Python: 0-1_knapsack_problem[edit]

Brute force algorithm[edit]

```
from itertools import combinations

def anycomb(items):
    ' return combinations of any length from the items '
    return ( comb
             for r in range(1, len(items)+1)
             for comb in combinations(items, r)
             )

def totalvalue(comb):
    ' Totalise a particular combination of items'
    totwt = totval = 0
    for item, wt, val in comb:
        totwt  += wt
        totval += val
    return (totval, -totwt) if totwt <= 400 else (0, 0)

items = (
    ("map", 9, 150), ("compass", 13, 35), ("water", 153, 200), ("sandwich",
    ("glucose", 15, 60), ("tin", 68, 45), ("banana", 27, 60), ("apple", 39,
    ("cheese", 23, 30), ("beer", 52, 10), ("suntan cream", 11, 70), ("camera
    ("t-shirt", 24, 15), ("trousers", 48, 10), ("umbrella", 73, 40),
    ("waterproof trousers", 42, 70), ("waterproof overclothes", 43, 75),
    ("note-case", 22, 80), ("sunglasses", 7, 20), ("towel", 18, 12),
    ("socks", 4, 50), ("book", 30, 10),
    )
bagged = max( anycomb(items), key=totalvalue) # max val or min wt if values
print("Bagged the following items\n  " +
      '\n  '.join(sorted(item for item,_,_ in bagged)))
val, wt = totalvalue(bagged)
print("for a total value of %i and a total weight of %i" % (val, -wt))
```

Output:

```
Bagged the following items
  banana
  compass
  glucose
  map
  note-case
  sandwich
  socks
  sunglasses
  suntan cream
  water
  waterproof overclothes
  waterproof trousers
for a total value of 1030 and a total weight of 396
```

Dynamic programming solution[edit]

```
try:
    xrange
except:
    xrange = range

def totalvalue(comb):
    ' Totalise a particular combination of items'
    totwt = totval = 0
    for item, wt, val in comb:
        totwt  += wt
        totval += val
    return (totval, -totwt) if totwt <= 400 else (0, 0)

items = (
    ("map", 9, 150), ("compass", 13, 35), ("water", 153, 200), ("sandwich",
    ("glucose", 15, 60), ("tin", 68, 45), ("banana", 27, 60), ("apple", 39,
    ("cheese", 23, 30), ("beer", 52, 10), ("suntan cream", 11, 70), ("camer
    ("t-shirt", 24, 15), ("trousers", 48, 10), ("umbrella", 73, 40),
    ("waterproof trousers", 42, 70), ("waterproof overclothes", 43, 75),
    ("note-case", 22, 80), ("sunglasses", 7, 20), ("towel", 18, 12),
    ("socks", 4, 50), ("book", 30, 10),
    )

def knapsack01_dp(items, limit):
    table = [[0 for w in range(limit + 1)] for j in xrange(len(items) + 1)]

    for j in xrange(1, len(items) + 1):
        item, wt, val = items[j-1]
        for w in xrange(1, limit + 1):
            if wt > w:
                table[j][w] = table[j-1][w]
            else:
                table[j][w] = max(table[j-1][w],
                                  table[j-1][w-wt] + val)

    result = []
    w = limit
    for j in range(len(items), 0, -1):
        was_added = table[j][w] != table[j-1][w]

        if was_added:
            item, wt, val = items[j-1]
            result.append(items[j-1])
            w -= wt

    return result


bagged = knapsack01_dp(items, 400)
print("Bagged the following items\n  " +
      '\n  '.join(sorted(item for item,_,_ in bagged)))
val, wt = totalvalue(bagged)
print("for a total value of %i and a total weight of %i" % (val, -wt))
```

Recursive dynamic programming algorithm[edit]

```
def total_value(items, max_weight):
    return  sum([x[2] for x in items]) if sum([x[1] for x in items]) < max_

cache = {}
def solve(items, max_weight):
    if not items:
        return ()
    if (items,max_weight) not in cache:
        head = items[0]
        tail = items[1:]
        include = (head,) + solve(tail, max_weight - head[1])
        dont_include = solve(tail, max_weight)
        if total_value(include, max_weight) > total_value(dont_include, max_
            answer = include
        else:
            answer = dont_include
        cache[(items,max_weight)] = answer
    return cache[(items,max_weight)]

items = (
    ("map", 9, 150), ("compass", 13, 35), ("water", 153, 200), ("sandwich",
    ("glucose", 15, 60), ("tin", 68, 45), ("banana", 27, 60), ("apple", 39,
    ("cheese", 23, 30), ("beer", 52, 10), ("suntan cream", 11, 70), ("camer
    ("t-shirt", 24, 15), ("trousers", 48, 10), ("umbrella", 73, 40),
    ("waterproof trousers", 42, 70), ("waterproof overclothes", 43, 75),
    ("note-case", 22, 80), ("sunglasses", 7, 20), ("towel", 18, 12),
    ("socks", 4, 50), ("book", 30, 10),
    )
max_weight = 400

solution = solve(items, max_weight)
print "items:"
for x in solution:
    print x[0]
print "value:", total_value(solution, max_weight)
print "weight:", sum([x[1] for x in solution])
```

Python: 2048[edit]

```
#!/usr/bin/env python3

import curses
from random import randrange, choice # generate and place new tile
from collections import defaultdict

letter_codes = [ord(ch) for ch in 'WASDRQwasdrq']
actions = ['Up', 'Left', 'Down', 'Right', 'Restart', 'Exit']
actions_dict = dict(zip(letter_codes, actions * 2))

def get_user_action(keyboard):
```

```
        char = "N"
        while char not in actions_dict:
                char = keyboard.getch()
        return actions_dict[char]

def transpose(field):
        return [list(row) for row in zip(*field)]

def invert(field):
        return [row[::-1] for row in field]

class GameField(object):
        def __init__(self, height=4, width=4, win=2048):
                self.height = height
                self.width = width
                self.win_value = 2048
                self.score = 0
                self.highscore = 0
                self.reset()

        def reset(self):
                if self.score > self.highscore:
                        self.highscore = self.score
                self.score = 0
                self.field = [[0 for i in range(self.width)] for j in range
                self.spawn()
                self.spawn()

        def move(self, direction):
                def move_row_left(row):
                        def tighten(row): # squeese non-zero elements toget
                                new_row = [i for i in row if i != 0]
                                new_row += [0 for i in range(len(row) - len
                                return new_row

                        def merge(row):
                                pair = False
                                new_row = []
                                for i in range(len(row)):
                                        if pair:
                                                new_row.append(2 * row[i])
                                                self.score += 2 * row[i]
                                                pair = False
                                        else:
                                                if i + 1 < len(row) and row
                                                        pair = True
                                                        new_row.append(0)
                                                else:
                                                        new_row.append(row[
                                assert len(new_row) == len(row)
                                return new_row
                        return tighten(merge(tighten(row)))

                moves = {}
                moves['Left']  = lambda field:
                                [move_row_left(row) for row in field]
```

```
        moves['Right'] = lambda field:
                invert(moves['Left'](invert(field)))
        moves['Up']    = lambda field:
                transpose(moves['Left'](transpose(field)))
        moves['Down']  = lambda field:
                transpose(moves['Right'](transpose(field)))

        if direction in moves:
            if self.move_is_possible(direction):
                self.field = moves[direction](self.field)
                self.spawn()
                return True
            else:
                return False

    def is_win(self):
        return any(any(i >= self.win_value for i in row) for row in

    def is_gameover(self):
        return not any(self.move_is_possible(move) for move in acti

    def draw(self, screen):
        help_string1 = '(W)Up (S)Down (A)Left (D)Right'
        help_string2 = '     (R)Restart (Q)Exit'
        gameover_string = '           GAME OVER'
        win_string = '          YOU WIN!'
        def cast(string):
            screen.addstr(string + '\n')

        def draw_hor_separator():
            top = '┌' + ('┬──────' * self.width + '┐')[1:]
            mid = '├' + ('┼──────' * self.width + '┤')[1:]
            bot = '└' + ('┴──────' * self.width + '┘')[1:]
            separator = defaultdict(lambda: mid)
            separator[0], separator[self.height] = top, bot
            if not hasattr(draw_hor_separator, "counter"):
                draw_hor_separator.counter = 0
            cast(separator[draw_hor_separator.counter])
            draw_hor_separator.counter += 1

        def draw_row(row):
            cast(''.join('│{: ^5} '.format(num) if num > 0 else

        screen.clear()
        cast('SCORE: ' + str(self.score))
        if 0 != self.highscore:
            cast('HGHSCORE: ' + str(self.highscore))
        for row in self.field:
            draw_hor_separator()
            draw_row(row)
        draw_hor_separator()
        if self.is_win():
            cast(win_string)
        else:
            if self.is_gameover():
                cast(gameover_string)
```

```
                        else:
                                cast(help_string1)
                cast(help_string2)

        def spawn(self):
                new_element = 4 if randrange(100) > 89 else 2
                (i,j) = choice([(i,j) for i in range(self.width) for j in ra
                self.field[i][j] = new_element

        def move_is_possible(self, direction):
                def row_is_left_movable(row):
                        def change(i): # true if there'll be change in i-th
                                if row[i] == 0 and row[i + 1] != 0: # Move
                                        return True
                                if row[i] != 0 and row[i + 1] == row[i]: # 
                                        return True
                                return False
                        return any(change(i) for i in range(len(row) - 1))

                check = {}
                check['Left']  = lambda field:
                                any(row_is_left_movable(row) for row in fie

                check['Right'] = lambda field:
                                 check['Left'](invert(field))

                check['Up']    = lambda field:
                                check['Left'](transpose(field))

                check['Down']  = lambda field:
                                check['Right'](transpose(field))

                if direction in check:
                        return check[direction](self.field)
                else:
                        return False

def main(stdscr):
        curses.use_default_colors()
        game_field = GameField(win=32)
        state_actions = {} # Init, Game, Win, Gameover, Exit
        def init():
                game_field.reset()
                return 'Game'

        state_actions['Init'] = init

        def not_game(state):
                game_field.draw(stdscr)
                action = get_user_action(stdscr)
                responses = defaultdict(lambda: state)
                responses['Restart'], responses['Exit'] = 'Init', 'Exit'
                return responses[action]

        state_actions['Win'] = lambda: not_game('Win')
        state_actions['Gameover'] = lambda: not_game('Gameover')
```

```
        def game():
                game_field.draw(stdscr)
                action = get_user_action(stdscr)
                if action == 'Restart':
                        return 'Init'
                if action == 'Exit':
                        return 'Exit'
                if game_field.move(action): # move successful
                        if game_field.is_win():
                                return 'Win'
                        if game_field.is_gameover():
                                return 'Gameover'
                return 'Game'

        state_actions['Game'] = game

        state = 'Init'
        while state != 'Exit':
                state = state_actions[state]()

curses.wrapper(main)
```

Ruby[edit]

Python: 24_game_Player[edit]

The function is called **solve**, and is integrated into the game player.

Python: 9_billion_names_of_God_the_integer[edit]

```
cache = [[1]]
def cumu(n):
    for l in range(len(cache), n+1):
        r = [0]
        for x in range(1, l+1):
            r.append(r[-1] + cache[l-x][min(x, l-x)])
        cache.append(r)
    return cache[n]

def row(n):
    r = cumu(n)
    return [r[i+1] - r[i] for i in range(n)]

print "rows:"
for x in range(1, 11): print "%2d:"%x, row(x)
```

```
print "\nsums:"
for x in [23, 123, 1234, 12345]: print x, cumu(x)[-1]
```

Output:

(I didn't actually wait long enough to see what the sum for 12345 is)

```
rows:
 1: [1]
 2: [1, 1]
 3: [1, 1, 1]
 4: [1, 2, 1, 1]
 5: [1, 2, 2, 1, 1]
 6: [1, 3, 3, 2, 1, 1]
 7: [1, 3, 4, 3, 2, 1, 1]
 8: [1, 4, 5, 5, 3, 2, 1, 1]
 9: [1, 4, 7, 6, 5, 3, 2, 1, 1]
10: [1, 5, 8, 9, 7, 5, 3, 2, 1, 1]

sums:
23 1255
123 2552338241
1234 156978797223733228787865722354959930
^C
```

To calculate partition functions only:

```
def partitions(N):
    diffs,k,s = [],1,1
    while k * (3*k-1) < 2*N:
        diffs.extend([(2*k - 1, s), (k, s)])
        k,s = k+1,-s

    out = [1] + [0]*N
    for p in range(0, N+1):
        x = out[p]
        for (o,s) in diffs:
           p += o
           if p > N: break
           out[p] += x*s

    return out

p = partitions(12345)
for x in [23,123,1234,12345]: print x, p[x]
```

This version uses only a fraction of the memory and of the running time, compared to the first one that has to generate all the rows:

Translation of: C

```
def partitions(n):
    partitions.p.append(0)
```

```
    for k in xrange(1, n + 1):
        d = n - k * (3 * k - 1) // 2
        if d < 0:
            break

        if k & 1:
            partitions.p[n] += partitions.p[d]
        else:
            partitions.p[n] -= partitions.p[d]

        d -= k
        if d < 0:
            break

        if k & 1:
            partitions.p[n] += partitions.p[d]
        else:
            partitions.p[n] -= partitions.p[d]

    return partitions.p[-1]

partitions.p = [1]

def main():
    ns = set([23, 123, 1234, 12345])
    max_ns = max(ns)

    for i in xrange(1, max_ns + 1):
        if i > max_ns:
            break
        p = partitions(i)
        if i in ns:
            print "%6d: %s" % (i, p)

main()
```

Output:

```
    23: 1255
   123: 2552338241
  1234: 156978797223733228787865722354959930
 12345: 694203579539261168195629772052093844606676730946714636202703217008
```

Python: A+B[edit]

Console[edit]

In Python 2, `input` returns ints, while `raw_input` returns strings. In Python 3, `input` returns strings, and `raw_input` does not exist.

The first two lines allow the program to be run in either Python 2 or 3. In Python 2, `raw_input` exists, and the lines are effectively skipped. In

Python 3, calling `raw_input` triggers an error, so the `except` loop activates and assigns "raw_input" the value of Python 3's "input" function. Regardless of version, these two lines make sure that `raw_input` will return a string.

```
try: raw_input
except: raw_input = input

print(sum(int(x) for x in raw_input().split()))
```

File[edit]

For Python 2.X and 3.X taking input from stdin stream which can be redirected to be file input under Unix

```
import sys

for line in sys.stdin:
    print(sum(int(i) for i in line.split()))
```

Python: Abstract_type[edit]

```
class BaseQueue(object):
    """Abstract/Virtual Class
    """
    def __init__(self):
        self.contents = list()
        raise NotImplementedError
    def Enqueue(self, item):
        raise NotImplementedError
    def Dequeue(self):
        raise NotImplementedError
    def Print_Contents(self):
        for i in self.contents:
            print i,
```

Python allows multiple inheritance and it's more common to implement "mix-in" classes rather than abstract interfaces. (Mix-in classes can implement functionality as well define interfaces).

In this example we're simply following the Python convention of raising the built-in "NotImplementedError" for each function which must be implemented by our subclasses. This is a "purely virtual" class because all of its methods raise the exception. (It is sufficient for __init__ to do so for any partial virtual abstractions since that still ensures that the exception will be raised if anyone attempts to instantiate the base/abstract class directly rather than one of its concrete (fully implemented) descendents).

The method signatures and the instantiation of a "contents" list shown here can be viewed as documentary hints to anyone inheriting from this class. They won't actually do anything in the derived classes (since these methods must be over-ridden therein).

In this case we've implemented one method (*Print_Contents*). This would be inherited by any derived classes. It could be over-ridden, of course. If it's not over-ridden it establishes a requirement that all derived classes provide some "contents" attribute which must allow for iteration and printing as shown. Without this method the class would be "purely virtual" or "purely abstract." With its inclusion the class becomes "partially implemented."

> **Note:** This "BaseQueue" example should not be confused with Python's standard library Queue class. That is used as the principle "producer/consumer" communications mechanism among threads (and newer *multiprocessing* processes).

Starting from Python 2.6, abstract classes can be created using the standard abc module:

```
from abc import ABCMeta, abstractmethod

class BaseQueue():
    """Abstract Class
    """
    __metaclass__ = ABCMeta

    def __init__(self):
        self.contents = list()

    @abstractmethod
    def Enqueue(self, item):
        pass

    @abstractmethod
    def Dequeue(self):
        pass

    def Print_Contents(self):
        for i in self.contents:
            print i,
```

Python: Abundant,_deficient_and_perfect_number_cl

Importing Proper divisors from prime factors:

```
>>> from proper_divisors import proper_divs
>>> from collections import Counter
>>>
>>> rangemax = 20000
>>>
>>> def pdsum(n):
...     return sum(proper_divs(n))
...
>>> def classify(n, p):
...     return 'perfect' if n == p else 'abundant' if p > n else 'deficient
...
>>> classes = Counter(classify(n, pdsum(n)) for n in range(1, 1 + rangemax)
>>> classes.most_common()
[('deficient', 15043), ('abundant', 4953), ('perfect', 4)]
>>>
```

Output:

```
Between 1 and 20000:
  4953 abundant numbers
  15043 deficient numbers
  4 perfect numbers
```

Python: Accumulator_Factory[edit]

Works with: Python version 2.x/3.x

```
>>> def accumulator(sum):
  def f(n):
    f.sum += n
    return f.sum
  f.sum = sum
  return f

>>> x = accumulator(1)
>>> x(5)
6
>>> x(2.3)
8.3000000000000007
>>> x = accumulator(1)
>>> x(5)
6
>>> x(2.3)
8.3000000000000007
>>> x2 = accumulator(3)
>>> x2(5)
8
>>> x2(3.3)
11.300000000000001
>>> x(0)
8.3000000000000007
>>> x2(0)
11.300000000000001
```

Translation of: Ruby

Works with: Python version 3.x

```
def accumulator(sum):
  def f(n):
    nonlocal sum
    sum += n
    return sum
  return f

x = accumulator(1)
x(5)
print(accumulator(3))
print(x(2.3))
```

Output:

```
<function f at 0xb7c2d0ac>
8.3
```

Works with: Python version 2.5+

```
def accumulator(sum):
  while True:
    sum += yield sum

x = accumulator(1)
x.send(None)
x.send(5)
print(accumulator(3))
print(x.send(2.3))
```

Python: Accumulator_factory[edit]

Works with: Python version 2.x/3.x

```
>>> def accumulator(sum):
  def f(n):
    f.sum += n
    return f.sum
  f.sum = sum
  return f
```

```
>>> x = accumulator(1)
>>> x(5)
6
>>> x(2.3)
8.3000000000000007
>>> x = accumulator(1)
>>> x(5)
6
>>> x(2.3)
8.3000000000000007
>>> x2 = accumulator(3)
>>> x2(5)
8
>>> x2(3.3)
11.300000000000001
>>> x(0)
8.3000000000000007
>>> x2(0)
11.300000000000001
```

Translation of: Ruby

Works with: Python version 3.x

```
def accumulator(sum):
  def f(n):
    nonlocal sum
    sum += n
    return sum
  return f

x = accumulator(1)
x(5)
print(accumulator(3))
print(x(2.3))
```

Output:

```
<function f at 0xb7c2d0ac>
8.3
```

Works with: Python version 2.5+

```
def accumulator(sum):
  while True:
    sum += yield sum
```

```
x = accumulator(1)
x.send(None)
x.send(5)
print(accumulator(3))
print(x.send(2.3))
```

Output:

```
<generator object accumulator at 0x106555e60>
8.3
```

Python: Ackermann_Function[edit]

Works with: Python version 2.5

```
def ack1(M, N):
   return (N + 1) if M == 0 else (
      ack1(M-1, 1) if N == 0 else ack1(M-1, ack1(M, N-1)))
```

Another version:

```
def ack2(M, N):
    if M == 0:
        return N + 1
    elif N == 0:
        return ack2(M - 1, 1)
    else:
        return ack2(M - 1, ack2(M, N - 1))
```

Example of use:

```
>>> import sys
>>> sys.setrecursionlimit(3000)
>>> ack1(0,0)
1
>>> ack1(3,4)
125
>>> ack2(0,0)
1
```

```
>>> ack2(3,4)
125
```

From the Mathematica ack3 example:

```
def ack2(M, N):
   return (N + 1)   if M == 0 else (
          (N + 2)   if M == 1 else (
          (2*N + 3) if M == 2 else (
          (8*(2**N - 1) + 5) if M == 3 else (
          ack2(M-1, 1) if N == 0 else ack2(M-1, ack2(M, N-1))))))
```

Results confirm those of Mathematica for ack(4,1) and ack(4,2)

R[edit]

Python: Ackermann_function[edit]

Works with: Python version 2.5

```
def ack1(M, N):
   return (N + 1) if M == 0 else (
      ack1(M-1, 1) if N == 0 else ack1(M-1, ack1(M, N-1)))
```

Another version:

```
def ack2(M, N):
    if M == 0:
        return N + 1
    elif N == 0:
        return ack2(M - 1, 1)
    else:
        return ack2(M - 1, ack2(M, N - 1))
```

Example of use:

```
>>> import sys
>>> sys.setrecursionlimit(3000)
>>> ack1(0,0)
1
>>> ack1(3,4)
125
>>> ack2(0,0)
1
>>> ack2(3,4)
125
```

From the Mathematica ack3 example:

```
def ack2(M, N):
   return (N + 1)   if M == 0 else (
          (N + 2)   if M == 1 else (
          (2*N + 3) if M == 2 else (
          (8*(2**N - 1) + 5) if M == 3 else (
          ack2(M-1, 1) if N == 0 else ack2(M-1, ack2(M, N-1))))))
```

Results confirm those of Mathematica for ack(4,1) and ack(4,2)

Python: Active_object[edit]

Assignment is thread-safe in Python, so no extra locks are needed in this c

```
from time import time, sleep
from threading import Thread

class Integrator(Thread):
    'continuously integrate a function `K`, at each `interval` seconds'
    def __init__(self, K=lambda t:0, interval=1e-4):
        Thread.__init__(self)
        self.interval  = interval
        self.K   = K
        self.S   = 0.0
        self.__run = True
        self.start()

    def run(self):
        "entry point for the thread"
```

```
        interval = self.interval
        start = time()
        t0, k0 = 0, self.K(0)
        while self.__run:
            sleep(interval)
            t1 = time() - start
            k1 = self.K(t1)
            self.S += (k1 + k0)*(t1 - t0)/2.0
            t0, k0 = t1, k1

    def join(self):
        self.__run = False
        Thread.join(self)

if __name__ == "__main__":
    from math import sin, pi

    ai = Integrator(lambda t: sin(pi*t))
    sleep(2)
    print ai.S
    ai.K = lambda t: 0
    sleep(0.5)
    print ai.S
```

Python: Add_a_variable_to_a_class_instance_at_run

```
class empty(object):
    pass
e = empty()
```

If the variable (attribute) name is known at "compile" time (hard-coded):

```
   e.foo = 1
```

If the variable name is determined at runtime:

```
   setattr(e, name, value)
```

Note: Somewhat counter-intuitively one cannot simply use *e = object(); e.fo*

Because functions are first class objects in Python one can not only add va

```
class empty(object):
    def __init__(this):
        this.foo = "whatever"

def patch_empty(obj):
    def fn(self=obj):
        print self.foo
    obj.print_output = fn

e = empty()
patch_empty(e)
e.print_output()
# >>> whatever
```

Note: The name *self* is not special; it's merely the pervasive Python c

Python: Address_Operations[edit]

Python traditionally doesn't support low-level operations on memory address

The Python *id()* function returns a unique ID for any object. This just happ

```
foo = object()  # Create (instantiate) an empty object
address = id(foo)
```

In addition some folks have written binary Python modules which implement "

Python: Address_of_a_variable[edit]

Python traditionally doesn't support low-level operations on memory address

The Python *id()* function returns a unique ID for any object. This just happ

```
foo = object()  # Create (instantiate) an empty object
address = id(foo)
```

In addition some folks have written binary Python modules which implement "

Python: Aliquot_sequence_classifications[edit]

Importing Proper divisors from prime factors:

```
from proper_divisors import proper_divs
from functools import lru_cache


@lru_cache()
def pdsum(n):
    return sum(proper_divs(n))


def aliquot(n, maxlen=16, maxterm=2**47):
    if n == 0:
        return 'terminating', [0]
    s, slen, new = [n], 1, n
    while slen <= maxlen and new < maxterm:
        new = pdsum(s[-1])
        if new in s:
            if s[0] == new:
                if slen == 1:
                    return 'perfect', s
                elif slen == 2:
                    return 'amicable', s
                else:
                    return 'sociable of length %i' % slen, s
            elif s[-1] == new:
                return 'aspiring', s
            else:
                return 'cyclic back to %i' % new, s
        elif new == 0:
            return 'terminating', s + [0]
        else:
            s.append(new)
            slen += 1
    else:
        return 'non-terminating', s

if __name__ == '__main__':
    for n in range(1, 11):
        print('%s: %r' % aliquot(n))
    print()
    for n in [11, 12, 28, 496, 220, 1184,  12496, 1264460, 790, 909, 562, 1
        print('%s: %r' % aliquot(n))
```

Output:

terminating: [1, 0]

Python: Almost_prime[edit]

This imports Prime decomposition#Python

```
from prime_decomposition import decompose
from itertools import islice, count
try:
    from functools import reduce
except:
    pass


def almostprime(n, k=2):
    d = decompose(n)
    try:
        terms = [next(d) for i in range(k)]
        return reduce(int.__mul__, terms, 1) == n
    except:
        return False

if __name__ == '__main__':
    for k in range(1,6):
        print('%i: %r' % (k, list(islice((n for n in count() if almostprime
```

Output:

```
1: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
2: [4, 6, 9, 10, 14, 15, 21, 22, 25, 26]
3: [8, 12, 18, 20, 27, 28, 30, 42, 44, 45]
4: [16, 24, 36, 40, 54, 56, 60, 81, 84, 88]
5: [32, 48, 72, 80, 108, 112, 120, 162, 168, 176]
```

Racket[edit]

Python: Amb[edit]

(Note: The code is also imported and used as a module in the solution to this task).

