer == 'scissors':
        print "you win ..."
    else:
        print "oops..." # an error
elif user == 'paper':
    print "not finished..."
elif user == 'scissors':
    print "not finished..."
else:
    print "oops, you messed up!"