Python does not have the amb function, but the declarative style of programm

- Setting ranges
- Setting the constraint
- Iterating over all solutions

can be done in what appears to be a declarative manner with the following c

```
import itertools as _itertools

class Amb(object):
    def __init__(self):
        self._names2values   = {}       # set of values for each global nam
        self._func           = None     # Boolean constraint function
        self._valueiterator  = None     # itertools.product of names values
        self._funcargnames   = None     # Constraint parameter names

    def __call__(self, arg=None):
        if hasattr(arg, '__code__'):
            ##
            ## Called with a constraint function.
            ##
            globls = arg.__globals__ if hasattr(arg, '__globals__') else ar
            # Names used in constraint
            argv = arg.__code__.co_varnames[:arg.__code__.co_argcount]
            for name in argv:
                if name not in self._names2values:
                    assert name in globls, \
                           "Global name %s not found in function globals" %
                    self._names2values[name] = globls[name]
            # Gather the range of values of all names used in the constrain
            valuesets = [self._names2values[name] for name in argv]
            self._valueiterator = _itertools.product(*valuesets)
            self._func = arg
            self._funcargnames = argv
            return self
        elif arg is not None:
            ##
            ## Assume called with an iterable set of values
            ##
            arg = frozenset(arg)
            return arg
        else:
            ##
            ## blank call tries to return next solution
            ##
            return self._nextinsearch()
```

```
    def _nextinsearch(self):
        arg = self._func
        globls = arg.__globals__
        argv = self._funcargnames
        found = False
        for values in self._valueiterator:
            if arg(*values):
                # Set globals.
                found = True
                for n, v in zip(argv, values):
                    globls[n] = v
                break
        if not found: raise StopIteration
        return values

    def __iter__(self):
        return self

    def __next__(self):
        return self()
    next = __next__ # Python 2

if __name__ == '__main__':
    if True:
        amb = Amb()

        print("\nSmall Pythagorean triples problem:")
        x = amb(range(1,11))
        y = amb(range(1,11))
        z = amb(range(1,11))

        for _dummy in amb( lambda x, y, z: x*x + y*y == z*z ):
            print ('%s %s %s' % (x, y, z))


    if True:
        amb = Amb()

        print("\nRosetta Code Amb problem:")
        w1 = amb(["the", "that", "a"])
        w2 = amb(["frog", "elephant", "thing"])
        w3 = amb(["walked", "treaded", "grows"])
        w4 = amb(["slowly", "quickly"])

        for _dummy in amb( lambda w1, w2, w3, w4: \
                             w1[-1] == w2[0] and \
                             w2[-1] == w3[0] and \
                             w3[-1] == w4[0] ):
            print ('%s %s %s %s' % (w1, w2, w3, w4))

    if True:
        amb = Amb()

        print("\nAmb problem from "
            "http://www.randomhacks.net/articles/2005/10/11/amb-operator:")
```

```
        x = amb([1, 2, 3])
        y = amb([4, 5, 6])

        for _dummy in amb( lambda x, y: x * y != 8 ):
            print ('%s %s' % (x, y))
```

Output:

```
Small Pythagorean triples problem:
3 4 5
4 3 5
6 8 10
8 6 10

Rosetta Code Amb problem:
that thing grows slowly

Amb problem from http://www.randomhacks.net/articles/2005/10/11/amb-operato
1 4
1 5
1 6
2 5
2 6
3 4
3 5
3 6
```

Python: Amicable_pairs[edit]

Importing Proper divisors from prime factors:

```
from proper_divisors import proper_divs

def amicable(rangemax=20000):
    n2divsum = {n: sum(proper_divs(n)) for n in range(1, rangemax + 1)}
    for num, divsum in n2divsum.items():
        if num < divsum and divsum <= rangemax and n2divsum[divsum] == num:
            yield num, divsum

if __name__ == '__main__':
    for num, divsum in amicable():
        print('Amicable pair: %i and %i With proper divisors:\n    %r\n    
              % (num, divsum, sorted(proper_divs(num)), sorted(proper_divs(
```

Output:

```
Amicable pair: 220 and 284 With proper divisors:
    [1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110]
    [1, 2, 4, 71, 142]
Amicable pair: 1184 and 1210 With proper divisors:
    [1, 2, 4, 8, 16, 32, 37, 74, 148, 296, 592]
    [1, 2, 5, 10, 11, 22, 55, 110, 121, 242, 605]
Amicable pair: 2620 and 2924 With proper divisors:
    [1, 2, 4, 5, 10, 20, 131, 262, 524, 655, 1310]
    [1, 2, 4, 17, 34, 43, 68, 86, 172, 731, 1462]
Amicable pair: 5020 and 5564 With proper divisors:
    [1, 2, 4, 5, 10, 20, 251, 502, 1004, 1255, 2510]
    [1, 2, 4, 13, 26, 52, 107, 214, 428, 1391, 2782]
Amicable pair: 6232 and 6368 With proper divisors:
    [1, 2, 4, 8, 19, 38, 41, 76, 82, 152, 164, 328, 779, 1558, 3116]
    [1, 2, 4, 8, 16, 32, 199, 398, 796, 1592, 3184]
Amicable pair: 10744 and 10856 With proper divisors:
    [1, 2, 4, 8, 17, 34, 68, 79, 136, 158, 316, 632, 1343, 2686, 5372]
    [1, 2, 4, 8, 23, 46, 59, 92, 118, 184, 236, 472, 1357, 2714, 5428]
Amicable pair: 12285 and 14595 With proper divisors:
    [1, 3, 5, 7, 9, 13, 15, 21, 27, 35, 39, 45, 63, 65, 91, 105, 117, 135,
    [1, 3, 5, 7, 15, 21, 35, 105, 139, 417, 695, 973, 2085, 2919, 4865]
Amicable pair: 17296 and 18416 With proper divisors:
    [1, 2, 4, 8, 16, 23, 46, 47, 92, 94, 184, 188, 368, 376, 752, 1081, 216
    [1, 2, 4, 8, 16, 1151, 2302, 4604, 9208]
```

Racket[edit]

Python: Animate_a_pendulum[edit]

Library: pygame
[edit]

Translation of: C

```
import pygame, sys
from pygame.locals import *
from math import sin, cos, radians

pygame.init()
```

```
WINDOWSIZE = 250
TIMETICK = 100
BOBSIZE = 15

window = pygame.display.set_mode((WINDOWSIZE, WINDOWSIZE))
pygame.display.set_caption("Pendulum")

screen = pygame.display.get_surface()
screen.fill((255,255,255))

PIVOT = (WINDOWSIZE/2, WINDOWSIZE/10)
SWINGLENGTH = PIVOT[1]*4

class BobMass(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.theta = 45
        self.dtheta = 0
        self.rect = pygame.Rect(PIVOT[0]-SWINGLENGTH*cos(radians(self.theta
                                PIVOT[1]+SWINGLENGTH*sin(radians(self.theta
                                1,1)
        self.draw()

    def recomputeAngle(self):
        scaling = 3000.0/(SWINGLENGTH**2)

        firstDDtheta = -sin(radians(self.theta))*scaling
        midDtheta = self.dtheta + firstDDtheta
        midtheta = self.theta + (self.dtheta + midDtheta)/2.0

        midDDtheta = -sin(radians(midtheta))*scaling
        midDtheta = self.dtheta + (firstDDtheta + midDDtheta)/2
        midtheta = self.theta + (self.dtheta + midDtheta)/2

        midDDtheta = -sin(radians(midtheta)) * scaling
        lastDtheta = midDtheta + midDDtheta
        lasttheta = midtheta + (midDtheta + lastDtheta)/2.0

        lastDDtheta = -sin(radians(lasttheta)) * scaling
        lastDtheta = midDtheta + (midDDtheta + lastDDtheta)/2.0
        lasttheta = midtheta + (midDtheta + lastDtheta)/2.0

        self.dtheta = lastDtheta
        self.theta = lasttheta
        self.rect = pygame.Rect(PIVOT[0]-
                                SWINGLENGTH*sin(radians(self.theta)),
                                PIVOT[1]+
                                SWINGLENGTH*cos(radians(self.theta)),1,1)


    def draw(self):
        pygame.draw.circle(screen, (0,0,0), PIVOT, 5, 0)
        pygame.draw.circle(screen, (0,0,0), self.rect.center, BOBSIZE, 0)
        pygame.draw.aaline(screen, (0,0,0), PIVOT, self.rect.center)
        pygame.draw.line(screen, (0,0,0), (0, PIVOT[1]), (WINDOWSIZE, PIVOT
```

```
    def update(self):
        self.recomputeAngle()
        screen.fill((255,255,255))
        self.draw()

bob = BobMass()

TICK = USEREVENT + 2
pygame.time.set_timer(TICK, TIMETICK)

def input(events):
    for event in events:
        if event.type == QUIT:
            sys.exit(0)
        elif event.type == TICK:
            bob.update()

while True:
    input(pygame.event.get())
    pygame.display.flip()
```

Python: Animation[edit]

Using pygame[edit]

Library: pygame

```
import pygame, sys
from pygame.locals import *
pygame.init()

YSIZE = 40
XSIZE = 150

TEXT = "Hello World! "
FONTSIZE = 32

LEFT = False
RIGHT = True

DIR = RIGHT

TIMETICK = 180
TICK = USEREVENT + 2

TEXTBOX = pygame.Rect(10,10,XSIZE,YSIZE)

pygame.time.set_timer(TICK, TIMETICK)
```

```
window = pygame.display.set_mode((XSIZE, YSIZE))
pygame.display.set_caption("Animation")

font = pygame.font.SysFont(None, FONTSIZE)
screen = pygame.display.get_surface()

def rotate():
    index = DIR and -1 or 1
    global TEXT
    TEXT = TEXT[index:]+TEXT[:index]

def click(position):
    if TEXTBOX.collidepoint(position):
        global DIR
        DIR = not DIR

def draw():
    surface = font.render(TEXT, True, (255,255,255), (0,0,0))
    global TEXTBOX
    TEXTBOX = screen.blit(surface, TEXTBOX)

def input(event):
    if event.type == QUIT:
        sys.exit(0)
    elif event.type == MOUSEBUTTONDOWN:
        click(event.pos)
    elif event.type == TICK:
        draw()
        rotate()

while True:
    input(pygame.event.wait())
    pygame.display.flip()
```

Python: Anonymous_recursion[edit]

```
>>> Y = lambda f: (lambda x: x(x))(lambda y: f(lambda *args: y(y)(*args)))
>>> fib = lambda f: lambda n: None if n < 0 else (0 if n == 0 else (1 if n =
>>> [ Y(fib)(i) for i in range(-2, 10) ]
[None, None, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Same thing as the above, but modified so that the function is uncurried:

```
>>>from functools import partial
>>> Y = lambda f: (lambda x: x(x))(lambda y: partial(f, lambda *args: y(y)(
>>> fib = lambda f, n: None if n < 0 else (0 if n == 0 else (1 if n == 1 el
>>> [ Y(fib)(i) for i in range(-2, 10) ]
[None, None, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

A different approach: the function always receives itself as the first argu

Python: Append_a_record_to_the_end_of_a_text_file

Python: Apply_a_callback_to_an_Array[edit]

```
def square(n):
    return n * n

numbers = [1, 3, 5, 7]

squares1 = [square(n) for n in numbers]     # list comprehension

squares2a = map(square, numbers)            # functional form

squares2b = map(lambda x: x*x, numbers)     # functional form with `lambda`

squares3 = [n * n for n in numbers]         # no need for a function,
                                            # anonymous or otherwise

isquares1 = (n * n for n in numbers)        # iterator, lazy

import itertools
isquares2 = itertools.imap(square, numbers) # iterator, lazy
```

To print squares of integers in the range from 0 to 9, type:

```
print " ".join(str(n * n) for n in range(10))
```

Or:

```
print " ".join(map(str, map(square, range(10))))
```

Result:

```
0 1 4 9 16 25 36 49 64 81
```

Python: Apply_a_callback_to_an_array[edit]

```
def square(n):
    return n * n

numbers = [1, 3, 5, 7]

squares1 = [square(n) for n in numbers]     # list comprehension

squares2a = map(square, numbers)            # functional form

squares2b = map(lambda x: x*x, numbers)     # functional form with `lambda`
```

```
squares3 = [n * n for n in numbers]         # no need for a function,
                                            # anonymous or otherwise

isquares1 = (n * n for n in numbers)        # iterator, lazy

import itertools
isquares2 = itertools.imap(square, numbers) # iterator, lazy
```

To print squares of integers in the range from 0 to 9, type:

```
print " ".join(str(n * n) for n in range(10))
```

Or:

```
print " ".join(map(str, map(square, range(10))))
```

Result:

```
0 1 4 9 16 25 36 49 64 81
```

Python: Arbitrary-precision_integers_(included)[e

Python comes with built-in support for arbitrary precision integers. The ty

```
>>> y = str( 5**4**3**2 )
>>> print ("5**4**3**2 = %s...%s and has %i digits" % (y[:20], y[-20:], len
5**4**3**2 = 62060698786608744707...92256259918212890625 and has 183231 dig
```

R[edit]

Python: Arena_storage_pool[edit]

In Python:

- Everything is an object.

Python: Array_concatenation[edit]

The + operator concatenates two lists and returns a new list.
The list.extend method appends elements of another list to the receive

```
arr1 = [1, 2, 3]
arr2 = [4, 5, 6]
arr3 = [7, 8, 9]
arr4 = arr1 + arr2
assert arr4 == [1, 2, 3, 4, 5, 6]
arr4.extend(arr3)
assert arr4 == [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Note: list.extend is normally accomplished using the += operator like

```
arr5 = [4, 5, 6]
arr6 = [7, 8, 9]
arr6 += arr5
assert arr6 == [7, 8, 9, 4, 5, 6]
```

Python: Array_length[edit]

```
>>> print(len(['apple', 'orange']))
2
>>>
```

R[edit]

Python: Arrays[edit]

Python lists are dynamically resizeable.

```
array = []

array.append(1)
array.append(3)

array[0] = 2

print array[0]
```

A simple, single-dimensional array can also be initialized thus:

```
myArray = [0] * size
```

However this will not work as intended if one tries to generalize from

```
myArray = [[0]* width] * height] # DOES NOT WORK AS INTENDED!!!
```

This creates a list of "height" number of references to one list objec

To initialize a list of lists one could use a pair of nested list comp

```
myArray = [[0 for x in range(width)] for y in range(height)]
```

That is equivalent to:

```
myArray = list()
for x in range(height):
   myArray.append([0] * width)
```

To retrieve an element in an array, use any of the following methods:

```
# Retrieve an element directly from the array.
item = array[index]

# Use the array like a stack.  Note that using the pop() method remove
array.pop()  # Pop last item in a list
array.pop(0) # Pop first item in a list

# Using a negative element counts from the end of the list.
item = array[-1] # Retrieve last element in a list.
```

Python produces an IndexError when accessing elements out of range:

```
try:
    # This will cause an exception, which will then be caught.
    print array[len(array)]
except IndexError as e:
    # Print the exception.
    print e
```

Python: Aspect_Oriented_Programming[edit]

Python has special syntax for decorators acting on functions and metho

Tcl[edit]

Python: Assertions[edit]

```
a = 5
#...input or change a here
assert a == 42 # throws an AssertionError when a is not 42
assert a == 42, "Error message" # throws an AssertionError
       # when a is not 42 with "Error message" for the message
       # the error message can be any expression
```

It is possible to turn off assertions by running Python with the -O (o

Python: Assigning_Values_to_an_Array[edit]

To change existing item, (raise IndexError if the index does not exist

```
 array[index] = value
```

To append to the end of the array:

```
 array.append(value)
```

It's also possible modify Python lists using "slices" which can replac

```
mylist = [0,1,2,3]
mylist[1:3] = [1, 1.2, 1.3]
print mylist
## >>> [0, 1, 1.2, 1.3, 3]
## We've replaced 1 and 2 with 1, 1.2 and 1.3, effectively inserting
```

Hint: slice notation should be read as: "from starting index **up to** (bu

It's even possible (though obscure) to use extended slices with a "str

```
mylist = [0,1,2,3]
mylist[0:4:2] = ['x', 'y']   # can also be written as mylist[::2] in t
print mylist
## >>> ['x', 1, 'y', 3]
```

Python lists also support *.insert(),* and *.remove()* methods, for cases

```
mylist = [0,1]
mylist.extend([2,3])
print mylist
## >>> [0, 1, 2, 3]
## mylist.append([2,3]) would have appended one item to the list
## ... and that item would have been list containing two nested items
```

Python: Atomic_updates[edit]

Works with: Python version 2.5 and above

This code uses a *threading.Lock* to serialize access to the bucket set.

```
from __future__ import with_statement # required for Python 2.5
import threading
```

```
import random
import time

terminate = threading.Event()

class Buckets:
    def __init__(self, nbuckets):
        self.nbuckets = nbuckets
        self.values = [random.randrange(10) for i in range(nbuckets)]
        self.lock = threading.Lock()

    def __getitem__(self, i):
        return self.values[i]

    def transfer(self, src, dst, amount):
        with self.lock:
            amount = min(amount, self.values[src])
            self.values[src] -= amount
            self.values[dst] += amount

    def snapshot(self):
        # copy of the current state (synchronized)
        with self.lock:
            return self.values[:]

def randomize(buckets):
    nbuckets = buckets.nbuckets
    while not terminate.isSet():
        src = random.randrange(nbuckets)
        dst = random.randrange(nbuckets)
        if dst!=src:
            amount = random.randrange(20)
            buckets.transfer(src, dst, amount)

def equalize(buckets):
    nbuckets = buckets.nbuckets
    while not terminate.isSet():
        src = random.randrange(nbuckets)
        dst = random.randrange(nbuckets)
        if dst!=src:
            amount = (buckets[src] - buckets[dst]) // 2
            if amount>=0: buckets.transfer(src, dst, amount)
            else: buckets.transfer(dst, src, -amount)

def print_state(buckets):
    snapshot = buckets.snapshot()
    for value in snapshot:
        print '%2d' % value,
    print '=', sum(snapshot)

# create 15 buckets
buckets = Buckets(15)

# the randomize thread
t1 = threading.Thread(target=randomize, args=[buckets])
t1.start()
```

```
# the equalize thread
t2 = threading.Thread(target=equalize, args=[buckets])
t2.start()

# main thread, display
try:
    while True:
        print_state(buckets)
        time.sleep(1)
except KeyboardInterrupt: # ^C to finish
    terminate.set()

# wait until all worker threads finish
t1.join()
t2.join()
```

Sample Output:

```
 5  5 11  5  5  5  5  5  5  0  6  5  5  6  5 = 78
 9  0  0  0 20  5  0 21 10  0  0  8  5  0  0 = 78
 4  0  4 12  4  4  9  2 14  0 11  2  0 12  0 = 78
 5  5  6  5  5  5  6  5  6  5  5  5  5  5  5 = 78
 2  0  3  0  0  0  0  4 13  4  9  0  1  9 33 = 78
 0  0  0 22 11  0 13 12  0  0  0 20  0  0  0 = 78
```

Python: AudioAlarm[edit]

Python natively supports playing .wav files (via the wave library), bu

Works with: Python version 3.4.1

```
import time
import os

seconds = input("Enter a number of seconds: ")
sound = input("Enter an mp3 filename: ")

time.sleep(float(seconds))
os.startfile(sound + ".mp3")
```

Racket[edit]

Python: Average_loop_length[edit]

Translation of: C

```
from __future__ import division # Only necessary for Python 2.X
from math import factorial
from random import randrange

MAX_N = 20
TIMES = 1000000

def analytical(n):
        return sum(factorial(n) / pow(n, i) / factorial(n -i) for i in

def test(n, times):
    count = 0
    for i in range(times):
        x, bits = 1, 0
        while not (bits & x):
            count += 1
            bits |= x
            x = 1 << randrange(n)
    return count / times

if __name__ == '__main__':
    print(" n\tavg\texp.\tdiff\n-------------------------------")
    for n in range(1, MAX_N+1):
        avg = test(n, TIMES)
        theory = analytical(n)
        diff = (avg / theory - 1) * 100
        print("%2d %8.4f %8.4f %6.3f%%" % (n, avg, theory, diff))
```

Output:

```
 n      avg     exp.    diff
-------------------------------
 1   1.0000   1.0000  0.000%
 2   1.5006   1.5000  0.037%
 3   1.8887   1.8889 -0.012%
 4   2.2190   2.2188  0.011%
 5   2.5101   2.5104 -0.012%
 6   2.7750   2.7747  0.012%
 7   3.0158   3.0181 -0.076%
 8   3.2447   3.2450 -0.009%
 9   3.4586   3.4583  0.009%
10   3.6598   3.6602 -0.010%
11   3.8510   3.8524 -0.036%
```

```
12   4.0368   4.0361  0.017%
13   4.2099   4.2123 -0.058%
14   4.3784   4.3820 -0.083%
15   4.5484   4.5458  0.058%
16   4.7045   4.7043  0.006%
17   4.8611   4.8579  0.067%
18   5.0074   5.0071  0.007%
19   5.1534   5.1522  0.024%
20   5.2927   5.2936 -0.017%
```

Python: Bacon_cipher[edit]

This deviates from the Bacon method as it encodes to different capital

```
import string

sometext = """All children, except one, grow up. They soon know that t
up, and the way Wendy knew was this. One day when she was two years ol
she was playing in a garden, and she plucked another flower and ran wi
it to her mother. I suppose she must have looked rather delightful, fo
Mrs. Darling put her hand to her heart and cried, "Oh, why can't you
remain like this for ever!" This was all that passed between them on
the subject, but henceforth Wendy knew that she must grow up. You alwa
know after you are two. Two is the beginning of the end.

Of course they lived at 14 [their house number on their street], and
until Wendy came her mother was the chief one. She was a lovely lady,
with a romantic mind and such a sweet mocking mouth. Her romantic
mind was like the tiny boxes, one within the other, that come from the
puzzling East, however many you discover there is always one more; and
her sweet mocking mouth had one kiss on it that Wendy could never get,
though there it was, perfectly conspicuous in the right-hand corner.""

lc2bin = {ch: '{:05b}'.format(i)
          for i, ch in enumerate(string.ascii_lowercase + ' .')}
bin2lc = {val: key for key, val in lc2bin.items()}

phrase = 'Rosetta code Bacon cipher example secret phrase to encode in

def to_5binary(msg):
    return ( ch == '1' for ch in ''.join(lc2bin.get(ch, '') for ch in

def encrypt(message, text):
    bin5 = to_5binary(message)
    textlist = list(text.lower())
    out = []
    for capitalise in bin5:
        while textlist:
            ch = textlist.pop(0)
```

```
            if ch.isalpha():
                if capitalise:
                    ch = ch.upper()
                out.append(ch)
                break
            else:
                out.append(ch)
        else:
            raise Exception('ERROR: Ran out of characters in sometext'
    return ''.join(out) + '...'


def  decrypt(bacontext):
    binary = []
    bin5 = []
    out = []
    for ch in bacontext:
        if ch.isalpha():
            binary.append('1' if ch.isupper() else '0')
            if len(binary) == 5:
                bin5 = ''.join(binary)
                out.append(bin2lc[bin5])
                binary = []
    return ''.join(out)


print('PLAINTEXT = \n%s\n' % phrase)
encrypted = encrypt(phrase, sometext)
print('ENCRYPTED = \n%s\n' % encrypted)
decrypted = decrypt(encrypted)
print('DECRYPTED = \n%s\n' % decrypted)
assert phrase == decrypted, 'Round-tripping error'
```

Output:

```
PLAINTEXT =
rosetta code bacon cipher example secret phrase to encode in the capit

ENCRYPTED =
All cHiLDReN, exCept One, GroW UP. thEY soon kNOw That tHey WILl groW
Up, aNd tHE wAy wendY knew was tHis. ONE daY WhEN ShE was tWo yEars oL
SHe wAS PlaYinG in a GARdEn, anD shE pLUCked anoTHer fLOWEr AnD Ran Wi
It To Her MothEr. i supPoSe shE muSt hAve LOOKeD raTHER deLIGHtfuL, fo
mrS. daRlinG puT HeR hAnd TO hER HeARt And cRied, "OH, wHy caN't yOU
RemaiN LikE thIS fOr eVer!" thIS wAS AlL tHat PAssED BetWeeN ThEm on
tHe subjecT, BUT hEnceForTH wendy kNeW ThAt shE muSt grow uP. yoU AlWa
kNOW afTEr YOU aRe tWO. Two iS tHE BeGinNING of The End.

OF coUrsE theY LIvEd aT 14 [THEir housE NuM...

DECRYPTED =
```

rosetta code bacon cipher example secret phrase to encode in the capit

Python: Balanced_brackets[edit]

```
>>> def gen(N):
...     txt = ['[', ']'] * N
...     random.shuffle( txt )
...     return ''.join(txt)
...
>>> def balanced(txt):
...     braced = 0
...     for ch in txt:
...         if ch == '[': braced += 1
...         if ch == ']':
...             braced -= 1
...             if braced < 0: return False
...     return braced == 0
...
>>> for txt in (gen(N) for N in range(10)):
...     print ("%-22r is%s balanced" % (txt, '' if balanced(txt) else
...
''                       is balanced
'[]'                     is balanced
'[][]'                   is balanced
'][[[]]'                 is not balanced
'[]][[][]'               is not balanced
'[][[][]]]['             is not balanced
'][]][][[]][['           is not balanced
'[[]]]]][]][[[['         is not balanced
'[[[[]][]]][[][]]'       is balanced
'][[][[]]][]]][[[[]'     is not balanced
```

Qi[edit]

Python: Balanced_ternary[edit]

Translation of: CommonLisp

Python: Base64_encode_data[edit]

```
import urllib
```

```
import base64

data = urllib.urlopen('http://rosettacode.org/favicon.ico').read()
print base64.b64encode(data)
```

(For me this gets the wrong data; the data is actually an error messag

Python: Basic_Animation[edit]

Using pygame[edit]

Library: pygame

```
import pygame, sys
from pygame.locals import *
pygame.init()

YSIZE = 40
XSIZE = 150

TEXT = "Hello World! "
FONTSIZE = 32

LEFT = False
RIGHT = True

DIR = RIGHT

TIMETICK = 180
TICK = USEREVENT + 2

TEXTBOX = pygame.Rect(10,10,XSIZE,YSIZE)

pygame.time.set_timer(TICK, TIMETICK)

window = pygame.display.set_mode((XSIZE, YSIZE))
pygame.display.set_caption("Animation")

font = pygame.font.SysFont(None, FONTSIZE)
screen = pygame.display.get_surface()

def rotate():
    index = DIR and -1 or 1
    global TEXT
    TEXT = TEXT[index:]+TEXT[:index]

def click(position):
```

```
    if TEXTBOX.collidepoint(position):
        global DIR
        DIR = not DIR

def draw():
    surface = font.render(TEXT, True, (255,255,255), (0,0,0))
    global TEXTBOX
    TEXTBOX = screen.blit(surface, TEXTBOX)

def input(event):
    if event.type == QUIT:
        sys.exit(0)
    elif event.type == MOUSEBUTTONDOWN:
        click(event.pos)
    elif event.type == TICK:
        draw()
        rotate()

while True:
    input(pygame.event.wait())
    pygame.display.flip()
```

Using Tkinter[edit]

```
import Tkinter as tki

def scroll_text(s, how_many):
    return s[how_many:] + s[:how_many]

direction = 1
tk = tki.Tk()
var = tki.Variable(tk)

def mouse_handler(point):
    global direction
    direction *= -1

def timer_handler():
    var.set(scroll_text(var.get(),direction))
    tk.after(125, timer_handler)

var.set('Hello, World! ')
tki.Label(tk, textvariable=var).pack()
tk.bind("<Button-1>", mouse_handler)
tk.after(125, timer_handler)
tk.title('Python Animation')
tki.mainloop()
```

Python: Basic_bitmap_storage.1[edit]

See Basic bitmap storage/Python

R[edit]

Python: Basic_input_loop[edit]

Python file objects can be iterated like lists:

Python: Basic_string_manipulation_functions[ed

2.x[edit]

Python 2.x's string type (str) is a native byte string type. They can

- String creation

```
s1 = "A 'string' literal \n"
s2 = 'You may use any of \' or " as delimiter'
s3 = """This text
   goes over several lines
       up to the closing triple quote"""
```

- String assignment

There is nothing special about assignments:

```
s = "Hello "
t = "world!"
u = s + t   # + concatenates
```

- String comparison

They're compared byte by byte, lexicographically:

```
assert "Hello" == 'Hello'
assert '\t' == '\x09'
assert "one" < "two"
assert "two" >= "three"
```

- String cloning and copying

Strings are immutable, so there is no need to clone/copy them. If you

- Check if a string is empty

```
if x=='': print "Empty string"
if not x: print "Empty string, provided you know x is a string"
```

- Append a byte to a string

```
txt = "Some text"
txt += '\x07'
# txt refers now to a new string having "Some text\x07"
```

- Extract a substring from a string

Strings are sequences, they can be indexed with s[index] (index is 0-b

```
txt = "Some more text"
assert txt[4] == " "
assert txt[0:4] == "Some"
assert txt[:4] == "Some" # you can omit the starting index if 0
assert txt[5:9] == "more"
assert txt[5:] == "more text" # omitting the second index means "to th
```

Negative indexes count from the end: -1 is the last byte, and so on:

```
txt = "Some more text"
assert txt[-1] == "t"
assert txt[-4:] == "text"
```

- Replace every occurrence of a byte (or a string) in a string wit

Strings are objects and have methods, like replace:

```
v1 = "hello world"
v2 = v1.replace("l", "L")
print v2 # prints heLLo worLd
```

- Join strings

If they're separate variables, use the + operator:

```
v1 = "hello"
v2 = "world"
msg = v1 + " " + v2
```

If the elements to join are contained inside any iterable container (e

```
items = ["Smith", "John", "417 Evergreen Av", "Chimichurri", "481-3172
joined = ",".join(items)
print joined
# output:
# Smith,John,417 Evergreen Av,Chimichurri,481-3172
```

The reverse operation (split) is also possible:

```
line = "Smith,John,417 Evergreen Av,Chimichurri,481-3172"
fields = line.split(',')
print fields
# output:
# ['Smith', 'John', '417 Evergreen Av', 'Chimichurri', '481-3172']
```

3.x[edit]

Python 3.x has two binary string types: bytes (immutable) and bytearra

To specify a literal immutable byte string (bytes), prefix a string li

```
s1 = b"A 'byte string' literal \n"
s2 = b'You may use any of \' or " as delimiter'
s3 = b"""This text
   goes over several lines
       up to the closing triple quote"""
```

You can use the normal string escape sequences to encode special bytes

Indexing a byte string results in an integer (the byte value at that b

```
x = b'abc'
x[0] # evaluates to 97
```

Similarly, a byte string can be converted to and from a list of intege

```
x = b'abc'
list(x) # evaluates to [97, 98, 99]
bytes([97, 98, 99]) # evaluates to b'abc'
```

Python: Benford's_law[edit]

Works with Python 3.X & 2.7

```
from __future__ import division
from itertools import islice, count
from collections import Counter
from math import log10
from random import randint

expected = [log10(1+1/d) for d in range(1,10)]

def fib():
    a,b = 1,1
    while True:
        yield a
        a,b = b,a+b

# powers of 3 as a test sequence
def power_of_threes():
    return (3**k for k in count(0))

def heads(s):
    for a in s: yield int(str(a)[0])
```

```
def show_dist(title, s):
    c = Counter(s)
    size = sum(c.values())
    res = [c[d]/size for d in range(1,10)]

    print("\n%s Benfords deviation" % title)
    for r, e in zip(res, expected):
        print("%5.1f%% %5.1f%%  %5.1f%%" % (r*100., e*100., abs(r - e)

def rand1000():
    while True: yield randint(1,9999)

if __name__ == '__main__':
    show_dist("fibbed", islice(heads(fib()), 1000))
    show_dist("threes", islice(heads(power_of_threes()), 1000))

    # just to show that not all kind-of-random sets behave like that
    show_dist("random", islice(heads(rand1000()), 10000))
```

Output:

```
fibbed Benfords deviation
 30.1%  30.1%    0.0%
 17.7%  17.6%    0.1%
 12.5%  12.5%    0.0%
  9.6%   9.7%    0.1%
  8.0%   7.9%    0.1%
  6.7%   6.7%    0.0%
  5.6%   5.8%    0.2%
  5.3%   5.1%    0.2%
  4.5%   4.6%    0.1%

threes Benfords deviation
 30.0%  30.1%    0.1%
 17.7%  17.6%    0.1%
 12.3%  12.5%    0.2%
  9.8%   9.7%    0.1%
  7.9%   7.9%    0.0%
  6.6%   6.7%    0.1%
  5.9%   5.8%    0.1%
  5.2%   5.1%    0.1%
  4.6%   4.6%    0.0%

random Benfords deviation
 11.2%  30.1%   18.9%
 10.9%  17.6%    6.7%
 11.6%  12.5%    0.9%
 11.1%   9.7%    1.4%
 11.6%   7.9%    3.7%
 11.4%   6.7%    4.7%
```

```
10.3%   5.8%    4.5%
11.0%   5.1%    5.9%
10.9%   4.6%    6.3%
```

Python: Best_shuffle[edit]

Swap if it is locally better algorithm[edit]

With added randomization of swaps!

```
import random

def count(w1,wnew):
    return sum(c1==c2 for c1,c2 in zip(w1, wnew))

def best_shuffle(w):
    wnew = list(w)
    n = len(w)
    rangelists = (list(range(n)), list(range(n)))
    for r in rangelists:
        random.shuffle(r)
    rangei, rangej = rangelists
    for i in rangei:
        for j in rangej:
            if i != j and wnew[j] != wnew[i] and w[i] != wnew[j] and w
                wnew[j], wnew[i] = wnew[i], wnew[j]
                break
    wnew = ''.join(wnew)
    return wnew, count(w, wnew)


if __name__ == '__main__':
    test_words = ('tree abracadabra seesaw elk grrrrrr up a '
                  + 'antidisestablishmentarianism hounddogs').split()
    test_words += ['aardvarks are ant eaters', 'immediately', 'abba']
    for w in test_words:
        wnew, c = best_shuffle(w)
        print("%29s, %-29s ,(%i)" % (w, wnew, c))
```

Sample output

Two runs showing variability in shuffled results

```
>>> ================================ RESTART ================================
>>>
                        tree, eetr                                ,(0)
                 abracadabra, daaracbraab                         ,(0)
                      seesaw, asswee                              ,(0)
                         elk, kel                                 ,(0)
                     grrrrrr, rrgrrrr                             ,(5)
                          up, pu                                  ,(0)
                           a, a                                   ,(1)
 antidisestablishmentarianism, sintmdnirhimasibtnasetaisael       ,(0)
                   hounddogs, ohodgnsud                           ,(0)
     aardvarks are ant eaters, sesanretatva kra errada            ,(0)
                 immediately, tedlyaeiimm                         ,(0)
                        abba, baab                                ,(0)
>>> ================================ RESTART ================================
>>>
                        tree, eert                                ,(0)
                 abracadabra, bdacararaab                         ,(0)
                      seesaw, ewsase                              ,(0)
                         elk, kel                                 ,(0)
                     grrrrrr, rrrrrrg                             ,(5)
                          up, pu                                  ,(0)
                           a, a                                   ,(1)
 antidisestablishmentarianism, rtitiainnnshtmdesibalassemai       ,(0)
                   hounddogs, ddousngoh                           ,(0)
     aardvarks are ant eaters, sretrnat a edseavra akar           ,(0)
                 immediately, litiaemmyed                         ,(0)
                        abba, baab                                ,(0)
>>>
```

Alternative algorithm #1[edit]

```
#!/usr/bin/env python

def best_shuffle(s):
    # Count the supply of characters.
    from collections import defaultdict
    count = defaultdict(int)
    for c in s:
        count[c] += 1

    # Shuffle the characters.
    r = []
    for x in s:
        # Find the best character to replace x.
        best = None
        rankb = -2
        for c, rankc in count.items():
            # Prefer characters with more supply.
            # (Save characters with less supply.)
            # Avoid identical characters.
            if c == x: rankc = -1
```

```
            if rankc > rankb:
                best = c
                rankb = rankc

        # Add character to list. Remove it from supply.
        r.append(best)
        count[best] -= 1
        if count[best] >= 0: del count[best]

    # If the final letter became stuck (as "ababcd" became "bacabd",
    # and the final "d" became stuck), then fix it.
    i = len(s) - 1
    if r[i] == s[i]:
        for j in range(i):
            if r[i] != s[j] and r[j] != s[i]:
                r[i], r[j] = r[j], r[i]
                break

    # Convert list to string. PEP 8, "Style Guide for Python Code",
    # suggests that ''.join() is faster than + when concatenating
    # many strings. See http://www.python.org/dev/peps/pep-0008/
    r = ''.join(r)

    score = sum(x == y for x, y in zip(r, s))

    return (r, score)

for s in "abracadabra", "seesaw", "elk", "grrrrrr", "up", "a":
    shuffled, score = best_shuffle(s)
    print("%s, %s, (%d)" % (s, shuffled, score))
```

Output:

```
abracadabra, raabarabacd, (0)
seesaw, wsaese, (0)
elk, kel, (0)
grrrrrr, rgrrrrr, (5)
up, pu, (0)
a, a, (1)
```

Python: Beyond_ASCII_variable_names[edit]

Within the ASCII range (U+0001..U+007F), the valid characters for iden

Python 3.0 introduces additional characters from outside the ASCII ran

Identifiers are unlimited in length. Case is significant.

```
>>> Δx = 1
```

```
>>> Δx += 1
>>> print(Δx)
2
>>>
```

Python: Binary_digits[edit]

Works with: Python version 3.X and 2.6+

```
>>> for i in range(16): print('{0:b}'.format(i))

0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
```

Works with: Python version 3.X and 2.6+

```
>>> for i in range(16): print(bin(i))

0b0
0b1
0b10
0b11
0b100
0b101
0b110
0b111
0b1000
0b1001
0b1010
0b1011
0b1100
```

```
0b1101
0b1110
0b1111
```

Pre-Python 2.6:

```
>>> oct2bin = {'0': '000', '1': '001', '2': '010', '3': '011', '4': '1
>>> bin = lambda n: ''.join(oct2bin[octdigit] for octdigit in '%o' % n
>>> for i in range(16): print(bin(i))

0
1
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111
```

Python: Binary_string_manipulation_functions[e

2.x[edit]

Python 2.x's string type (str) is a native byte string type. They can

- String creation

```
s1 = "A 'string' literal \n"
s2 = 'You may use any of \' or " as delimiter'
s3 = """This text
   goes over several lines
       up to the closing triple quote"""
```

- String assignment

There is nothing special about assignments:

```
s = "Hello "
t = "world!"
u = s + t   # + concatenates
```

- String comparison

They're compared byte by byte, lexicographically:

```
assert "Hello" == 'Hello'
assert '\t' == '\x09'
assert "one" < "two"
assert "two" >= "three"
```

- String cloning and copying

Strings are immutable, so there is no need to clone/copy them. If you

- Check if a string is empty

```
if x=='': print "Empty string"
if not x: print "Empty string, provided you know x is a string"
```

- Append a byte to a string

```
txt = "Some text"
txt += '\x07'
# txt refers now to a new string having "Some text\x07"
```

- Extract a substring from a string

Strings are sequences, they can be indexed with s[index] (index is 0-b

```
txt = "Some more text"
assert txt[4] == " "
assert txt[0:4] == "Some"
assert txt[:4] == "Some" # you can omit the starting index if 0
assert txt[5:9] == "more"
assert txt[5:] == "more text" # omitting the second index means "to th
```

Negative indexes count from the end: -1 is the last byte, and so on:

```
txt = "Some more text"
assert txt[-1] == "t"
assert txt[-4:] == "text"
```

- Replace every occurrence of a byte (or a string) in a string wit

Strings are objects and have methods, like replace:

```
v1 = "hello world"
v2 = v1.replace("l", "L")
print v2 # prints heLLo worLd
```

- Join strings

If they're separate variables, use the + operator:

```
v1 = "hello"
v2 = "world"
msg = v1 + " " + v2
```

If the elements to join are contained inside any iterable container (e

```
items = ["Smith", "John", "417 Evergreen Av", "Chimichurri", "481-3172
joined = ",".join(items)
print joined
# output:
# Smith,John,417 Evergreen Av,Chimichurri,481-3172
```

The reverse operation (split) is also possible:

```
line = "Smith,John,417 Evergreen Av,Chimichurri,481-3172"
fields = line.split(',')
print fields
# output:
# ['Smith', 'John', '417 Evergreen Av', 'Chimichurri', '481-3172']
```

3.x[edit]

Python 3.x has two binary string types: bytes (immutable) and bytearra

To specify a literal immutable byte string (bytes), prefix a string li

```
s1 = b"A 'byte string' literal \n"
s2 = b'You may use any of \' or " as delimiter'
s3 = b"""This text
   goes over several lines
       up to the closing triple quote"""
```

You can use the normal string escape sequences to encode special bytes

Indexing a byte string results in an integer (the byte value at that b

```
x = b'abc'
x[0] # evaluates to 97
```

Similarly, a byte string can be converted to and from a list of intege

```
x = b'abc'
list(x) # evaluates to [97, 98, 99]
bytes([97, 98, 99]) # evaluates to b'abc'
```

Python: Binary_strings[edit]

2.x[edit]

Python 2.x's string type (str) is a native byte string type. They can

- String creation

```
s1 = "A 'string' literal \n"
s2 = 'You may use any of \' or " as delimiter'
s3 = """This text
   goes over several lines
```

```
    up to the closing triple quote"""
```

- String assignment

There is nothing special about assignments:

```
s = "Hello "
t = "world!"
u = s + t   # + concatenates
```

- String comparison

They're compared byte by byte, lexicographically:

```
assert "Hello" == 'Hello'
assert '\t' == '\x09'
assert "one" < "two"
assert "two" >= "three"
```

- String cloning and copying

Strings are immutable, so there is no need to clone/copy them. If you

- Check if a string is empty

```
if x=='': print "Empty string"
if not x: print "Empty string, provided you know x is a string"
```

- Append a byte to a string

```
txt = "Some text"
txt += '\x07'
# txt refers now to a new string having "Some text\x07"
```

- Extract a substring from a string

Strings are sequences, they can be indexed with s[index] (index is 0-b

```
txt = "Some more text"
assert txt[4] == " "
```

```
assert txt[0:4] == "Some"
assert txt[:4] == "Some" # you can omit the starting index if 0
assert txt[5:9] == "more"
assert txt[5:] == "more text" # omitting the second index means "to th
```

Negative indexes count from the end: -1 is the last byte, and so on:

```
txt = "Some more text"
assert txt[-1] == "t"
assert txt[-4:] == "text"
```

- Replace every occurrence of a byte (or a string) in a string wit

Strings are objects and have methods, like replace:

```
v1 = "hello world"
v2 = v1.replace("l", "L")
print v2 # prints heLLo worLd
```

- Join strings

If they're separate variables, use the + operator:

```
v1 = "hello"
v2 = "world"
msg = v1 + " " + v2
```

If the elements to join are contained inside any iterable container (e

```
items = ["Smith", "John", "417 Evergreen Av", "Chimichurri", "481-3172
joined = ",".join(items)
print joined
# output:
# Smith,John,417 Evergreen Av,Chimichurri,481-3172
```

The reverse operation (split) is also possible:

```
line = "Smith,John,417 Evergreen Av,Chimichurri,481-3172"
```

```
fields = line.split(',')
print fields
# output:
# ['Smith', 'John', '417 Evergreen Av', 'Chimichurri', '481-3172']
```

3.x[edit]

Python 3.x has two binary string types: bytes (immutable) and bytearra

To specify a literal immutable byte string (bytes), prefix a string li

```
s1 = b"A 'byte string' literal \n"
s2 = b'You may use any of \' or " as delimiter'
s3 = b"""This text
   goes over several lines
       up to the closing triple quote"""
```

You can use the normal string escape sequences to encode special bytes

Indexing a byte string results in an integer (the byte value at that b

```
x = b'abc'
x[0] # evaluates to 97
```

Similarly, a byte string can be converted to and from a list of intege

```
x = b'abc'
list(x) # evaluates to [97, 98, 99]
bytes([97, 98, 99]) # evaluates to b'abc'
```

Python: Birthday_problem[edit]

Note: the first (unused), version of function equal_birthdays() uses a

```
from random import randint
```

```
def equal_birthdays(sharers=2, groupsize=23, rep=100000):
    'Note: 4 sharing common birthday may have 2 dates shared between t
    g = range(groupsize)
    sh = sharers - 1
    eq = sum((groupsize - len(set(randint(1,365) for i in g)) >= sh)
             for j in range(rep))
    return (eq * 100.) / rep

def equal_birthdays(sharers=2, groupsize=23, rep=100000):
    'Note: 4 sharing common birthday must all share same common day'
    g = range(groupsize)
    sh = sharers - 1
    eq = 0
    for j in range(rep):
        group = [randint(1,365) for i in g]
        if (groupsize - len(set(group)) >= sh and
            any( group.count(member) >= sharers for member in set(grou
            eq += 1
    return (eq * 100.) / rep

group_est = [2]
for sharers in (2, 3, 4, 5):
    groupsize = group_est[-1]+1
    while equal_birthdays(sharers, groupsize, 100) < 50.:
        # Coarse
        groupsize += 1
    for groupsize in range(int(groupsize - (groupsize - group_est[-1])
        # Finer
        eq = equal_birthdays(sharers, groupsize, 250)
        if eq > 50.:
            break
    for groupsize in range(groupsize - 1, groupsize +999):
        # Finest
        eq = equal_birthdays(sharers, groupsize, 50000)
        if eq > 50.:
            break
    group_est.append(groupsize)
    print("%i independent people in a group of %s share a common birth
```

Output:

```
2 independent people in a group of 23 share a common birthday. ( 50.9)
3 independent people in a group of 87 share a common birthday. ( 50.0)
4 independent people in a group of 188 share a common birthday. ( 50.9
5 independent people in a group of 314 share a common birthday. ( 50.6
```

Enumeration method[edit]

The following enumerates all birthday distributation of n people in a

```
from collections import defaultdict
days = 365

def find_half(c):

    # inc_people takes birthday combinations of n people and generates
    # new set for n+1
    def inc_people(din, over):
        # 'over' is the number of combinations that have at least c pe
        # sharing a birthday. These are not contained in the set.

        dout,over = defaultdict(int), over * days
        for k,s in din.items():
            for i,v in enumerate(k):
                if v + 1 >= c:
                    over += s
                else:
                    dout[tuple(sorted(k[0:i] + (v + 1,) + k[i+1:]))] +
            dout[(1,) + k] += s * (days - len(k))
        return dout, over

    d, combos, good, n = {():1}, 1, 0, 0

    # increase number of people until at least half of the cases have
    # at least c people sharing a birthday
    while True:
        n += 1
        combos *= days # or, combos = sum(d.values()) + good
        d,good = inc_people(d, good)

        #!!! print d.items()
        if good * 2 >= combos:
            return n, good, combos

# In all fairness, I don't know if the code works for x >= 4: I probab
# have enough RAM for it, and certainly not enough patience. But it sh
# In theory.
for x in range(2, 5):
    n, good, combos = find_half(x)
    print "%d of %d people sharing birthday: %d out of %d combos"% (x,
```

Output:

```
2 of 23 people sharing birthday: 43450860051057961364418604769486195
3 of 88 people sharing birthday: 15497024004014734259832774247376969
...?
```

Enumeration method #2[edit]

```
# ought to use a memoize class for all this
# factorial
def fact(n, cache={0:1}):
    if not n in cache:
        cache[n] = n * fact(n - 1)
    return cache[n]

# permutations
def perm(n, k, cache={}):
    if not (n,k) in cache:
        cache[(n,k)] = fact(n) / fact(n - k)
    return cache[(n,k)]

def choose(n, k, cache={}):
    if not (n,k) in cache:
        cache[(n,k)] = perm(n, k) / fact(k)
    return cache[(n, k)]

# ways of distribute p people's birthdays into d days, with
# no more than m sharing any one day
def combos(d, p, m, cache={}):
    if not p: return 1
    if not m: return 0
    if p <= m: return d**p        # any combo would satisfy

    k = (d, p, m)
    if not k in cache:
        result = 0
        for x in range(0, p//m + 1):
            c = combos(d - x, p - x * m, m - 1)
            # ways to occupy x days with m people each
            if c: result += c * choose(d, x) * perm(p, x * m) / fact(m
        cache[k] = result

    return cache[k]

def find_half(m):
    n = 0
    while True:
        n += 1
        total = 365 ** n
        c = total - combos(365, n, m - 1)
        if c * 2 >= total:
            print "%d of %d people: %d/%d combos" % (n, m, c, total)
            return

for x in range(2, 6): find_half(x)
```

Output:

```
23 of 2 people: 43450860....3125/85651679....3125 combos
88 of 3 people: 15497...50390625/30322...50390625 combos
187 of 4 people: 708046698...0703125/1408528546...0703125 combos
313 of 5 people: 498385488882289...2578125/99464149835930...2578125 co
```

Racket[edit]

Python: Bitwise_operations[edit]

```
def bitwise(a, b):
        print 'a and b:', a & b
        print 'a or b:', a | b
        print 'a xor b:', a ^ b
        print 'not a:', ~a
        print 'a << b:', a << b # left shift
        print 'a >> b:', a >> b # arithmetic right shift
```

Python does not have built in rotate or logical right shift operations

Note: Newer Python versions (circa 2.4?) will automatically promote in

```
# 8-bit bounded shift:
x = x << n & 0xff
# ditto for 16 bit:
x = x << n & 0xffff
# ... and 32-bit:
x = x << n & 0xffffffff
# ... and 64-bit:
x = x << n & 0xffffffffffffffff
```

We can easily implement our own rotation functions. For left rotations

```
def bitstr(n, width=None):
   """return the binary representation of n as a string and
      optionally zero-fill (pad) it to a given length
   """
   result = list()
   while n:
```

```
        result.append(str(n%2))
        n = int(n/2)
    if (width is not None) and len(result) < width:
        result.extend(['0'] * (width - len(result)))
    result.reverse()
    return ''.join(result)

def mask(n):
    """Return a bitmask of length n (suitable for masking against an
        int to coerce the size to a given length)
    """
    if n >= 0:
        return 2**n - 1
    else:
        return 0

def rol(n, rotations=1, width=8):
    """Return a given number of bitwise left rotations of an integer n
        for a given bit field width.
    """
    rotations %= width
    if rotations < 1:
        return n
    n &= mask(width) ## Should it be an error to truncate here?
    return ((n << rotations) & mask(width)) | (n >> (width - rotations

def ror(n, rotations=1, width=8):
    """Return a given number of bitwise right rotations of an integer
        for a given bit field width.
    """
    rotations %= width
    if rotations < 1:
        return n
    n &= mask(width)
    return (n >> rotations) | ((n << (width - rotations)) & mask(width
```

In this example we show a relatively straightforward function for conv

R[edit]

Python: Bogosort[edit]

```
import random

def bogosort(l):
    while not in_order(l):
        random.shuffle(l)
```

```
    return l

def in_order(l):
    if not l:
        return True
    last = l[0]
    for x in l[1:]:
        if x < last:
            return False
        last = x
    return True
```

Alternative definition for *in_order* (Python 2.5)

```
def in_order(l):
    return all( l[i] <= l[i+1] for i in xrange(0,len(l)-1))
```

An alternative implementation for Python 2.5 or later:

```
import random
def bogosort(lst):
   random.shuffle(lst)  # must shuffle it first or it's a bug if lst w
   while lst != sorted(lst):
       random.shuffle(lst)
   return lst
```

Another alternative implementation, using iterators for maximum effici

```
import operator
import random
from itertools import dropwhile, imap, islice, izip, repeat, starmap

def shuffled(x):
    x = x[:]
    random.shuffle(x)
    return x

bogosort = lambda l: next(dropwhile(
    lambda l: not all(starmap(operator.le, izip(l, islice(l, 1, None))
    imap(shuffled, repeat(l))))
```

Python: Boolean_values[edit]

Python has a boolean data type with the only two possible values denot

The boolean type is a member of the numeric family of types, and when

In a boolean context, Python extends what is meant by true and false b

A user-created class that defines a .__nonzero__() method to return Fa

None is also False in a boolean context.

Some examples:

```
>>> True
True
>>> not True
False
>>> # As numbers
>>> False + 0
0
>>> True + 0
1
>>> False + 0j
0j
>>> True * 3.141
3.141
>>> # Numbers as booleans
>>> not 0
True
>>> not not 0
False
>>> not 1234
False
>>> bool(0.0)
False
>>> bool(0j)
False
>>> bool(1+2j)
True
>>> # Collections as booleans
>>> bool([])
False
>>> bool([None])
True
>>> 'I contain something' if (None,) else 'I am empty'
'I contain something'
>>> bool({})
False
>>> bool("")
False
>>> bool("False")
True
```

R[edit]

Python: Box_the_compass[edit]

```
majors   = 'north east south west'.split()
majors   *= 2 # no need for modulo later
quarter1 = 'N,N by E,N-NE,NE by N,NE,NE by E,E-NE,E by N'.split(',')
quarter2 = [p.replace('NE','EN') for p in quarter1]

def degrees2compasspoint(d):
    d = (d % 360) + 360/64
    majorindex, minor = divmod(d, 90.)
    majorindex = int(majorindex)
    minorindex  = int( (minor*4) // 45 )
    p1, p2 = majors[majorindex: majorindex+2]
    if p1 in {'north', 'south'}:
        q = quarter1
    else:
        q = quarter2
    return q[minorindex].replace('N', p1).replace('E', p2).capitalize(

if __name__ == '__main__':
    for i in range(33):
        d = i * 11.25
        m = i % 3
        if   m == 1: d += 5.62
        elif m == 2: d -= 5.62
        n = i % 32 + 1
        print( '%2i %-18s %7.2f°' % (n, degrees2compasspoint(d), d) )
```

Output

```
 1 North                 0.00°
 2 North by east        16.87°
 3 North-northeast      16.88°
 4 Northeast by north   33.75°
 5 Northeast            50.62°
 6 Northeast by east    50.63°
 7 East-northeast       67.50°
 8 East by north        84.37°
 9 East                 84.38°
10 East by south       101.25°
11 East-southeast      118.12°
12 Southeast by east   118.13°
13 Southeast           135.00°
14 Southeast by south  151.87°
15 South-southeast     151.88°
```

```
16 South by east        168.75°
17 South                185.62°
18 South by west        185.63°
19 South-southwest      202.50°
20 Southwest by south   219.37°
21 Southwest            219.38°
22 Southwest by west    236.25°
23 West-southwest       253.12°
24 West by south        253.13°
25 West                 270.00°
26 West by north        286.87°
27 West-northwest       286.88°
28 Northwest by west    303.75°
29 Northwest            320.62°
30 Northwest by north   320.63°
31 North-northwest      337.50°
32 North by west        354.37°
 1 North                354.38°
```

Python: Boxing_the_compass[edit]

```
majors   = 'north east south west'.split()
majors   *= 2 # no need for modulo later
quarter1 = 'N,N by E,N-NE,NE by N,NE,NE by E,E-NE,E by N'.split(',')
quarter2 = [p.replace('NE','EN') for p in quarter1]

def degrees2compasspoint(d):
    d = (d % 360) + 360/64
    majorindex, minor = divmod(d, 90.)
    majorindex = int(majorindex)
    minorindex  = int( (minor*4) // 45 )
    p1, p2 = majors[majorindex: majorindex+2]
    if p1 in {'north', 'south'}:
        q = quarter1
    else:
        q = quarter2
    return q[minorindex].replace('N', p1).replace('E', p2).capitalize(

if __name__ == '__main__':
    for i in range(33):
        d = i * 11.25
        m = i % 3
        if   m == 1: d += 5.62
        elif m == 2: d -= 5.62
        n = i % 32 + 1
        print( '%2i %-18s %7.2f°' % (n, degrees2compasspoint(d), d) )
```

Output

```
 1 North                  0.00°
 2 North by east         16.87°
 3 North-northeast       16.88°
 4 Northeast by north    33.75°
 5 Northeast             50.62°
 6 Northeast by east     50.63°
 7 East-northeast        67.50°
 8 East by north         84.37°
 9 East                  84.38°
10 East by south        101.25°
11 East-southeast       118.12°
12 Southeast by east    118.13°
13 Southeast            135.00°
14 Southeast by south   151.87°
15 South-southeast      151.88°
16 South by east        168.75°
17 South                185.62°
18 South by west        185.63°
19 South-southwest      202.50°
20 Southwest by south   219.37°
21 Southwest            219.38°
22 Southwest by west    236.25°
23 West-southwest       253.12°
24 West by south        253.13°
25 West                 270.00°
26 West by north        286.87°
27 West-northwest       286.88°
28 Northwest by west    303.75°
29 Northwest            320.62°
30 Northwest by north   320.63°
31 North-northwest      337.50°
32 North by west        354.37°
 1 North                354.38°
```

Python: Brace_expansion[edit]

```
def getitem(s, depth=0):
    out = [""]
    while s:
        c = s[0]
        if depth and (c == ',' or c == '}'):
            return out,s
        if c == '{':
            x = getgroup(s[1:], depth+1)
            if x:
                out,s = [a+b for a in out for b in x[0]], x[1]
                continue
        if c == '\\' and len(s) > 1:
            s, c = s[1:], c + s[1]

        out, s = [a+c for a in out], s[1:]
```

```
    return out,s

def getgroup(s, depth):
    out, comma = [], False
    while s:
        g,s = getitem(s, depth)
        if not s: break
        out += g

        if s[0] == '}':
            if comma: return out, s[1:]
            return ['{' + a + '}' for a in out], s[1:]

        if s[0] == ',':
            comma,s = True, s[1:]

    return None

# stolen cowbells from perl6 example
for s in '''~/{Downloads,Pictures}/*.{jpg,gif,png}
It{{em,alic}iz,erat}e{d,}, please.
{,{,gotta have{ ,\, again\, }}more }cowbell!
{}} some }{,{\\\\{ edge, edge} \,}{ cases, {here} \\\\\\\\\\}'''.split(
    print "\n\t".join([s] + getitem(s)[0]) + "\n"
```

Output:

```
~/{Downloads,Pictures}/*.{jpg,gif,png}
        ~/Downloads/*.jpg
        ~/Downloads/*.gif
        ~/Downloads/*.png
        ~/Pictures/*.jpg
        ~/Pictures/*.gif
        ~/Pictures/*.png

It{{em,alic}iz,erat}e{d,}, please.
        Itemized, please.
        Itemize, please.
        Italicized, please.
        Italicize, please.
        Iterated, please.
        Iterate, please.

{,{,gotta have{ ,\, again\, }}more }cowbell!
        cowbell!
        more cowbell!
        gotta have more cowbell!
        gotta have\, again\, more cowbell!

{}} some }{,{\\{ edge, edge} \,}{ cases, {here} \\\\\}
```

```
        {}} some }{,{\\ edge \,}{ cases, {here} \\\\\}
        {}} some }{,{\\ edge \,}{ cases, {here} \\\\\}
```

Racket[edit]

Python: Break_OO_privacy[edit]

Python isn't heavily into private class names. Although private class

```
>>> class MyClassName:
        __private = 123
        non_private = __private * 2


>>> mine = MyClassName()
>>> mine.non_private
246
>>> mine.__private
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    mine.__private
AttributeError: 'MyClassName' object has no attribute '__private'
>>> mine._MyClassName__private
123
>>>
```

Python: Bresenham's_line_algorithm[edit]

Works with: Python version 3.1

Extending the example given here and using the algorithm from the Ada

Python: Bubble_Sort[edit]

```
def bubble_sort(seq):
    """Inefficiently sort the mutable sequence (list) in place.
       seq MUST BE A MUTABLE SEQUENCE.
```

```
        As with list.sort() and random.shuffle this does NOT return
    """
    changed = True
    while changed:
        changed = False
        for i in xrange(len(seq) - 1):
            if seq[i] > seq[i+1]:
                seq[i], seq[i+1] = seq[i+1], seq[i]
                changed = True
    return None

if __name__ == "__main__":
   """Sample usage and simple test suite"""

   from random import shuffle

   testset = range(100)
   testcase = testset[:] # make a copy
   shuffle(testcase)
   assert testcase != testset  # we've shuffled it
   bubble_sort(testcase)
   assert testcase == testset  # we've unshuffled it back into a copy
```

Python: Bulls_and_Cows[edit]

```
'''
 Bulls and cows. A game pre-dating, and similar to, Mastermind.
'''

import random

digits = '123456789'
size = 4
chosen = ''.join(random.sample(digits,size))
#print chosen # Debug
print '''I have chosen a number from %s unique digits from 1 to 9 arra
You need to input a %i digit, unique digit number as a guess at what I
guesses = 0
while True:
    guesses += 1
    while True:
        # get a good guess
        guess = raw_input('\nNext guess [%i]: ' % guesses).strip()
        if len(guess) == size and \
           all(char in digits for char in guess) \
```

```
           and len(set(guess)) == size:
            break
        print "Problem, try again. You need to enter %i unique digits
    if guess == chosen:
        print '\nCongratulations you guessed correctly in',guesses,'at
        break
    bulls = cows = 0
    for i in range(size):
        if guess[i] == chosen[i]:
            bulls += 1
        elif guess[i] in chosen:
            cows += 1
    print '  %i Bulls\n  %i Cows' % (bulls, cows)
```

Sample output:

Python: Bulls_and_Cows_game[edit]

```
'''
 Bulls and cows. A game pre-dating, and similar to, Mastermind.
'''

import random

digits = '123456789'
size = 4
chosen = ''.join(random.sample(digits,size))
#print chosen # Debug
print '''I have chosen a number from %s unique digits from 1 to 9 arra
You need to input a %i digit, unique digit number as a guess at what I
guesses = 0
while True:
    guesses += 1
    while True:
        # get a good guess
        guess = raw_input('\nNext guess [%i]: ' % guesses).strip()
        if len(guess) == size and \
           all(char in digits for char in guess) \
           and len(set(guess)) == size:
            break
        print "Problem, try again. You need to enter %i unique digits
    if guess == chosen:
        print '\nCongratulations you guessed correctly in',guesses,'at
        break
    bulls = cows = 0
    for i in range(size):
        if guess[i] == chosen[i]:
            bulls += 1
        elif guess[i] in chosen:
```

```
            cows += 1
    print '  %i Bulls\n  %i Cows' % (bulls, cows)
```

Sample output:

```
I have chosen a number from 4 unique digits from 1 to 9 arranged in a
You need to input a 4 digit, unique digit number as a guess at what I

Next guess [1]: 79
Problem, try again. You need to enter 4 unique digits from 1 to 9

Next guess [1]: 7983
  2 Bulls
  2 Cows

Next guess [2]: 7938

Congratulations you guessed correctly in 2 attempts
```

Python: CRC-32[edit]

Library[edit]

zlib.crc32 and binascii.crc32 give identical results.

```
>>> import zlib
>>> hex(zlib.crc32('The quick brown fox jumps over the lazy dog'))
'0x414fa339'
```

```
>>> import binascii
>>> hex(binascii.crc32('The quick brown fox jumps over the lazy dog'))
'0x414fa339'
```

If you have Python 2.x, these functions might return a negative intege

Python: CSV_data_manipulation[edit]

Note that the csv module is not required for such a simple and regular

```
import fileinput

changerow, changecolumn, changevalue = 2, 4, '"Spam"'

with fileinput.input('csv_data_manipulation.csv', inplace=True) as f:
    for line in f:
        if fileinput.filelineno() == changerow:
            fields = line.rstrip().split(',')
            fields[changecolumn-1] = changevalue
            line = ','.join(fields) + '\n'
        print(line, end='')
```

Output:

After this the data file csv_data_manipulation.csv gets changed from t

```
C1,C2,C3,C4,C5
1,5,9,"Spam",17
2,6,10,14,18
3,7,11,15,19
4,8,12,16,20
```

Python: CSV_to_HTML_translation[edit]

(Note: rendered versions of both outputs are shown at the foot of this

Simple solution[edit]

```
csvtxt = '''\
Character,Speech
The multitude,The messiah! Show us the messiah!
Brians mother,<angry>Now you listen here! He's not the messiah; he's a
The multitude,Who are you?
Brians mother,I'm his mother; that's who!
The multitude,Behold his mother! Behold his mother!\
'''
```

```
from cgi import escape

def _row2tr(row, attr=None):
    cols = escape(row).split(',')
    return ('<TR>'
            + ''.join('<TD>%s</TD>' % data for data in cols)
            + '</TR>')

def csv2html(txt):
    htmltxt = '<TABLE summary="csv2html program output">\n'
    for rownum, row in enumerate(txt.split('\n')):
        htmlrow = _row2tr(row)
        htmlrow = '  <TBODY>%s</TBODY>\n' % htmlrow
        htmltxt += htmlrow
    htmltxt += '</TABLE>\n'
    return htmltxt

htmltxt = csv2html(csvtxt)
print(htmltxt)
```

Sample HTML output

```
<TABLE summary="csv2html program output">
  <TBODY><TR><TD>Character</TD><TD>Speech</TD></TR></TBODY>
  <TBODY><TR><TD>The multitude</TD><TD>The messiah! Show us the messia
  <TBODY><TR><TD>Brians mother</TD><TD>&lt;angry&gt;Now you listen her
  <TBODY><TR><TD>The multitude</TD><TD>Who are you?</TD></TR></TBODY>
  <TBODY><TR><TD>Brians mother</TD><TD>I'm his mother; that's who!</TD
  <TBODY><TR><TD>The multitude</TD><TD>Behold his mother! Behold his m
</TABLE>
```

Extra credit solution[edit]

```
def _row2trextra(row, attr=None):
    cols = escape(row).split(',')
    attr_tr = attr.get('TR', '')
    attr_td = attr.get('TD', '')
    return (('<TR%s>' % attr_tr)
            + ''.join('<TD%s>%s</TD>' % (attr_td, data) for data in co
            + '</TR>')

def csv2htmlextra(txt, header=True, attr=None):
    ' attr is a dictionary mapping tags to attributes to add to that t

    attr_table = attr.get('TABLE', '')
    attr_thead = attr.get('THEAD', '')
    attr_tbody = attr.get('TBODY', '')
    htmltxt = '<TABLE%s>\n' % attr_table
```

```
    for rownum, row in enumerate(txt.split('\n')):
        htmlrow = _row2trextra(row, attr)
        rowclass = ('THEAD%s' % attr_thead) if (header and rownum == 0
        htmlrow = '  <%s>%s</%s>\n' % (rowclass, htmlrow, rowclass[:5]
        htmltxt += htmlrow
    htmltxt += '</TABLE>\n'
    return htmltxt

htmltxt = csv2htmlextra(csvtxt, True,
                        dict(TABLE=' border="1" summary="csv2html extr
                             THEAD=' bgcolor="yellow"',
                             TBODY=' bgcolor="orange"'
                             )
                        )
print(htmltxt)
```

Sample HTML output

```
<TABLE border="1" summary="csv2html extra program output">
  <THEAD bgcolor="yellow"><TR><TD>Character</TD><TD>Speech</TD></TR></
  <TBODY bgcolor="orange"><TR><TD>The multitude</TD><TD>The messiah! S
  <TBODY bgcolor="orange"><TR><TD>Brians mother</TD><TD>&lt;angry&gt;N
  <TBODY bgcolor="orange"><TR><TD>The multitude</TD><TD>Who are you?</
  <TBODY bgcolor="orange"><TR><TD>Brians mother</TD><TD>I'm his mother
  <TBODY bgcolor="orange"><TR><TD>The multitude</TD><TD>Behold his mot
</TABLE>
```

Python: C_FFI[edit]

```
import ctypes
libc = ctypes.CDLL("/lib/libc.so.6")
libc.strcmp("abc", "def")     # -1
libc.strcmp("hello", "hello") #  0
```

Python: Caesar_cipher[edit]

```
def caesar(s, k, decode = False):
        if decode: k = 26 - k
```

```
        return "".join([chr((ord(i) - 65 + k) % 26 + 65)
                                for i in s.upper()
                                if ord(i) >= 65 and ord(i) <= 90 ])

msg = "The quick brown fox jumped over the lazy dogs"
print msg
enc = caesar(msg, 11)
print enc
print caesar(enc, 11, decode = True)
```

Output:

Python: Calendar[edit]

The Python calendar.pryear function prints calendars with the followin

Although rumoured to be getting an anti-gravity module, Python as yet

```
>>> import calendar
>>> help(calendar.prcal)
Help on method pryear in module calendar:

pryear(self, theyear, w=0, l=0, c=6, m=3) method of calendar.TextCalen
    Print a years calendar.

>>> calendar.prcal(1969)
                                  1969

      January                   February                    March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa
       1  2  3  4  5                      1  2                       1
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       3  4  5  6  7  8
13 14 15 16 17 18 19      10 11 12 13 14 15 16      10 11 12 13 14 15
20 21 22 23 24 25 26      17 18 19 20 21 22 23      17 18 19 20 21 22
27 28 29 30 31            24 25 26 27 28            24 25 26 27 28 29
                                                    31

       April                      May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa
    1  2  3  4  5  6                1  2  3  4
 7  8  9 10 11 12 13       5  6  7  8  9 10 11       2  3  4  5  6  7
14 15 16 17 18 19 20      12 13 14 15 16 17 18       9 10 11 12 13 14
21 22 23 24 25 26 27      19 20 21 22 23 24 25      16 17 18 19 20 21
28 29 30                  26 27 28 29 30 31         23 24 25 26 27 28
                                                    30

        July                     August                   September
```

```
Mo Tu We Th Fr Sa Su       Mo Tu We Th Fr Sa Su       Mo Tu We Th Fr Sa
    1  2  3  4  5  6                      1  2  3        1  2  3  4  5  6
 7  8  9 10 11 12 13        4  5  6  7  8  9 10        8  9 10 11 12 13
14 15 16 17 18 19 20       11 12 13 14 15 16 17       15 16 17 18 19 20
21 22 23 24 25 26 27       18 19 20 21 22 23 24       22 23 24 25 26 27
28 29 30 31                25 26 27 28 29 30 31       29 30

      October                    November                   December
Mo Tu We Th Fr Sa Su       Mo Tu We Th Fr Sa Su       Mo Tu We Th Fr Sa
       1  2  3  4  5                         1  2        1  2  3  4  5  6
 6  7  8  9 10 11 12        3  4  5  6  7  8  9        8  9 10 11 12 13
13 14 15 16 17 18 19       10 11 12 13 14 15 16       15 16 17 18 19 20
20 21 22 23 24 25 26       17 18 19 20 21 22 23       22 23 24 25 26 27
27 28 29 30 31             24 25 26 27 28 29 30       29 30 31

>>> print('1234567890'*8)
12345678901234567890123456789012345678901234567890123456789012345678901234567890
>>>
```

Python: Call_a_foreign-language_function[edit]

```
import ctypes
libc = ctypes.CDLL("/lib/libc.so.6")
libc.strcmp("abc", "def")     # -1
libc.strcmp("hello", "hello") #  0
```

Python: Call_a_function[edit]

Under the hood all Python function/method parameters are named. All ar

```
def no_args():
    pass
# call
no_args()

def fixed_args(x, y):
    print('x=%r, y=%r' % (x, y))
# call
fixed_args(1, 2)        # x=1, y=2
```

```
## Can also called them using the parameter names, in either order:
fixed_args(y=2, x=1)

## Can also "apply" fixed_args() to a sequence:
myargs=(1,2) # tuple
fixed_args(*myargs)

def opt_args(x=1):
    print(x)
# calls
opt_args()                 # 1
opt_args(3.141)            # 3.141

def var_args(*v):
    print(v)
# calls
var_args(1, 2, 3)          # (1, 2, 3)
var_args(1, (2,3))         # (1, (2, 3))
var_args()                 # ()

## Named arguments
fixed_args(y=2, x=1)       # x=1, y=2

## As a statement
if 1:
    no_args()

## First-class within an expression
assert no_args() is None

def return_something():
    return 1
x = return_something()

def is_builtin(x):
        print(x.__name__ in dir(__builtins__))
# calls
is_builtin(pow)            # True
is_builtin(is_builtin)     # False

# Very liberal function definition

def takes_anything(*args, **kwargs):
    for each in args:
        print(each)
    for key, value in sorted(kwargs.items()):
        print("%s:%s" % (key, value))
    # Passing those to another, wrapped, function:
    wrapped_fn(*args, **kwargs)
    # (Function being wrapped can have any parameter list
    # ... that doesn't have to match this prototype)

## A subroutine is merely a function that has no explicit
## return statement and will return None.

## Python uses "Call by Object Reference".
```

```
## See, for example, http://www.python-course.eu/passing_arguments.php

## For partial function application see:
##   http://rosettacode.org/wiki/Partial_function_application#Python
```

Python: Call_an_object_method[edit]

```
class MyClass(object):
        @classmethod
        def myClassMethod(self, x):
                pass
        @staticmethod
        def myStaticMethod(x):
                pass
        def myMethod(self, x):
                return 42 + x

myInstance = MyClass()

# Instance method
myInstance.myMethod(someParameter)
# A method can also be retrieved as an attribute from the class, and t
MyClass.myMethod(myInstance, someParameter)


# Class or static methods
MyClass.myClassMethod(someParameter)
MyClass.myStaticMethod(someParameter)
# You can also call class or static methods on an instance, which will
myInstance.myClassMethod(someParameter)
myInstance.myStaticMethod(someParameter)
```

Racket[edit]

Python: Call_foreign_language_function[edit]

```
import ctypes
libc = ctypes.CDLL("/lib/libc.so.6")
libc.strcmp("abc", "def")     # -1
libc.strcmp("hello", "hello") #  0
```

Python: Call_function_in_shared_library[edit]

Example that call User32.dll::GetDoubleClickTime() in windows.

```
import ctypes

user32_dll = ctypes.cdll.LoadLibrary('User32.dll')
print user32_dll.GetDoubleClickTime()
```

Python: Calling_a_method[edit]

```
class MyClass(object):
        @classmethod
        def myClassMethod(self, x):
                pass
        @staticmethod
        def myStaticMethod(x):
                pass
        def myMethod(self, x):
                return 42 + x

myInstance = MyClass()

# Instance method
myInstance.myMethod(someParameter)
# A method can also be retrieved as an attribute from the class, and t
MyClass.myMethod(myInstance, someParameter)


# Class or static methods
MyClass.myClassMethod(someParameter)
MyClass.myStaticMethod(someParameter)
# You can also call class or static methods on an instance, which will
myInstance.myClassMethod(someParameter)
myInstance.myStaticMethod(someParameter)
```

Racket[edit]

Python: Carmichael_3_strong_pseudoprimes[edit]

```
class Isprime():
    '''
    Extensible sieve of Eratosthenes

    >>> isprime.check(11)
    True
    >>> isprime.multiples
    {2, 4, 6, 8, 9, 10}
    >>> isprime.primes
    [2, 3, 5, 7, 11]
    >>> isprime(13)
    True
    >>> isprime.multiples
    {2, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 18, 20, 21, 22}
    >>> isprime.primes
    [2, 3, 5, 7, 11, 13, 17, 19]
    >>> isprime.nmax
    22
    >>>
    '''
    multiples = {2}
    primes = [2]
    nmax = 2

    def __init__(self, nmax):
        if nmax > self.nmax:
            self.check(nmax)

    def check(self, n):
        if type(n) == float:
            if not n.is_integer(): return False
            n = int(n)
        multiples = self.multiples
        if n <= self.nmax:
            return n not in multiples
        else:
            # Extend the sieve
            primes, nmax = self.primes, self.nmax
            newmax = max(nmax*2, n)
            for p in primes:
                multiples.update(range(p*((nmax + p + 1) // p), newmax
            for i in range(nmax+1, newmax+1):
                if i not in multiples:
                    primes.append(i)
                    multiples.update(range(i*2, newmax+1, i))
            self.nmax = newmax
            return n not in multiples

    __call__ = check


def carmichael(p1):
    ans = []
    if isprime(p1):
```

```
        for h3 in range(2, p1):
            g = h3 + p1
            for d in range(1, g):
                if (g * (p1 - 1)) % d == 0 and (-p1 * p1) % h3 == d %
                    p2 = 1 + ((p1 - 1)* g // d)
                    if isprime(p2):
                        p3 = 1 + (p1 * p2 // h3)
                        if isprime(p3):
                            if (p2 * p3) % (p1 - 1) == 1:
                                #print('%i X %i X %i' % (p1, p2, p3))
                                ans += [tuple(sorted((p1, p2, p3)))]
    return ans

isprime = Isprime(2)

ans = sorted(sum((carmichael(n) for n in range(62) if isprime(n)), [])
print(',\n'.join(repr(ans[i:i+5])[1:-1] for i in range(0, len(ans)+1,
```

Output:

```
(3, 11, 17), (5, 13, 17), (5, 17, 29), (5, 29, 73), (7, 13, 19),
(7, 13, 31), (7, 19, 67), (7, 23, 41), (7, 31, 73), (7, 73, 103),
(13, 37, 61), (13, 37, 97), (13, 37, 241), (13, 61, 397), (13, 97, 421
(17, 41, 233), (17, 353, 1201), (19, 43, 409), (19, 199, 271), (23, 19
(29, 113, 1093), (29, 197, 953), (31, 61, 211), (31, 61, 271), (31, 61
(31, 151, 1171), (31, 181, 331), (31, 271, 601), (31, 991, 15361), (37
(37, 73, 181), (37, 73, 541), (37, 109, 2017), (37, 613, 1621), (41, 6
(41, 73, 137), (41, 101, 461), (41, 241, 521), (41, 241, 761), (41, 88
(41, 1721, 35281), (43, 127, 211), (43, 127, 1093), (43, 127, 2731), (
(43, 211, 757), (43, 271, 5827), (43, 433, 643), (43, 547, 673), (43,
(43, 631, 13567), (43, 3361, 3907), (47, 1151, 1933), (47, 3359, 6073)
(53, 79, 599), (53, 157, 521), (53, 157, 2081), (59, 1451, 2089), (61,
(61, 181, 5521), (61, 241, 421), (61, 271, 571), (61, 277, 2113), (61,
(61, 541, 3001), (61, 661, 2521), (61, 1301, 19841), (61, 3361, 4021)
```

Python: Case-sensitivity_of_identifiers[edit]

Python names are case sensitive:

```
>>> dog = 'Benjamin'; Dog = 'Samba'; DOG = 'Bernie'
>>> print ('The three dogs are named ',dog,', ',Dog,', and ',DOG)
The three dogs are named  Benjamin ,  Samba , and  Bernie
>>>
```

R[edit]

Python: Casting_out_nines[edit]

This works slightly differently, generating the "wierd" (as defined by

```
# Casting out Nines
#
# Nigel Galloway: June 27th., 2012,
#
def CastOut(Base=10, Start=1, End=999999):
  ran = [y for y in range(Base-1) if y%(Base-1) == (y*y)%(Base-1)]
  x,y = divmod(Start, Base-1)
  while True:
    for n in ran:
      k = (Base-1)*x + n
      if k < Start:
        continue
      if k > End:
        return
      yield k
    x += 1

for V in CastOut(Base=16,Start=1,End=255):
  print(V, end=' ')
```

Python: Catamorphism[edit]

```
>>> from operator import add
>>> listoflists = [['the', 'cat'], ['sat', 'on'], ['the', 'mat']]
>>> help(reduce)
Help on built-in function reduce in module __builtin__:

reduce(...)
    reduce(function, sequence[, initial]) -> value

    Apply a function of two arguments cumulatively to the items of a s
    from left to right, so as to reduce the sequence to a single value
    For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates
    ((((1+2)+3)+4)+5).  If initial is present, it is placed before the
    of the sequence in the calculation, and serves as a default when t
    sequence is empty.

>>> reduce(add, listoflists, [])
```

```
['the', 'cat', 'sat', 'on', 'the', 'mat']
>>>
```

Additional example[edit]

```
from functools import reduce
from operator import add, mul

nums = range(1,11)

summation = reduce(add, nums)

product = reduce(mul, nums)

concatenation = reduce(lambda a, b: str(a) + str(b), nums)

print(summation, product, concatenation)
```

Racket[edit]

Python: Change_string_case[edit]

```
s = "alphaBETA"
print s.upper() # => "ALPHABETA"
print s.lower() # => "alphabeta"

print s.swapcase() # => "ALPHAbeta"

print "fOo bAR".capitalize() # => "Foo bar"
print "fOo bAR".title() # => "Foo Bar"

import string
print string.capwords("fOo bAR") # => "Foo Bar"
```

string.capwords() allows the user to define word separators, and by de

```
print "foo's bar".title()          # => "Foo'S Bar"
print string.capwords("foo's bar") # => "Foo's Bar"
```

R[edit]

Python: Character_code[edit]

Works with: Python version 2.x

Here character is just a string of length 1

8-bit characters:

Python: Character_codes[edit]

Works with: Python version 2.x

Here character is just a string of length 1

8-bit characters:

```
print ord('a') # prints "97"
print chr(97) # prints "a"
```

Unicode characters:

```
print ord(u'π') # prints "960"
print unichr(960) # prints "π"
```

Works with: Python version 3.x

Here character is just a string of length 1

```
print(ord('a')) # prints "97"
print(ord('π')) # prints "960"
```

```
print(chr(97)) # prints "a"
print(chr(960)) # prints "π"
```

Python: Chat_server[edit]

```
#!/usr/bin/env python

import socket
import thread
import time

HOST = ""
PORT = 4004

def accept(conn):
    """
    Call the inner func in a thread so as not to block. Wait for a
    name to be entered from the given connection. Once a name is
    entered, set the connection to non-blocking and add the user to
    the users dict.
    """
    def threaded():
        while True:
            conn.send("Please enter your name: ")
            try:
                name = conn.recv(1024).strip()
            except socket.error:
                continue
            if name in users:
                conn.send("Name entered is already in use.\n")
            elif name:
                conn.setblocking(False)
                users[name] = conn
                broadcast(name, "+++ %s arrived +++" % name)
                break
    thread.start_new_thread(threaded, ())

def broadcast(name, message):
    """
    Send a message to all users from the given name.
    """
    print message
    for to_name, conn in users.items():
        if to_name != name:
            try:
                conn.send(message + "\n")
            except socket.error:
                pass

# Set up the server socket.
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.setblocking(False)
server.bind((HOST, PORT))
server.listen(1)
print "Listening on %s" % ("%s:%s" % server.getsockname())

# Main event loop.
users = {}
while True:
    try:
        # Accept new connections.
        while True:
            try:
                conn, addr = server.accept()
            except socket.error:
                break
            accept(conn)
        # Read from connections.
        for name, conn in users.items():
            try:
                message = conn.recv(1024)
            except socket.error:
                continue
            if not message:
                # Empty string is given on disconnect.
                del users[name]
                broadcast(name, "--- %s leaves ---" % name)
            else:
                broadcast(name, "%s> %s" % (name, message.strip()))
        time.sleep(.1)
    except (SystemExit, KeyboardInterrupt):
        break
```

Racket[edit]

Python: Check_input_device_is_a_terminal[edit]

```
from sys import stdin
if stdin.isatty():
    print("Input comes from tty.")
else:
    print("Input doesn't come from tty.")
```

```
$ python istty.pl
Input comes from tty.
$ true | python istty.pl
Input doesn't come from tty.
```

Python: Check_that_file_exists[edit]

Works with: Python version 2.5

The *os.path.exists* method will return True if a path exists False if i

```
import os

os.path.exists("input.txt")
os.path.exists("/input.txt")
os.path.exists("docs")
os.path.exists("/docs")
```

Python: Chinese_remainder_theorem[edit]

```
# Python 2.7
def chinese_remainder(n, a):
    sum = 0
    prod = reduce(lambda a, b: a*b, n)

    for n_i, a_i in zip(n, a):
        p = prod / n_i
        sum += a_i * mul_inv(p, n_i) * p
    return sum % prod


def mul_inv(a, b):
    b0 = b
    x0, x1 = 0, 1
    if b == 1: return 1
    while a > 1:
        q = a / b
        a, b = b, a%b
        x0, x1 = x1 - q * x0, x0
    if x1 < 0: x1 += b0
    return x1

if __name__ == '__main__':
    n = [3, 5, 7]
    a = [2, 3, 2]
    print chinese_remainder(n, a)
```

Output:

23

Python: Circles_of_given_radius_through_two_po

The function raises the ValueError exception for the special cases and uses try - except to catch these and extract the exception detail.

```
from collections import namedtuple
from math import sqrt

Pt = namedtuple('Pt', 'x, y')
Circle = Cir = namedtuple('Circle', 'x, y, r')

def circles_from_p1p2r(p1, p2, r):
    'Following explanation at http://mathforum.org/library/drmath/view
    if r == 0.0:
        raise ValueError('radius of zero')
    (x1, y1), (x2, y2) = p1, p2
    if p1 == p2:
        raise ValueError('coincident points gives infinite number of C
    # delta x, delta y between points
    dx, dy = x2 - x1, y2 - y1
    # dist between points
    q = sqrt(dx**2 + dy**2)
    if q > 2.0*r:
        raise ValueError('separation of points > diameter')
    # halfway point
    x3, y3 = (x1+x2)/2, (y1+y2)/2
    # distance along the mirror line
    d = sqrt(r**2-(q/2)**2)
    # One answer
    c1 = Cir(x = x3 - d*dy/q,
             y = y3 + d*dx/q,
             r = abs(r))
    # The other answer
    c2 = Cir(x = x3 + d*dy/q,
             y = y3 - d*dx/q,
             r = abs(r))
    return c1, c2

if __name__ == '__main__':
    for p1, p2, r in [(Pt(0.1234, 0.9876), Pt(0.8765, 0.2345), 2.0),
                      (Pt(0.0000, 2.0000), Pt(0.0000, 0.0000), 1.0),
                      (Pt(0.1234, 0.9876), Pt(0.1234, 0.9876), 2.0),
                      (Pt(0.1234, 0.9876), Pt(0.8765, 0.2345), 0.5),
                      (Pt(0.1234, 0.9876), Pt(0.1234, 0.9876), 0.0)]:
```

```
        print('Through points:\n  %r,\n  %r\n  and radius %f\nYou can
              % (p1, p2, r))
        try:
            print('  %r\n  %r\n' % circles_from_p1p2r(p1, p2, r))
        except ValueError as v:
            print('  ERROR: %s\n' % (v.args[0],))
```

Output:

```
Through points:
  Pt(x=0.1234, y=0.9876),
  Pt(x=0.8765, y=0.2345)
  and radius 2.000000
You can construct the following circles:
  Circle(x=1.8631118016581893, y=1.974211801658189, r=2.0)
  Circle(x=-0.8632118016581896, y=-0.7521118016581892, r=2.0)

Through points:
  Pt(x=0.0, y=2.0),
  Pt(x=0.0, y=0.0)
  and radius 1.000000
You can construct the following circles:
  Circle(x=0.0, y=1.0, r=1.0)
  Circle(x=0.0, y=1.0, r=1.0)

Through points:
  Pt(x=0.1234, y=0.9876),
  Pt(x=0.1234, y=0.9876)
  and radius 2.000000
You can construct the following circles:
  ERROR: coincident points gives infinite number of Circles

Through points:
  Pt(x=0.1234, y=0.9876),
  Pt(x=0.8765, y=0.2345)
  and radius 0.500000
You can construct the following circles:
  ERROR: separation of points > diameter

Through points:
  Pt(x=0.1234, y=0.9876),
  Pt(x=0.1234, y=0.9876)
  and radius 0.000000
You can construct the following circles:
  ERROR: radius of zero
```

Python: Classes[edit]

```
class MyClass:
    name2 = 2 # Class attribute

    def __init__(self):
        """
        Constructor  (Technically an initializer rather than a true "c
        """
        self.name1 = 0 # Instance attribute

    def someMethod(self):
        """
        Method
        """
        self.name1 = 1
        MyClass.name2 = 3


myclass = MyClass() # class name, invoked as a function is the constru

class MyOtherClass:
    count = 0  # Population of "MyOtherClass" objects
    def __init__(self, name, gender="Male", age=None):
        """
        One initializer required, others are optional (with different
        """
        MyOtherClass.count += 1
        self.name = name
        self.gender = gender
        if age is not None:
            self.age = age
    def __del__(self):
        MyOtherClass.count -= 1

person1 = MyOtherClass("John")
print person1.name, person1.gender  # "John Male"
print person1.age                   # Raises AttributeError exception!
person2 = MyOtherClass("Jane", "Female", 23)
print person2.name, person2.gender, person2.age  # "Jane Female 23"
```

```
Python allows for very flexible argument passing including normal name

New-style classes inherit from "object" or any descendant of the "obje
```

```
class MyClass(object):
    ...
```

These "new-style" classes support some features which were unavailable

R[edit]

Python: Client-Authenticated_HTTPS_Request[edi

```
import httplib

connection = httplib.HTTPSConnection('www.example.com',cert_file='myCe
connection.request('GET','/index.html')
response = connection.getresponse()
data = response.read()
```

Racket[edit]

Python: Closest_pair_problem.1[edit]

```
"""
  Compute nearest pair of points using two algorithms

  First algorithm is 'brute force' comparison of every possible pair.
  Second, 'divide and conquer', is based on:
    www.cs.iupui.edu/~xkzou/teaching/CS580/Divide-and-conquer-closestP
"""

from random import randint, randrange
from operator import itemgetter, attrgetter

infinity = float('inf')

# Note the use of complex numbers to represent 2D points making distan

def bruteForceClosestPair(point):
    numPoints = len(point)
    if numPoints < 2:
        return infinity, (None, None)
    return min( ((abs(point[i] - point[j]), (point[i], point[j]))
                 for i in range(numPoints-1)
                 for j in range(i+1,numPoints)),
                key=itemgetter(0))
```

```
def closestPair(point):
    xP = sorted(point, key= attrgetter('real'))
    yP = sorted(point, key= attrgetter('imag'))
    return _closestPair(xP, yP)

def _closestPair(xP, yP):
    numPoints = len(xP)
    if numPoints <= 3:
        return bruteForceClosestPair(xP)
    Pl = xP[:numPoints/2]
    Pr = xP[numPoints/2:]
    Yl, Yr = [], []
    xDivider = Pl[-1].real
    for p in yP:
        if p.real <= xDivider:
            Yl.append(p)
        else:
            Yr.append(p)
    dl, pairl = _closestPair(Pl, Yl)
    dr, pairr = _closestPair(Pr, Yr)
    dm, pairm = (dl, pairl) if dl < dr else (dr, pairr)
    # Points within dm of xDivider sorted by Y coord
    closeY = [p for p in yP  if abs(p.real - xDivider) < dm]
    numCloseY = len(closeY)
    if numCloseY > 1:
        # There is a proof that you only need compare a max of 7 next
        closestY = min( ((abs(closeY[i] - closeY[j]), (closeY[i], clos
                         for i in range(numCloseY-1)
                         for j in range(i+1,min(i+8, numCloseY))),
                        key=itemgetter(0))
        return (dm, pairm) if dm <= closestY[0] else closestY
    else:
        return dm, pairm

def times():
    ''' Time the different functions
    '''
    import timeit

    functions = [bruteForceClosestPair, closestPair]
    for f in functions:
        print 'Time for', f.__name__, timeit.Timer(
            '%s(pointList)' % f.__name__,
            'from closestpair import %s, pointList' % f.__name__).time



pointList = [randint(0,1000)+1j*randint(0,1000) for i in range(2000)]

if __name__ == '__main__':
    pointList = [(5+9j), (9+3j), (2+0j), (8+4j), (7+4j), (9+10j), (1+9
    print pointList
    print '  bruteForceClosestPair:', bruteForceClosestPair(pointList)
    print '            closestPair:', closestPair(pointList)
    for i in range(10):
```

```
        pointList = [randrange(11)+1j*randrange(11) for i in range(10)
        print '\n', pointList
        print ' bruteForceClosestPair:', bruteForceClosestPair(pointLi
        print '           closestPair:', closestPair(pointList)
    print '\n'
    times()
    times()
    times()
```

Output:

followed by timing comparisons

(Note how the two algorithms agree on the minimum distance, but may re

```
[(5+9j), (9+3j), (2+0j), (8+4j), (7+4j), (9+10j), (1+9j), (8+2j), 1
  bruteForceClosestPair: (1.0, ((8+4j), (7+4j)))
            closestPair: (1.0, ((8+4j), (7+4j)))

[(10+6j), (7+0j), (9+4j), (4+8j), (7+5j), (6+4j), (1+9j), (6+4j), (
 bruteForceClosestPair: (0.0, ((6+4j), (6+4j)))
           closestPair: (0.0, ((6+4j), (6+4j)))

[(4+10j), (8+5j), (10+3j), (9+7j), (2+5j), (6+7j), (6+2j), (9+6j),
 bruteForceClosestPair: (1.0, ((9+7j), (9+6j)))
           closestPair: (1.0, ((9+7j), (9+6j)))
```

Python: Cocktail_Sort[edit]

This implementation takes advantage of the identical processing of the

```
def cocktailSort(A):
    up = range(len(A)-1)
    while True:
        for indices in (up, reversed(up)):
            swapped = False
            for i in indices:
                if A[i] > A[i+1]:
                    A[i], A[i+1] =  A[i+1], A[i]
                    swapped = True
            if not swapped:
                return
```

Usage:

```
test1 = [7, 6, 5, 9, 8, 4, 3, 1, 2, 0]
cocktailSort(test1)
print test1
#>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

test2=list('big fjords vex quick waltz nymph')
cocktailSort(test2)
print ''.join(test2)
#>>>      abcdefghiijklmnopqrstuvwxyz
```

Python: Code_Tag_Fixer[edit]

```
# coding: utf-8

import sys
import re

langs = ['ada', 'cpp-qt', 'pascal', 'lscript', 'z80', 'visualprolog',
'html4strict', 'cil', 'objc', 'asm', 'progress', 'teraterm', 'hq9plus'
'genero', 'tsql', 'email', 'pic16', 'tcl', 'apt_sources', 'io', 'apach
'vhdl', 'avisynth', 'winbatch', 'vbnet', 'ini', 'scilab', 'ocaml-brief
'sas', 'actionscript3', 'qbasic', 'perl', 'bnf', 'cobol', 'powershell'
'php', 'kixtart', 'visualfoxpro', 'mirc', 'make', 'javascript', 'cpp',
'sdlbasic', 'cadlisp', 'php-brief', 'rails', 'verilog', 'xml', 'csharp
'actionscript', 'nsis', 'bash', 'typoscript', 'freebasic', 'dot',
'applescript', 'haskell', 'dos', 'oracle8', 'cfdg', 'glsl', 'lotusscri
'mpasm', 'latex', 'sql', 'klonec', 'ruby', 'ocaml', 'smarty', 'python'
'oracle11', 'caddcl', 'robots', 'groovy', 'smalltalk', 'diff', 'fortra
'cfm', 'lua', 'modula3', 'vb', 'autoit', 'java', 'text', 'scala',
'lotusformulas', 'pixelbender', 'reg', '_div', 'whitespace', 'providex
'asp', 'css', 'lolcode', 'lisp', 'inno', 'mysql', 'plsql', 'matlab',
'oobas', 'vim', 'delphi', 'xorg_conf', 'gml', 'prolog', 'bf', 'per',
'scheme', 'mxml', 'd', 'basic4gl', 'm68k', 'gnuplot', 'idl', 'abap',
'intercal', 'c_mac', 'thinbasic', 'java5', 'xpp', 'boo', 'klonecpp',
'blitzbasic', 'eiffel', 'povray', 'c', 'gettext']

slang = '/lang'
code='code'

text = sys.stdin.read()

for i in langs:
    text = text.replace("<%s>" % i,"<lang %s>" % i)
    text = text.replace("</%s>" % i, "<%s>" % slang)
```

```
text = re.sub("(?s)<%s (.+?)>(.*?)</%s>"%(code,code), r"<lang \1>\2<%s

sys.stdout.write(text)
```

Python: Code_segment_unload[edit]

The del statement can make objects (both code and data), available for

Python: Collatz_conjecture[edit]

```
def hailstone(n):
    seq = [n]
    while n>1:
        n = 3*n + 1 if n & 1 else n//2
        seq.append(n)
    return seq

if __name__ == '__main__':
    h = hailstone(27)
    assert len(h)==112 and h[:4]==[27, 82, 41, 124] and h[-4:]==[8, 4,
    print("Maximum length %i was found for hailstone(%i) for numbers <
          max((len(hailstone(i)), i) for i in range(1,100000)))
```

Python: Collections[edit]

Works with: Python version 2.5

Python supports lists, tuples, dictionaries and now sets as built-in c

```
collection = [0, '1']                 # Lists are mutable (editable) a
x = collection[0]                     # accessing an item (which happe
collection.append(2)                  # adding something to the end of
collection.insert(0, '-1')            # inserting a value into the beg
y = collection[0]                     # now returns a string of "-1"
collection.extend([2,'3'])            # same as [collection.append(i)
collection += [2,'3']                 # same as previous line
collection[2:6]                       # a "slice" (collection of the l
len(collection)                       # get the length of (number of e
collection = (0, 1)                   # Tuples are immutable (not edit
collection[:]                         # ... slices work on these too;
```

```
collection[-4:-1]                     # negative slices count from the
collection[::2]                       # slices can also specify a stri
collection="some string"              # strings are treated as sequenc
x = collection[::-1]                  # slice with negative step retur
collection[::2] == "some string"[::2] # True, literal objects don't ne
collection.__getitem__(slice(0,len(collection),2))  # same as previous
collection = {0: "zero", 1: "one"}    # Dictionaries (Hash)
collection['zero'] = 2                # Dictionary members accessed us
collection = set([0, '1'])            # sets (Hash)
```

Python: Color_of_a_screen_pixel[edit]

Library: PyWin32

```
def get_pixel_colour(i_x, i_y):
        import win32gui
        i_desktop_window_id = win32gui.GetDesktopWindow()
        i_desktop_window_dc = win32gui.GetWindowDC(i_desktop_window_id
        long_colour = win32gui.GetPixel(i_desktop_window_dc, i_x, i_y)
        i_colour = int(long_colour)
        return (i_colour & 0xff), ((i_colour >> 8) & 0xff), ((i_colour

print get_pixel_colour(0, 0)
```

Library: PIL

Works with: Windows
 only

```
def get_pixel_colour(i_x, i_y):
        import PIL.ImageGrab
        return PIL.ImageGrab.grab().load()[i_x, i_y]

print get_pixel_colour(0, 0)
```

Library: PIL

Python: Column_Aligner.1[edit]

```
from StringIO import StringIO

textinfile = '''Given$a$text$file$of$many$lines,$where$fields$within$a
are$delineated$by$a$single$'dollar'$character,$write$a$program
that$aligns$each$column$of$fields$by$ensuring$that$words$in$each$
column$are$separated$by$at$least$one$space.
Further,$allow$for$each$word$in$a$column$to$be$either$left$
```

```
justified,$right$justified,$or$center$justified$within$its$column.'''

j2justifier = dict(L=str.ljust, R=str.rjust, C=str.center)

def aligner(infile, justification = 'L'):
  ''' \
  Justify columns of textual tabular input where the row separator is
  and the field separator is a 'dollar' character.
  justification can be L, R, or C; (Left, Right, or Centered).

  Return the justified output as a string
  '''
  assert justification in j2justifier, "justification can be L, R, or
  justifier = j2justifier[justification]

  fieldsbyrow= [line.strip().split('$') for line in infile]
  # pad to same number of fields per row
  maxfields = max(len(row) for row in fieldsbyrow)
  fieldsbyrow = [fields + ['']*(maxfields - len(fields))
                    for fields in fieldsbyrow]
  # rotate
  fieldsbycolumn = zip(*fieldsbyrow)
  # calculate max fieldwidth per column
  colwidths = [max(len(field) for field in column)
               for column in fieldsbycolumn]
  # pad fields in columns to colwidth with spaces
  fieldsbycolumn = [ [justifier(field, width) for field in column]
                     for width, column in zip(colwidths, fieldsbycolum
  # rotate again
  fieldsbyrow = zip(*fieldsbycolumn)

  return "\n".join( " ".join(row) for row in fieldsbyrow)


for align in 'Left Right Center'.split():
  infile = StringIO(textinfile)
  print "\n# %s Column-aligned output:" % align
  print aligner(infile, align[0])
```

Output:

```
# Left Column-aligned output:
Given      a          text       file   of     many      lines,     wh
are        delineated by         a      single 'dollar'  character, wr
that       aligns     each       column of     fields    by         en
column     are        separated  by     at     least     one        sp
Further,   allow      for        each   word   in        a          co
justified, right      justified, or     center justified within     it
```

Python: Combinations[edit]

Starting from Python 2.6 and 3.0 you have a pre-defined function that

```
>>> from itertools import combinations
>>> list(combinations(range(5),3))
[(0, 1, 2), (0, 1, 3), (0, 1, 4), (0, 2, 3), (0, 2, 4), (0, 3, 4), (1,
```

Earlier versions could use functions like the following:

Translation of: E

```
def comb(m, lst):
    if m == 0: return [[]]
    return [[x] + suffix for i, x in enumerate(lst)
            for suffix in comb(m - 1, lst[i + 1:])]
```

Example:

```
>>> comb(3, range(5))
[[0, 1, 2], [0, 1, 3], [0, 1, 4], [0, 2, 3], [0, 2, 4], [0, 3, 4], [1,
```

Translation of: Haskell

```
def comb(m, s):
    if m == 0: return [[]]
    if s == []: return []
    return [s[:1] + a for a in comb(m-1, s[1:])] + comb(m, s[1:])

print comb(3, range(5))
```

R[edit]

Python: Combinations_and_permutations[edit]

Library: SciPy
[edit]

```
from __future__ import print_function

from scipy.misc import factorial as fact
from scipy.misc import comb

def perm(N, k, exact=0):
    return comb(N, k, exact) * fact(k, exact)

exact=True
print('Sample Perms 1..12')
for N in range(1, 13):
    k = max(N-2, 1)
    print('%iP%i =' % (N, k), perm(N, k, exact), end=', ' if N % 5 els

print('\n\nSample Combs 10..60')
for N in range(10, 61, 10):
    k = N-2
    print('%iC%i =' % (N, k), comb(N, k, exact), end=', ' if N % 50 el

exact=False
print('\n\nSample Perms 5..1500 Using FP approximations')
for N in [5, 15, 150, 1500, 15000]:
    k = N-2
    print('%iP%i =' % (N, k), perm(N, k, exact))

print('\nSample Combs 100..1000 Using FP approximations')
for N in range(100, 1001, 100):
    k = N-2
    print('%iC%i =' % (N, k), comb(N, k, exact))
```

Output:

```
Sample Perms 1..12
1P1 = 1, 2P1 = 2, 3P1 = 3, 4P2 = 12, 5P3 = 60
6P4 = 360, 7P5 = 2520, 8P6 = 20160, 9P7 = 181440, 10P8 = 1814400
11P9 = 19958400, 12P10 = 239500800,

Sample Combs 10..60
10C8 = 45, 20C18 = 190, 30C28 = 435, 40C38 = 780, 50C48 = 1225
60C58 = 1770,
```

Python: Command-line_arguments[edit]

sys.argv is a list containing all command line arguments, including th

```
import sys
program_name = sys.argv[0]
arguments = sys.argv[1:]
count = len(arguments)
```

When running a module by invoking Python, the Python interpreter proce

For powerful option parsing capabilities check out the optparse module

Python: Command_Line_Arguments[edit]

sys.argv is a list containing all command line arguments, including th

```
import sys
program_name = sys.argv[0]
arguments = sys.argv[1:]
count = len(arguments)
```

When running a module by invoking Python, the Python interpreter proce

For powerful option parsing capabilities check out the optparse module

Python: Command_Line_Interpreter[edit]

Start the interpreter by typing python at the command line (or select

```
python
Python 2.6.1 (r261:67517, Dec  4 2008, 16:51:00) [MSC v.1500 32 bit (I
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def f(string1, string2, separator):
        return separator.join([string1, '', string2])

>>> f('Rosetta', 'Code', ':')
'Rosetta::Code'
>>>
```

R[edit]

Python: Comments[edit]

Python uses the "#" symbol to mark it's comments. After placing a "#",

```
# This is a comment
foo = 5 # You can also append comments to statements
```

Certain 'do nothing' expressions resemble comments

```
"""Un-assigned strings in triple-quotes might be used
   as multi-line comments
"""

'''
   "triple quoted strings" can be delimited by either 'single' or "dou
   of other quote marks without any need to \escape\ them using any sp
   lines without special escape characters.
'''
```

Note that strings inserted among program statements in Python are trea

Documentation Strings[edit]

Python makes pervasive use of strings which immediately follow class a

```
#!/usr/bin/env python
# Example of using doc strings
"""My Doc-string example"""

class Foo:
     '''Some documentation for the Foo class'''
     def __init__(self):
        "Foo's initialization method's documentation"
```

```
def bar():
    """documentation for the bar function"""

if __name__ == "__main__":
    print (__doc__)
    print (Foo.__doc__)
    print (Foo.__init__.__doc__)
    print (bar.__doc__)
```

... would print each of the various documentation strings in this exam

Python "docstrings" are used by a number of tools to automatically gen

(As noted above extraneous strings interspersed throughout a Python so

Python: Common_number_base_formatting[edit]

Works with: Python version 2.6

Binary (b), Octal (o), Decimal (d), and Hexadecimal (X and x) are supp

```
>>> for n in range(34):
        print " {0:6b} {1:3o} {2:2d} {3:2X}".format(n, n, n, n)
        #The following would give the same output, and,
        #due to the outer brackets, works with Python 3.0 too
        #print ( " {n:6b} {n:3o} {n:2d} {n:2X}".format(n=n) )


      0   0  0  0
      1   1  1  1
     10   2  2  2
     11   3  3  3
    100   4  4  4
```

Works with: Python version 2.5

Octal (o), Decimal (d), and Hexadecimal (X and x), but not binary are

```
for n in range(34):
        print " %3o %2d %2X" % (n, n, n)
```

For each of these bases there is also a built-in function that will co

```
n = 33
#Python 3.x:
print(bin(n), oct(n), n, hex(n)) # bin() only available in Python 3.x
# output: 0b100001 0o41 33 0x21

#Python 2.x:
#print oct(n), n, hex(n)
# output: 041 33 0x21
```

Python: Common_number_base_parsing[edit]

The int function will interpret strings as numbers expressed to some b

```
>>> text = '100'
>>> for base in range(2,21):
    print ("String '%s' in base %i is  %i in base 10"
           % (text, base, int(text, base)))
```

```
String '100' in base 2 is  4 in base 10
String '100' in base 3 is  9 in base 10
String '100' in base 4 is  16 in base 10
String '100' in base 5 is  25 in base 10
String '100' in base 6 is  36 in base 10
String '100' in base 7 is  49 in base 10
String '100' in base 8 is  64 in base 10
String '100' in base 9 is  81 in base 10
String '100' in base 10 is  100 in base 10
String '100' in base 11 is  121 in base 10
String '100' in base 12 is  144 in base 10
String '100' in base 13 is  169 in base 10
String '100' in base 14 is  196 in base 10
String '100' in base 15 is  225 in base 10
String '100' in base 16 is  256 in base 10
String '100' in base 17 is  289 in base 10
String '100' in base 18 is  324 in base 10
String '100' in base 19 is  361 in base 10
String '100' in base 20 is  400 in base 10
```

In addition, if you give a base of 0, it will try to figure out the ba

Python 3.x and 2.6:

```
>>> int("123459", 0)
123459
>>> int("0xabcf123", 0)
180154659
>>> int("0o7651", 0)
```

Python: Compare_a_list_of_strings[edit]

A useful pattern is that when you need some function of an item in a l
(Especially if an index is not needed elsewhere in the algorithm).

```
all(a == nexta for a, nexta in zip(strings, strings[1:]) # All equal
all(a < nexta for a, nexta in zip(strings, strings[1:]) # Strictly asc
```

Racket[edit]

Python: Compare_sorting_algorithms'_performanc

Works with: Python version 2.5

Examples of sorting routines[edit]

```
def builtinsort(x):
    x.sort()

def partition(seq, pivot):
   low, middle, up = [], [], []
   for x in seq:
       if x < pivot:
           low.append(x)
       elif x == pivot:
           middle.append(x)
       else:
           up.append(x)
   return low, middle, up
import random
def qsortranpart(seq):
   size = len(seq)
   if size < 2: return seq
   low, middle, up = partition(seq, random.choice(seq))
```

```
   return qsortranpart(low) + middle + qsortranpart(up)
```

Sequence generators[edit]

Python: Comparing_two_integers[edit]

```
#!/usr/bin/env python
a = input('Enter value of a: ')
b = input('Enter value of b: ')

if a < b:
    print 'a is less than b'
elif a > b:
    print 'a is greater than b'
elif a == b:
    print 'a is equal to b'
```

(Note: in Python3 *input()* will become *int(input())*)

An alternative implementation could use a Python dictionary to house a

Works with: Python version 2.x only, not 3.x

```
#!/usr/bin/env python
import sys
try:
   a = input('Enter value of a: ')
   b = input('Enter value of b: ')
except (ValueError, EnvironmentError), err:
   print sys.stderr, "Erroneous input:", err
   sys.exit(1)

dispatch = {
    -1: 'is less than',
     0: 'is equal to',
     1: 'is greater than'
     }
 print a, dispatch[cmp(a,b)], b
```

In this case the use of a dispatch table is silly. However, more gener

R[edit]

Python: Complex_conjugate[edit]

```
>>> z1 = 1.5 + 3j
>>> z2 = 1.5 + 1.5j
>>> z1 + z2
(3+4.5j)
>>> z1 - z2
1.5j
>>> z1 * z2
(-2.25+6.75j)
>>> z1 / z2
(1.5+0.5j)
>>> - z1
(-1.5-3j)
>>> z1.conjugate()
(1.5-3j)
>>> abs(z1)
3.3541019662496847
>>> z1 ** z2
(-1.1024829553277784-0.38306415117199333j)
>>> z1.real
1.5
>>> z1.imag
3.0
>>>
```

Python: Complex_numbers[edit]

```
>>> z1 = 1.5 + 3j
>>> z2 = 1.5 + 1.5j
>>> z1 + z2
(3+4.5j)
>>> z1 - z2
1.5j
>>> z1 * z2
(-2.25+6.75j)
>>> z1 / z2
(1.5+0.5j)
>>> - z1
(-1.5-3j)
>>> z1.conjugate()
(1.5-3j)
>>> abs(z1)
3.3541019662496847
```

```
>>> z1 ** z2
(-1.1024829553277784-0.38306415117199333j)
>>> z1.real
1.5
>>> z1.imag
3.0
>>>
```

Python: Compound_data_type[edit]

The simplest way it to use a tuple, or a list if it should be mutable:

Python: Concurrent_computing[edit]

Works with: Python version 3.2

Using the new to Python 3.2 concurrent.futures library and choosing to

```
Python 3.2 (r32:88445, Feb 20 2011, 21:30:00) [MSC v.1500 64 bit (AMD6
Type "help", "copyright", "credits" or "license" for more information.
>>> from concurrent import futures
>>> with futures.ProcessPoolExecutor() as executor:
...   _ = list(executor.map(print, 'Enjoy Rosetta Code'.split()))
...
Enjoy
Rosetta
Code
>>>
```

Works with: Python version 2.5

```
import threading
import random

def echo(text):
    print(text)

threading.Timer(random.random(), echo, ("Enjoy",)).start()
threading.Timer(random.random(), echo, ("Rosetta",)).start()
threading.Timer(random.random(), echo, ("Code",)).start()
```

Or, by using a for loop to start one thread per list entry, where our

```
import threading
import random

def echo(text):
    print(text)

for text in ["Enjoy", "Rosetta", "Code"]:
    threading.Timer(random.random(), echo, (text,)).start()
```

threading.Thread[edit]

```
import random, sys, time
import threading

lock = threading.Lock()

def echo(s):
    time.sleep(1e-2*random.random())
    # use `.write()` with lock due to `print` prints empty lines occas
    with lock:
        sys.stdout.write(s)
        sys.stdout.write('\n')

for line in 'Enjoy Rosetta Code'.split():
    threading.Thread(target=echo, args=(line,)).start()
```

multiprocessing[edit]

Works with: Python version 2.6

```
from __future__ import print_function
from multiprocessing import Pool

def main():
    p = Pool()
    p.map(print, 'Enjoy Rosetta Code'.split())

if __name__=="__main__":
    main()
```

twisted[edit]

```
import random
from twisted.internet    import reactor, task, defer
from twisted.python.util import println

delay = lambda: 1e-4*random.random()
d = defer.DeferredList([task.deferLater(reactor, delay(), println, lin
                        for line in 'Enjoy Rosetta Code'.split()])
d.addBoth(lambda _: reactor.stop())
reactor.run()
```

gevent[edit]

```
from __future__ import print_function
import random
import gevent

delay = lambda: 1e-4*random.random()
gevent.joinall([gevent.spawn_later(delay(), print, line)
               for line in 'Enjoy Rosetta Code'.split()])
```

Python: Conjugate_transpose[edit]

Internally, matrices must be represented as rectangular tuples of tupl

```
def conjugate_transpose(m):
    return tuple(tuple(n.conjugate() for n in row) for row in zip(*m))

def mmul( ma, mb):
    return tuple(tuple(sum( ea*eb for ea,eb in zip(a,b)) for b in zip(

def mi(size):
    'Complex Identity matrix'
    sz = range(size)
    m = [[0 + 0j for i in sz] for j in sz]
    for i in range(size):
        m[i][i] = 1 + 0j
    return tuple(tuple(row) for row in m)

def __allsame(vector):
    first, rest = vector[0], vector[1:]
    return all(i == first for i in rest)

def __allnearsame(vector, eps=1e-14):
    first, rest = vector[0], vector[1:]
    return all(abs(first.real - i.real) < eps and abs(first.imag - i.i
```

```
                for i in rest)

    def isequal(matrices, eps=1e-14):
        'Check any number of matrices for equality within eps'
        x = [len(m) for m in matrices]
        if not __allsame(x): return False
        y = [len(m[0]) for m in matrices]
        if not __allsame(y): return False
        for s in range(x[0]):
            for t in range(y[0]):
                if not __allnearsame([m[s][t] for m in matrices], eps): re
        return True


    def ishermitian(m, ct):
        return isequal([m, ct])

    def isnormal(m, ct):
        return isequal([mmul(m, ct), mmul(ct, m)])

    def isunitary(m, ct):
        mct, ctm = mmul(m, ct), mmul(ct, m)
        mctx, mcty, cmx, ctmy = len(mct), len(mct[0]), len(ctm), len(ctm[0
        ident = mi(mctx)
        return isequal([mct, ctm, ident])

    def printm(comment, m):
        print(comment)
        fields = [['%g%+gj' % (f.real, f.imag) for f in row] for row in m]
        width = max(max(len(f) for f in row) for row in fields)
        lines = (', '.join('%*s' % (width, f) for f in row) for row in fie
        print('\n'.join(lines))

    if __name__ == '__main__':
        for matrix in [
                ((( 3.000+0.000j), (+2.000+1.000j)),
                (( 2.000-1.000j), (+1.000+0.000j))),

                ((( 1.000+0.000j), (+1.000+0.000j), (+0.000+0.000j)),
                (( 0.000+0.000j), (+1.000+0.000j), (+1.000+0.000j)),
                (( 1.000+0.000j), (+0.000+0.000j), (+1.000+0.000j))),

                ((( 2**0.5/2+0.000j), (+2**0.5/2+0.000j), (+0.000+0.000j))
                (( 0.000+2**0.5/2j), (+0.000-2**0.5/2j), (+0.000+0.000j)),
                (( 0.000+0.000j), (+0.000+0.000j), (+0.000+1.000j)))]:
            printm('\nMatrix:', matrix)
            ct = conjugate_transpose(matrix)
            printm('Its conjugate transpose:', ct)
            print('Hermitian? %s.' % ishermitian(matrix, ct))
            print('Normal?    %s.' % isnormal(matrix, ct))
            print('Unitary?   %s.' % isunitary(matrix, ct))
```

Output:

```
Matrix:
3+0j, 2+1j
2-1j, 1+0j
Its conjugate transpose:
3-0j, 2+1j
2-1j, 1-0j
Hermitian? True.
Normal?    True.
Unitary?   False.

Matrix:
1+0j, 1+0j, 0+0j
0+0j, 1+0j, 1+0j
```

Python: Connect_to_Active_Directory[edit]

Works with: Python version 2.6

Library: python-ldap

python-ldap Documentation

```
import ldap

l = ldap.initialize("ldap://ldap.example.com")
try:
    l.protocol_version = ldap.VERSION3
    l.set_option(ldap.OPT_REFERRALS, 0)

    bind = l.simple_bind_s("me@example.com", "password")
finally:
    l.unbind()
```

Racket[edit]

Python: Constrained_random_points_on_a_circle[

Note that the diagram shows the number of points at any given position

```
>>> from collections import defaultdict
>>> from random import choice
>>> world = defaultdict(int)
>>> possiblepoints = [(x,y) for x in range(-15,16)
                  for y in range(-15,16)
                  if 10 <= abs(x+y*1j) <= 15]
>>> for i in range(100): world[choice(possiblepoints)] += 1

>>> for x in range(-15,16):
        print(''.join(str(min([9, world[(x,y)]])) if world[(x,y)] else
                          for y in range(-15,16)))
```

```

             1     1
          1 1
      11 1     1  1     1
     111  1     1211
      1   2    1 1    11
      1  11         21
     1   1            11  1
   1  2                1 1

 1  2
   1 1                      1
   1 1
   2                      11
  1                         1
                         1


  1                          1
                         1
                         2
                            1
     1                  1 1
      1                2   1
   1   3            11  2
    11   1    1      1   2
            1   1    2
        1  1
         1      1     1
          2 2   1
               1
```

If the number of samples is increased to 1100:

```
>>> for i in range(1000): world[choice(possiblepoints)] += 1

>>> for x in range(-15,16):
        print(''.join(str(min([9, world[(x,y)]])) if world[(x,y)] else
                          for y in range(-15,16)))
```

```
               2
          41341421333
        5133333131253 1
      5231514 14214721 24
     326 2122214323412232 2
    5423515313212334412 5 22
   32331432         2422 33
   5453135           4144344
  132595               323123
  4 6353               432224
 5 4323                 3 5313
 23214                   41433
 42454                   33342
 332 4                   34314
 142 1                   35 53
124211                   53131
 22221                   152 4
 22213                   34562
 654 4                   4 212
 24354                   52232
 544222                 283323
  411123               453325
  251321               124332
   2124134           2443226
   2 113315         64324334
    2412452 324 32121132363
      422243432 4635 5433
      311333123432112633
        2131181233  424
          47414232164
               4
```

Python: Convert_an_integer_into_words[edit]

Note: This example is also used as a module in the Names to numbers#Py

```
TENS = [None, None, "twenty", "thirty", "forty",
        "fifty", "sixty", "seventy", "eighty", "ninety"]
SMALL = ["zero", "one", "two", "three", "four", "five",
         "six", "seven", "eight", "nine", "ten", "eleven",
         "twelve", "thirteen", "fourteen", "fifteen",
         "sixteen", "seventeen", "eighteen", "nineteen"]
HUGE = [None, None] + [h + "illion"
                       for h in ("m", "b", "tr", "quadr", "quint", "se
                                 "sept", "oct", "non", "dec")]

def nonzero(c, n, connect=''):
```

```
    return "" if n == 0 else connect + c + spell_integer(n)

def last_and(num):
    if ',' in num:
        pre, last = num.rsplit(',', 1)
        if ' and ' not in last:
            last = ' and' + last
        num = ''.join([pre, ',', last])
    return num

def big(e, n):
    if e == 0:
        return spell_integer(n)
    elif e == 1:
        return spell_integer(n) + " thousand"
    else:
        return spell_integer(n) + " " + HUGE[e]

def base1000_rev(n):
    # generates the value of the digits of n in base 1000
    # (i.e. 3-digit chunks), in reverse.
    while n != 0:
        n, r = divmod(n, 1000)
        yield r

def spell_integer(n):
    if n < 0:
        return "minus " + spell_integer(-n)
    elif n < 20:
        return SMALL[n]
    elif n < 100:
        a, b = divmod(n, 10)
        return TENS[a] + nonzero("-", b)
    elif n < 1000:
        a, b = divmod(n, 100)
        return SMALL[a] + " hundred" + nonzero(" ", b, ' and')
    else:
        num = ", ".join([big(e, x) for e, x in
                         enumerate(base1000_rev(n)) if x][::-1])
        return last_and(num)

if __name__ == '__main__':
    # examples
    for n in (0, -3, 5, -7, 11, -13, 17, -19, 23, -29):
        print('%+4i -> %s' % (n, spell_integer(n)))
    print('')

    n = 201021002001
    while n:
        print('%-12i -> %s' % (n, spell_integer(n)))
        n //= -10
    print('%-12i -> %s' % (n, spell_integer(n)))
    print('')
```

Output:

```
  +0 -> zero
  -3 -> minus three
  +5 -> five
  -7 -> minus seven
 +11 -> eleven
 -13 -> minus thirteen
 +17 -> seventeen
 -19 -> minus nineteen
 +23 -> twenty-three
 -29 -> minus twenty-nine

201021002001 -> two hundred and one billion, twenty-one million, two t
-20102100201 -> minus twenty billion, one hundred and two million, one
2010210020   -> two billion, ten million, two hundred and ten thousand
-201021002   -> minus two hundred and one million, twenty-one thousand
20102100     -> twenty million, one hundred and two thousand, and one
-2010210     -> minus two million, ten thousand, two hundred and ten
201021       -> two hundred and one thousand, and twenty-one
-20103       -> minus twenty thousand, one hundred and three
2010         -> two thousand, and ten
-201         -> minus two hundred and one
20           -> twenty
-2           -> minus two
0            -> zero
```

Python: Convert_decimal_number[edit]

Works with: Python version 2.6+

Note that the decimal values of the task description are truncated in

The first loop limits the size of the denominator, because the floatin

```
>>> from fractions import Fraction
>>> for d in (0.9054054, 0.518518, 0.75): print(d, Fraction.from_float

0.9054054 67/74
0.518518 14/27
0.75 3/4
>>> for d in '0.9054054 0.518518 0.75'.split(): print(d, Fraction(d))

0.9054054 4527027/5000000
0.518518 259259/500000
0.75 3/4
>>>
```

Racket[edit]

Python: Convert_decimal_number_to_rational[edi

Works with: Python version 2.6+

Note that the decimal values of the task description are truncated in

The first loop limits the size of the denominator, because the floatin

```
>>> from fractions import Fraction
>>> for d in (0.9054054, 0.518518, 0.75): print(d, Fraction.from_float

0.9054054 67/74
0.518518 14/27
0.75 3/4
>>> for d in '0.9054054 0.518518 0.75'.split(): print(d, Fraction(d))

0.9054054 4527027/5000000
0.518518 259259/500000
0.75 3/4
>>>
```

Python: Conway's_Game_of_Life.1[edit]

Using defaultdict[edit]

This implementation uses defaultdict(int) to create dictionaries that
This 'trick allows celltable to be initialized to just those keys
with a value of 1.

Python allows many types other than strings and ints to be keys
in a dictionary.
The example uses a dictionary with keys that are a two entry tuple to
which also returns a default value of zero.
This simplifies the calculation **N** as out-of-bounds indexing
of universe returns zero.

```
import random
from collections import defaultdict

printdead, printlive = '-#'
maxgenerations = 3
cellcount = 3,3
celltable = defaultdict(int, {
 (1, 2): 1,
 (1, 3): 1,
 (0, 3): 1,
 } ) # Only need to populate with the keys leading to life

##
## Start States
##
# blinker
u = universe = defaultdict(int)
u[(1,0)], u[(1,1)], u[(1,2)] = 1,1,1

## toad
#u = universe = defaultdict(int)
#u[(5,5)], u[(5,6)], u[(5,7)] = 1,1,1
#u[(6,6)], u[(6,7)], u[(6,8)] = 1,1,1

## glider
#u = universe = defaultdict(int)
#maxgenerations = 16
#u[(5,5)], u[(5,6)], u[(5,7)] = 1,1,1
#u[(6,5)] = 1
#u[(7,6)] = 1

## random start
#universe = defaultdict(int,
#                       # array of random start values
#                       ( ((row, col), random.choice((0,1)))
#                         for col in range(cellcount[0])
#                         for row in range(cellcount[1])
#                       ) )  # returns 0 for out of bounds

for i in range(maxgenerations):
    print "\nGeneration %3i:" % ( i, )
    for row in range(cellcount[1]):
        print "  ", ''.join(str(universe[(row,col)])
                            for col in range(cellcount[0])).replace(
                                '0', printdead).replace('1', printlive
    nextgeneration = defaultdict(int)
    for row in range(cellcount[1]):
        for col in range(cellcount[0]):
            nextgeneration[(row,col)] = celltable[
                ( universe[(row,col)],
                  -universe[(row,col)] + sum(universe[(r,c)]
                                             for r in range(row-1,row+
                                             for c in range(col-1, col
                ) ]
    universe = nextgeneration
```

Output:

(sample)

```
Generation   0:
   ---
   ###
   ---

Generation   1:
   -#-
   -#-
   -#-

Generation   2:
   ---
   ###
```

Boardless approach[edit]

A version using the boardless approach.
A world is represented as a set of (x, y) coordinates of all the alive

```
from collections import Counter

def life(world, N):
    "Play Conway's game of life for N generations from initial world."
    for g in range(N+1):
        display(world, g)
        counts = Counter(n for c in world for n in offset(neighboring_
        world = {c for c in counts
                if counts[c] == 3 or (counts[c] == 2 and c in world)}

neighboring_cells = [(-1, -1), (-1, 0), (-1, 1),
                     ( 0, -1),          ( 0, 1),
                     ( 1, -1), ( 1, 0), ( 1, 1)]

def offset(cells, delta):
    "Slide/offset all the cells by delta, a (dx, dy) vector."
    (dx, dy) = delta
    return {(x+dx, y+dy) for (x, y) in cells}

def display(world, g):
    "Display the world as a grid of characters."
    print '          GENERATION {}:'.format(g)
    Xs, Ys = zip(*world)
```

```
    Xrange = range(min(Xs), max(Xs)+1)
    for y in range(min(Ys), max(Ys)+1):
        print ''.join('#' if (x, y) in world else '.'
                      for x in Xrange)

blinker = {(1, 0), (1, 1), (1, 2)}
block   = {(0, 0), (1, 1), (0, 1), (1, 0)}
toad    = {(1, 2), (0, 1), (0, 0), (0, 2), (1, 3), (1, 1)}
glider  = {(0, 1), (1, 0), (0, 0), (0, 2), (2, 1)}
world   = (block | offset(blinker, (5, 2)) | offset(glider, (15, 5)) |
           | {(18, 2), (19, 2), (20, 2), (21, 2)} | offset(block, (35,


life(world, 5)
```

Output:

```
          GENERATION 0:
##................................
##................................
......#..........####.............
......#...........................
......#...........................
...............##........#..........
...............#.#.......##.........
...............#.........##.......##
..........................#.......##
          GENERATION 1:
##................................
##................##...............
```

Python: Conway's_Game_of_life[edit]

Using defaultdict[edit]

This implementation uses defaultdict(int) to create dictionaries that
This 'trick allows celltable to be initialized to just those keys
with a value of 1.

Python allows many types other than strings and ints to be keys
in a dictionary.
The example uses a dictionary with keys that are a two entry tuple to
which also returns a default value of zero.
This simplifies the calculation **N** as out-of-bounds indexing

of universe returns zero.

```
import random
from collections import defaultdict

printdead, printlive = '-#'
maxgenerations = 3
cellcount = 3,3
celltable = defaultdict(int, {
 (1, 2): 1,
 (1, 3): 1,
 (0, 3): 1,
 } ) # Only need to populate with the keys leading to life

##
## Start States
##
# blinker
u = universe = defaultdict(int)
u[(1,0)], u[(1,1)], u[(1,2)] = 1,1,1

## toad
#u = universe = defaultdict(int)
#u[(5,5)], u[(5,6)], u[(5,7)] = 1,1,1
#u[(6,6)], u[(6,7)], u[(6,8)] = 1,1,1

## glider
#u = universe = defaultdict(int)
#maxgenerations = 16
#u[(5,5)], u[(5,6)], u[(5,7)] = 1,1,1
#u[(6,5)] = 1
#u[(7,6)] = 1

## random start
#universe = defaultdict(int,
#                        # array of random start values
#                        ( ((row, col), random.choice((0,1)))
#                          for col in range(cellcount[0])
#                          for row in range(cellcount[1])
#                        ) )  # returns 0 for out of bounds

for i in range(maxgenerations):
    print "\nGeneration %3i:" % ( i, )
    for row in range(cellcount[1]):
        print "  ", ''.join(str(universe[(row,col)])
                            for col in range(cellcount[0])).replace(
                                '0', printdead).replace('1', printlive
    nextgeneration = defaultdict(int)
    for row in range(cellcount[1]):
        for col in range(cellcount[0]):
            nextgeneration[(row,col)] = celltable[
                ( universe[(row,col)],
                  -universe[(row,col)] + sum(universe[(r,c)]
                                             for r in range(row-1,row+
```

```
                                                    for c in range(col-1, col
             ) ]
    universe = nextgeneration
```

Output:

```
 (sample)

Generation   0:
   ---
   ###
   ---

Generation   1:
   -#-
   -#-
   -#-

Generation   2:
   ---
   ###
```

Boardless approach[edit]

A version using the boardless approach.
A world is represented as a set of (x, y) coordinates of all the alive

```
from collections import Counter

def life(world, N):
    "Play Conway's game of life for N generations from initial world."
    for g in range(N+1):
        display(world, g)
        counts = Counter(n for c in world for n in offset(neighboring_
        world = {c for c in counts
                if counts[c] == 3 or (counts[c] == 2 and c in world)}

neighboring_cells = [(-1, -1), (-1, 0), (-1, 1),
                     ( 0, -1),          ( 0, 1),
                     ( 1, -1), ( 1, 0), ( 1, 1)]

def offset(cells, delta):
    "Slide/offset all the cells by delta, a (dx, dy) vector."
    (dx, dy) = delta
    return {(x+dx, y+dy) for (x, y) in cells}
```

```
def display(world, g):
    "Display the world as a grid of characters."
    print '          GENERATION {}:'.format(g)
    Xs, Ys = zip(*world)
    Xrange = range(min(Xs), max(Xs)+1)
    for y in range(min(Ys), max(Ys)+1):
        print ''.join('#' if (x, y) in world else '.'
                      for x in Xrange)

blinker = {(1, 0), (1, 1), (1, 2)}
block   = {(0, 0), (1, 1), (0, 1), (1, 0)}
toad    = {(1, 2), (0, 1), (0, 0), (0, 2), (1, 3), (1, 1)}
glider  = {(0, 1), (1, 0), (0, 0), (0, 2), (2, 1)}
world   = (block | offset(blinker, (5, 2)) | offset(glider, (15, 5)) |
           | {(18, 2), (19, 2), (20, 2), (21, 2)} | offset(block, (35,


life(world, 5)
```

Output:

```
          GENERATION 0:
##...................................
##...................................
......#...........####...............
......#..............................
......#..............................
...............##........#...........
...............#.#.......##..........
...............#.........##........##
..........................#........##
          GENERATION 1:
##...................................
##..................##...............
```

Python: Copy_a_string[edit]

Works with: Python version 2.3, 2.4, and 2.5

Since strings are immutable, all copy operations return the same strin

```
>>> src = "hello"
>>> a = src
>>> b = src[:]
```

```
>>> import copy
>>> c = copy.copy(src)
>>> d = copy.deepcopy(src)
>>> src is a is b is c is d
True
```

To actually copy a string:

```
>>> a = 'hello'
>>> b = ''.join(a)
>>> a == b
True
>>> b is a  ### Might be True ... depends on "interning" implementatio
False
```

As a result of object "interning" some strings such as the empty strin

Be careful with *is* - use it only when you want to compare the identity

R[edit]

Python: Count_in_factors[edit]

This uses the functools.lru_cache standard library module to cache int

```
from functools import lru_cache

primes = [2, 3, 5, 7, 11, 13, 17]    # Will be extended

@lru_cache(maxsize=2000)
def pfactor(n):
    if n == 1:
        return [1]
    n2 = n // 2 + 1
    for p in primes:
        if p <= n2:
            d, m = divmod(n, p)
            if m == 0:
                if d > 1:
                    return [p] + pfactor(d)
                else:
                    return [p]
```

```
        else:
            if n > primes[-1]:
                primes.append(n)
            return [n]

if __name__ == '__main__':
    mx = 5000
    for n in range(1, mx + 1):
        factors = pfactor(n)
        if n <= 10 or n >= mx - 20:
            print( '%4i %5s %s' % (n,
                                   '' if factors != [n] or n == 1 else
                                   'x'.join(str(i) for i in factors)) )
        if n == 11:
            print('...')

    print('\nNumber of primes gathered up to', n, 'is', len(primes))
    print(pfactor.cache_info())
```

```
Output:
```

```
   1       1
   2 prime 2
   3 prime 3
   4       2x2
   5 prime 5
```

Python: Count_in_octal[edit]

```
import sys
for n in xrange(sys.maxint):
    print oct(n)
```

Racket[edit]

Python: Count_occurrences_of_a_substring[edit]

```
>>> "the three truths".count("th")
3
>>> "ababababab".count("abab")
2
```

Python: Count_programming_examples.1[edit]

```
import urllib, xml.dom.minidom

x = urllib.urlopen("http://www.rosettacode.org/w/api.php?action=query&

tasks = []
for i in xml.dom.minidom.parseString(x.read()).getElementsByTagName("c
    t = i.getAttribute('title').replace(" ", "_")
    y = urllib.urlopen("http://www.rosettacode.org/w/index.php?title=%
    tasks.append( y.read().lower().count("{{header|") )
    print t.replace("_", " ") + ": %d examples." % tasks[-1]

print "\nTotal: %d examples." % sum(tasks)
```

Python: Count_the_coins[edit]

Simple version[edit]

Python: Counting_sort[edit]

Follows the spirit of the counting sort but uses Pythons defaultdict(i

```
>>> from collections import defaultdict
>>> def countingSort(array, mn, mx):
        count = defaultdict(int)
        for i in array:
                count[i] += 1
        result = []
        for j in range(mn,mx+1):
                result += [j]* count[j]
        return result

>>> data = [9, 7, 10, 2, 9, 7, 4, 3, 10, 2, 7, 10, 2, 1, 3, 8, 7, 3, 9
>>> mini,maxi = 1,10
>>> countingSort(data, mini, maxi) == sorted(data)
True
```

Using a list:

Works with: Python version 2.6

```
def countingSort(a, min, max):
    cnt = [0] * (max - min + 1)
    for x in a:
        cnt[x - min] += 1

    return [x for x, n in enumerate(cnt, start=min)
              for i in xrange(n)]
```

R[edit]

Python: Create_a_Hash[edit]

Hashes are a built-in type called dictionaries (or mappings) in Python

```
hash = dict()  # 'dict' is the dictionary type.
hash = dict(red="FF0000", green="00FF00", blue="0000FF")
hash = { 'key1':1, 'key2':2, }
value = hash[key]
```

Numerous methods exist for the mapping type http://docs.python.org/lib

```
# empty dictionary
d = {}
d['spam'] = 1
d['eggs'] = 2

# dictionaries with two keys
d1 = {'spam': 1, 'eggs': 2}
d2 = dict(spam=1, eggs=2)

# dictionaries from tuple list
d1 = dict([('spam', 1), ('eggs', 2)])
d2 = dict(zip(['spam', 'eggs'], [1, 2]))

# iterating over keys
for key in d:
  print key, d[key]
```

```
# iterating over (key, value) pairs
for key, value in d.iteritems():
  print key, value
```

Note: Python dictionary keys can be of any arbitrary "hashable" type.

```
myDict = { '1': 'a string', 1: 'an integer', 1.0: 'a floating point nu
```

(Some other languages such as *awk* and *Perl* evaluate all keys such that

User defined classes which implement the *__hash__()* special method can
(accessible via the *id()* built-in function) is commonly used for this

Python: Create_a_Sequence_of_unique_elements[e

If all the elements are *hashable* (this excludes *list*, *dict*, *set*, and o

```
items = [1, 2, 3, 'a', 'b', 'c', 2, 3, 4, 'b', 'c', 'd']
unique = list(set(items))
```

or if we want to keep the order of the elements

```
items = [1, 2, 3, 'a', 'b', 'c', 2, 3, 4, 'b', 'c', 'd']
unique = []
helperset = set()
for x in items:
    if x not in helperset:
        unique.append(x)
        helperset.add(x)
```

If all the elements are comparable (i.e. <, >=, etc. operators; this w

```
import itertools
items = [1, 2, 3, 'a', 'b', 'c', 2, 3, 4, 'b', 'c', 'd']
unique = [k for k,g in itertools.groupby(sorted(items))]
```

If both of the above fails, we have to use the brute-force method, whi

```
items = [1, 2, 3, 'a', 'b', 'c', 2, 3, 4, 'b', 'c', 'd']
unique = []
for x in items:
    if x not in unique:
        unique.append(x)
```

Python: Create_a_file[edit]

```
import os
for directory in ['/', './']:
  open(directory + 'output.txt', 'w').close()  # create /output.txt, t
  os.mkdir(directory + 'docs')                 # create directory /doc
```

Works with: Python version 2.5

Exception-safe way to create file:

```
from __future__ import with_statement
import os
def create(directory):
    with open(os.path.join(directory, "output.txt"), "w"):
        pass
    os.mkdir(os.path.join(directory, "docs"))

create(".") # current directory
create("/") # root directory
```

R[edit]

Python: Create_a_file_on_magnetic_tape[edit]

```
>>> with open('/dev/tape', 'w') as t: t.write('Hi Tape!\n')
...
>>>
```

Racket[edit]

Python: Create_a_two-dimensional_array_at_runt

Works with: Python version 2.5

Python: Create_an_HTML_table[edit]

```
import random

def rand9999():
    return random.randint(1000, 9999)

def tag(attr='', **kwargs):
    for tag, txt in kwargs.items():
        return '<{tag}{attr}>{txt}</{tag}>'.format(**locals())

if __name__ == '__main__':
    header = tag(tr=''.join(tag(th=txt) for txt in ',X,Y,Z'.split(','))
    rows = '\n'.join(tag(tr=''.join(tag(' style="font-weight: bold;"',
                                    + ''.join(tag(td=rand9999())
                                              for j in range(3))))
                     for i in range(1, 6))
    table = tag(table='\n' + header + rows + '\n')
    print(table)
```

Sample output

Append Capabilities.

Data Representation	IO Library	Append Possible

Python: Creating_a_Hash_from_Two_Arrays[edit]

Works with: Python version 3.0+ and 2.7

Shows off the dict comprehensions in Python 3 (that were back-ported to 2.7

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
```

```
hash = {key: value for key, value in zip(keys, values)}
```

Works with: Python version 2.2+

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
hash = dict(zip(keys, values))

# Lazily, Python 2.3+, not 3.x:
from itertools import izip
hash = dict(izip(keys, values))
```

Works with: Python version 2.0+

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
hash = {}
for k,v in zip(keys, values):
    hash[k] = v
```

The original (Ruby) example uses a range of different types as keys. Here i

```
>>> class Hashable(object):
        def __hash__(self):
                return id(self) ^ 0xBEEF



>>> my_inst = Hashable()
>>> my_int = 1
>>> my_complex = 0 + 1j
>>> my_float = 1.2
>>> my_string = "Spam"
>>> my_bool = True
>>> my_unicode = u'Ham'
>>> my_list = ['a', 7]
>>> my_tuple = ( 0.0, 1.4 )
>>> my_set = set(my_list)
>>> def my_func():
        pass

>>> class my_class(object):
        pass

>>> keys = [my_inst, my_tuple, my_int, my_complex, my_float, my_string,
        my_bool, my_unicode, frozenset(my_set), tuple(my_list),
        my_func, my_class]
>>> values = range(12)
>>> d = dict(zip(keys, values))
>>> for key, value in d.items(): print key, ":", value

1 : 6
1j : 3
Ham : 7
Spam : 5
(0.0, 1.3999999999999999) : 1
frozenset(['a', 7]) : 8
```

```
1.2 : 4
('a', 7) : 9
<function my_func at 0x0128E7B0> : 10
<class '__main__.my_class'> : 11
<__main__.Hashable object at 0x012AFC50> : 0
>>> # Notice that the key "True" disappeared, and its value got associated w
>>> # This is because 1 == True in Python, and dictionaries cannot have two
```

Python: Creating_a_SOAP_Client[edit]

Works with: Python version 2.4 and 2.5

```
from SOAPpy import WSDL
proxy = WSDL.Proxy("http://example.com/soap/wsdl")
result = proxy.soapFunc("hello")
result = proxy.anotherSoapFunc(34234)
```

Note: SOAPpy is a third-party module and can be found at Python Web Service

Ruby[edit]

Python: Creating_a_Window[edit]

Works with: Python version 2.4 and 2.5
Library: Tkinter

```
  import Tkinter

  w = Tkinter.Tk()
  w.mainloop()
```

Library: wxPython

Python: Creating_a_function[edit]

Function definition:

```
def multiply(a, b):
    return a * b
```

Lambda function definition:

```
multiply = lambda a, b: a * b
```

A callable class definition allows functions and classes to use the same in

```
class Multiply:
    def __init__(self):
        pass
    def __call__(self, a, b):
        return a * b

multiply = Multiply()
print multiply(2, 4)    # prints 8
```

(No extra functionality is shown in *this* class definition).

Python: Creating_an_Array[edit]

List are mutable arrays. You can put anything into a list, including other

```
empty = []
numbers = [1, 2, 3, 4, 5]
zeros = [0] * 10
anything = [1, 'foo', 2.57, None, zeros]
digits = range(10)  # 0, 1 ... 9
evens  = range(0,10,2)  # 0, 2, 4 ... 8
evens = [x for x in range(10) if not x % 2]  # same using list comprehensi
words = 'perl style'.split()
```

Tuples are immutable arrays. Note that tuples are defined by the "," - the

```
empty = ()
numbers = (1, 2, 3, 4, 5)
numbers2 = 1,2,3,4,5           # same as previous
zeros = (0,) * 10
anything = (1, 'foo', 2.57, None, zeros)
```

Both lists and tuples can be created from other iterateables:

```
>>> list('abc')
['a', 'b', 'c']
```

Python: Creating_an_Associative_Array[edit]

Hashes are a built-in type called dictionaries (or mappings) in Python.

```
hash = dict()  # 'dict' is the dictionary type.
hash = dict(red="FF0000", green="00FF00", blue="0000FF")
hash = { 'key1':1, 'key2':2, }
value = hash[key]
```

Numerous methods exist for the mapping type http://docs.python.org/lib/type

```
# empty dictionary
d = {}
d['spam'] = 1
d['eggs'] = 2

# dictionaries with two keys
d1 = {'spam': 1, 'eggs': 2}
d2 = dict(spam=1, eggs=2)
```

```
# dictionaries from tuple list
d1 = dict([('spam', 1), ('eggs', 2)])
d2 = dict(zip(['spam', 'eggs'], [1, 2]))

# iterating over keys
for key in d:
  print key, d[key]

# iterating over (key, value) pairs
for key, value in d.iteritems():
  print key, value
```

Note: Python dictionary keys can be of any arbitrary "hashable" type. The f

```
myDict = { '1': 'a string', 1: 'an integer', 1.0: 'a floating point number'
```

(Some other languages such as *awk* and *Perl* evaluate all keys such that nume

User defined classes which implement the *__hash__()* special method can also
(accessible via the *id()* built-in function) is commonly used for this purpo

R[edit]

Python: Cubic_bezier_curves[edit]

Works with: Python version 3.1

Extending the example given here and using the algorithm from the C solutio

```
def cubicbezier(self, x0, y0, x1, y1, x2, y2, x3, y3, n=20):
    pts = []
    for i in range(n+1):
        t = i / n
        a = (1. - t)**3
        b = 3. * t * (1. - t)**2
        c = 3.0 * t**2 * (1.0 - t)
        d = t**3

        x = int(a * x0 + b * x1 + c * x2 + d * x3)
        y = int(a * y0 + b * y1 + c * y2 + d * y3)
        pts.append( (x, y) )
    for i in range(n):
```

```
        self.line(pts[i][0], pts[i][1], pts[i+1][0], pts[i+1][1])
Bitmap.cubicbezier = cubicbezier

bitmap = Bitmap(17,17)
bitmap.cubicbezier(16,1, 1,4, 3,16, 15,11)
bitmap.chardisplay()


'''
The origin, 0,0; is the lower left, with x increasing to the right,
and Y increasing upwards.

The chardisplay above produces the following output :
+-----------------+
|                 |
|                 |
|                 |
|                 |
|         @@@@    |
|      @@@    @@@ |
|     @           |
|     @           |
|     @           |
|     @           |
|      @          |
|      @          |
|       @         |
|        @        |
|         @@@@    |
|             @@@@|
|                 |
+-----------------+
'''
```

Python: Currency[edit]

This uses Pythons decimal module,
(and some copying of names from the Perl 6 example).

```
from decimal import Decimal as D
from collections import namedtuple

Item = namedtuple('Item', 'price, quant')

items = dict( hamburger=Item(D('5.50'), D('4000000000000000')),
              milkshake=Item(D('2.86'), D('2')) )
tax_rate = D('0.0765')

fmt = "%-10s %8s %18s %22s"
```

```
print(fmt % tuple('Item Price Quantity Extension'.upper().split()))

total_before_tax = 0
for item, (price, quant) in sorted(items.items()):
    ext = price * quant
    print(fmt % (item, price, quant, ext))
    total_before_tax += ext
print(fmt % ('', '', '', '--------------------'))
print(fmt % ('', '', 'subtotal', total_before_tax))

tax = (tax_rate * total_before_tax).quantize(D('0.00'))
print(fmt % ('', '', 'Tax', tax))

total = total_before_tax + tax
print(fmt % ('', '', '', '--------------------'))
print(fmt % ('', '', 'Total', total))
```

Python: Currying[edit]

```
def addN(n):
    def adder(x):
        return x + n
    return adder
```

```
>>> add2 = addN(2)
>>> add2
<function adder at 0x009F1E30>
>>> add2(7)
9
```

Python: Cut_a_rectangle[edit]

Translation of: D

```
def cut_it(h, w):
    dirs = ((1, 0), (-1, 0), (0, -1), (0, 1))
    if h & 1: h, w = w, h
    if h & 1: return 0
    if w == 1: return 1
    count = 0

    next = [w + 1, -w - 1, -1, 1]
    blen = (h + 1) * (w + 1) - 1
    grid = [False] * (blen + 1)
```

```
    def walk(y, x, count):
        if not y or y == h or not x or x == w:
            return count + 1

        t = y * (w + 1) + x
        grid[t] = grid[blen - t] = True

        if not grid[t + next[0]]:
            count = walk(y + dirs[0][0], x + dirs[0][1], count)
        if not grid[t + next[1]]:
            count = walk(y + dirs[1][0], x + dirs[1][1], count)
        if not grid[t + next[2]]:
            count = walk(y + dirs[2][0], x + dirs[2][1], count)
        if not grid[t + next[3]]:
            count = walk(y + dirs[3][0], x + dirs[3][1], count)

        grid[t] = grid[blen - t] = False
        return count

    t = h // 2 * (w + 1) + w // 2
    if w & 1:
        grid[t] = grid[t + 1] = True
        count = walk(h // 2, w // 2 - 1, count)
        res = count
        count = 0
        count = walk(h // 2 - 1, w // 2, count)
        return res + count * 2
    else:
        grid[t] = True
        count = walk(h // 2, w // 2 - 1, count)
        if h == w:
            return count * 2
        count = walk(h // 2 - 1, w // 2, count)
        return count

def main():
    for w in xrange(1, 10):
        for h in xrange(1, w + 1):
            if not((w * h) & 1):
                print "%d x %d: %d" % (w, h, cut_it(w, h))

main()
```

Python: DNS_query[edit]

```
>>> import socket
>>> ips = set(i[4][0] for i in socket.getaddrinfo('www.kame.net', 80))
>>> for ip in ips: print ip
...
2001:200:dff:fff1:216:3eff:feb1:44d7
203.178.141.194
```

Python: Data_Munging[edit]

```
import fileinput
import sys

nodata = 0;             # Current run of consecutive flags<0 in lines of fi
nodata_max=-1;          # Max consecutive flags<0 in lines of file
nodata_maxline=[];      # ... and line number(s) where it occurs

tot_file = 0            # Sum of file data
num_file = 0            # Number of file data items with flag>0

infiles = sys.argv[1:]

for line in fileinput.input():
  tot_line=0;             # sum of line data
  num_line=0;             # number of line data items with flag>0

  # extract field info
  field = line.split()
  date  = field[0]
  data  = [float(f) for f in field[1::2]]
  flags = [int(f)   for f in field[2::2]]

  for datum, flag in zip(data, flags):
    if flag<1:
      nodata += 1
    else:
      # check run of data-absent fields
      if nodata_max==nodata and nodata>0:
        nodata_maxline.append(date)
      if nodata_max<nodata and nodata>0:
        nodata_max=nodata
        nodata_maxline=[date]
      # re-initialise run of nodata counter
      nodata=0;
      # gather values for averaging
      tot_line += datum
      num_line += 1

  # totals for the file so far
  tot_file += tot_line
  num_file += num_line

  print "Line: %11s  Reject: %2i  Accept: %2i  Line_tot: %10.3f  Line_avg: 
        date,
        len(data) -num_line,
        num_line, tot_line,
        tot_line/num_line if (num_line>0) else 0)

print ""
```

```
print "File(s)  = %s" % (", ".join(infiles),)
print "Total    = %10.3f" % (tot_file,)
print "Readings = %6i" % (num_file,)
print "Average  = %10.3f" % (tot_file / num_file,)

print "\nMaximum run(s) of %i consecutive false readings ends at line start
    nodata_max, ", ".join(nodata_maxline))
```

Sample output:

```
bash$ /cygdrive/c/Python26/python readings.py readings.txt|tail
Line:  2004-12-29  Reject:  1  Accept: 23  Line_tot:     56.300  Line_avg:
Line:  2004-12-30  Reject:  1  Accept: 23  Line_tot:     65.300  Line_avg:
Line:  2004-12-31  Reject:  1  Accept: 23  Line_tot:     47.300  Line_avg:

File(s)  = readings.txt
Total    = 1358393.400
Readings = 129403
Average  =     10.497

Maximum run(s) of 589 consecutive false readings ends at line starting with
bash$
```

Python: Data_Munging_2[edit]

```
import re
import zipfile
import StringIO

def munge2(readings):

   datePat = re.compile(r'\d{4}-\d{2}-\d{2}')
   valuPat = re.compile(r'[-+]?\d+\.\d+')
   statPat = re.compile(r'-?\d+')
   allOk, totalLines = 0, 0
   datestamps = set([])
   for line in readings:
      totalLines += 1
      fields = line.split('\t')
      date = fields[0]
      pairs = [(fields[i],fields[i+1]) for i in range(1,len(fields),2)]

      lineFormatOk = datePat.match(date) and \
         all( valuPat.match(p[0]) for p in pairs ) and \
         all( statPat.match(p[1]) for p in pairs )
      if not lineFormatOk:
         print 'Bad formatting', line
```

```
            continue

        if len(pairs)!=24 or any( int(p[1]) < 1 for p in pairs ):
            print 'Missing values', line
            continue

        if date in datestamps:
            print 'Duplicate datestamp', line
            continue
        datestamps.add(date)
        allOk += 1

    print 'Lines with all readings: ', allOk
    print 'Total records: ', totalLines

#zfs = zipfile.ZipFile('readings.zip','r')
#readings = StringIO.StringIO(zfs.read('readings.txt'))
readings = open('readings.txt','r')
munge2(readings)
```

The results indicate 5013 good records, which differs from the Awk implemen

```
Missing values 2004-12-29       2.900   1       2.700   1       2.800

Missing values 2004-12-30       2.400   1       2.600   1       2.600
```

Second Version

Modification of the version above to:

Python: Data_Representation_-_Controlling_Fields_

The ctypes module allows for the creation of Structures that can map betwee

```
from ctypes import Structure, c_int

rs232_9pin  = "_0 CD RD TD DTR SG DSR RTS CTS RI".split()
rs232_25pin = ( "_0  PG  TD  RD  RTS CTS DSR SG  CD  pos neg"
                "_11 SCD SCS STD TC  SRD RC"
                "_18 SRS DTR SQD RI DRS XTC" ).split()

class RS232_9pin(Structure):
    _fields_ = [(__, c_int, 1) for __ in rs232_9pin]
```

```
class RS232_25pin(Structure):
    _fields_ = [(__, c_int, 1) for __ in rs232_25pin]
```

Racket[edit]

Python: Data_Representation_-_Getting_the_Size[ed

This information is only easily available for the array type:

```
>>> from array import array
>>> argslist = [('l', []), ('c', 'hello world'), ('u', u'hello \u2641'),
        ('l', [1, 2, 3, 4, 5]), ('d', [1.0, 2.0, 3.14])]
>>> for typecode, initializer in argslist:
        a = array(typecode, initializer)
        print a, '\tSize =', a.buffer_info()[1] * a.itemsize
        del a


array('l')      Size = 0
array('c', 'hello world')       Size = 11
array('u', u'hello \u2641')     Size = 14
array('l', [1, 2, 3, 4, 5])     Size = 20
array('d', [1.0, 2.0, 3.1400000000000001])      Size = 24
>>>
```

Python: Data_Representation_-_Specifying_Minimum_

For compatibility with the calling conventions of external C functions, the

	X	Y	Z
1	6040	4697	7055

Python: Date_Manipulation[edit]

I don't do anything with timezone here, but it is possible.

```
import datetime

def mt():
```

```
        datime1="March 7 2009 7:30pm EST"
        formatting = "%B %d %Y %I:%M%p "
        datime2 = datime1[:-3]  # format can't handle "EST" for some reason
        tdelta = datetime.timedelta(hours=12)           # twelve hours..
        s3 = datetime.datetime.strptime(datime2, formatting)
        datime2 = s3+tdelta
        print datime2.strftime("%B %d %Y %I:%M%p %Z") + datime1[-3:]

mt()
```

R[edit]

Python: Date_format[edit]

Formatting rules: http://docs.python.org/lib/module-time.html (strftime)

```
import datetime
today = datetime.date.today()
# This one is built in:
print today.isoformat()
# Or use a format string for full flexibility:
print today.strftime('%Y-%m-%d')
```

Python: Date_manipulation[edit]

I don't do anything with timezone here, but it is possible.

```
import datetime

def mt():
        datime1="March 7 2009 7:30pm EST"
        formatting = "%B %d %Y %I:%M%p "
        datime2 = datime1[:-3]  # format can't handle "EST" for some reason
        tdelta = datetime.timedelta(hours=12)           # twelve hours..
        s3 = datetime.datetime.strptime(datime2, formatting)
        datime2 = s3+tdelta
        print datime2.strftime("%B %d %Y %I:%M%p %Z") + datime1[-3:]

mt()
```

Python: Day_of_the_week[edit]

```
from datetime import date

for year in range(2008, 2122):
    day = date(year, 12, 25)
    if day.weekday() == 6:
        print(day.strftime('%d %b %Y'))
```

Output:

Python: Deal_cards_for_FreeCell[edit]

Translation of: D

```
from sys import argv

def randomGenerator(seed=1):
    max_int32 = (1 << 31) - 1
    seed = seed & max_int32

    while True:
        seed = (seed * 214013 + 2531011) & max_int32
        yield seed >> 16

def deal(seed):
    nc = 52
    cards = range(nc - 1, -1, -1)
    rnd = randomGenerator(seed)
    for i, r in zip(range(nc), rnd):
        j = (nc - 1) - r % (nc - i)
        cards[i], cards[j] = cards[j], cards[i]
    return cards

def show(cards):
    l = ["A23456789TJQK"[c / 4] + "CDHS"[c % 4] for c in cards]
    for i in range(0, len(cards), 8):
        print " ", " ".join(l[i : i+8])

if __name__ == '__main__':
    seed = int(argv[1]) if len(argv) == 2 else 11982
    print "Hand", seed
    deck = deal(seed)
    show(deck)
```

Output:

```
Hand 11982
  AH AS 4H AC 2D 6S TS JS
  3D 3H QS QC 8S 7H AD KS
  KD 6H 5S 4D 9H JH 9S 3C
  JC 5D 5C 8C 9D TD KH 7C
  6C 2C TH QH 6D TC 4S 7S
  JD 7D 8H 9C 2H QD 4C 5H
  KC 8D 2S 3S
```

Racket[edit]

Python: Death_Star[edit]

Translation of: C

```
import sys, math, collections
```

```
Sphere = collections.namedtuple("Sphere", "cx cy cz r")
V3 = collections.namedtuple("V3", "x y z")

def normalize((x, y, z)):
    len = math.sqrt(x**2 + y**2 + z**2)
    return V3(x / len, y / len, z / len)

def dot(v1, v2):
    d = v1.x*v2.x + v1.y*v2.y + v1.z*v2.z
    return -d if d < 0 else 0.0

def hit_sphere(sph, x0, y0):
    x = x0 - sph.cx
    y = y0 - sph.cy
    zsq = sph.r ** 2 - (x ** 2 + y ** 2)
    if zsq < 0:
        return (False, 0, 0)
    szsq = math.sqrt(zsq)
    return (True, sph.cz - szsq, sph.cz + szsq)

def draw_sphere(k, ambient, light):
    shades = ".:!*oe&#%@"
    pos = Sphere(20.0, 20.0, 0.0, 20.0)
    neg = Sphere(1.0, 1.0, -6.0, 20.0)

    for i in xrange(int(math.floor(pos.cy - pos.r)),
                    int(math.ceil(pos.cy + pos.r) + 1)):
        y = i + 0.5
        for j in xrange(int(math.floor(pos.cx - 2 * pos.r)),
                        int(math.ceil(pos.cx + 2 * pos.r) + 1)):
            x = (j - pos.cx) / 2.0 + 0.5 + pos.cx

            (h, zb1, zb2) = hit_sphere(pos, x, y)
            if not h:
                hit_result = 0
            else:
                (h, zs1, zs2) = hit_sphere(neg, x, y)
                if not h:
                    hit_result = 1
                elif zs1 > zb1:
                    hit_result = 1
                elif zs2 > zb2:
                    hit_result = 0
                elif zs2 > zb1:
                    hit_result = 2
                else:
                    hit_result = 1

            if hit_result == 0:
                sys.stdout.write(' ')
                continue
            elif hit_result == 1:
                vec = V3(x - pos.cx, y - pos.cy, zb1 - pos.cz)
            elif hit_result == 2:
                vec = V3(neg.cx-x, neg.cy-y, neg.cz-zs2)
```

```
            vec = normalize(vec)

            b = dot(light, vec) ** k + ambient
            intensity = int((1 - b) * len(shades))
            intensity = min(len(shades), max(0, intensity))
            sys.stdout.write(shades[intensity])
        print

light = normalize(V3(-50, 30, 50))
draw_sphere(2, 0.5, light)
```

Racket[edit]

Python: Decimal_floating_point_number_to_binary[e

Python has float.hex() and float.fromhex() that can be used to form our own

```
hex2bin = dict('{:x} {:04b}'.format(x,x).split() for x in range(16))
bin2hex = dict('{:b} {:x}'.format(x,x).split() for x in range(16))

def float_dec2bin(d):
    neg = False
    if d < 0:
        d = -d
        neg = True
    hx = float(d).hex()
    p = hx.index('p')
    bn = ''.join(hex2bin.get(char, char) for char in hx[2:p])
    return (('-' if neg else '') + bn.strip('0') + hx[p:p+2]
            + bin(int(hx[p+2:]))[2:])

def float_bin2dec(bn):
    neg = False
    if bn[0] == '-':
        bn = bn[1:]
        neg = True
    dp = bn.index('.')
    extra0 = '0' * (4 - (dp % 4))
    bn2 = extra0 + bn
    dp = bn2.index('.')
    p = bn2.index('p')
    hx = ''.join(bin2hex.get(bn2[i:min(i+4, p)].lstrip('0'), bn2[i])
                 for i in range(0, dp+1, 4))
    bn3 = bn2[dp+1:p]
    extra0 = '0' * (4 - (len(bn3) % 4))
    bn4 = bn3 + extra0
    hx += ''.join(bin2hex.get(bn4[i:i+4].lstrip('0'))
                  for i in range(0, len(bn4), 4))
    hx = (('-' if neg else '') + '0x' + hx + bn2[p:p+2]
          + str(int('0b' + bn2[p+2:], 2)))
    return float.fromhex(hx)
```

Output:

Run the above in idle then you can do the following interactively:

```
>>> x = 23.34375
>>> y = float_dec2bin(x)
>>> y
'1.011101011p+100'
>>> float_bin2dec(y)
23.34375
>>> y = float_dec2bin(-x)
>>> y
'-1.011101011p+100'
>>> float_bin2dec(y)
-23.34375
>>> float_bin2dec('1011.11101p+0')
11.90625
>>>
```

Python: Decision_tables[edit]

```
'''
Create a Decision table then use it
'''

def dt_creator():
    print("\n\nCREATING THE DECISION TABLE\n")
    conditions = input("Input conditions, in order, separated by commas: ")
    conditions = [c.strip() for c in conditions.split(',')]
    print( ("\nThat was %s conditions:\n  " % len(conditions))
           + '\n  '.join("%i: %s" % x for x in enumerate(conditions, 1)) )
    print("\nInput an action, a semicolon, then a list of tuples of rules t
    action2rules, action = [], ' '
    while action:
        action = input("%i: " % (len(action2rules) + 1)).strip()
        if action:
            name, _, rules = [x.strip() for x in action.partition(';')]
            rules = eval(rules)
            assert all(len(rule) == len(conditions) for rule in rules), \
                   "The number of conditions in a rule to trigger this acti
            action2rules.append((name, rules))
    actions = [x[0] for x in action2rules]
    # Map condition to actions
    rule2actions = dict((y,[]) for y in set(sum((x[1] for x in action2rules
    for action, rules in action2rules:
        for r in rules:
            rule2actions[r].append( action )
    return conditions, rule2actions

def dt_user(dt, default=['Pray!']):
    conditions, rule2actions = dt
    print("\n\nUSING THE DECISION TABLE\n")
```

```
    rule = tuple(int('y' == input("%s? (Answer y if statement is true or n)
    print("Try this:\n  " + '\n  '.join(rule2actions.get(rule, default)))

if __name__ == '__main__':
    dt = dt_creator()
    dt_user(dt)
    dt_user(dt)
    dt_user(dt)
```

Sample Run

```
CREATING THE DECISION TABLE

Input conditions, in order, separated by commas: Printer does not print, A

That was 3 conditions:
  1: Printer does not print
  2: A red light is flashing
  3: Printer is unrecognised

Input an action, a semicolon, then a list of tuples of rules that trigger i
1: Check the power cable; [(1,0,1)]
2: Check the printer-computer cable; [(1,1,1), (1,0,1)]
3: Ensure printer software is installed; [(1,1,1), (1,0,1), (0,1,1), (0,0,1
4: Check/replace ink;  [(1,1,1), (1,1,0), (0,1,1), (0,1,0)]
5: Check for paper jam; [(1,1,0), (1,0,0)]
6:
```

Python: Deepcopy[edit]

```
import copy
deepcopy_of_obj = copy.deepcopy(obj)
```

Racket[edit]

Python: Define_a_primitive_data_type[edit]

This doesn't really apply as Python names don't have a type, but something

Python: Delegates[edit]

```
class Delegator:
```

```
    def __init__(self):
        self.delegate = None
    def operation(self):
        if hasattr(self.delegate, 'thing') and callable(self.delegate.thing):
            return self.delegate.thing()
        return 'default implementation'

class Delegate:
    def thing(self):
        return 'delegate implementation'

if __name__ == '__main__':

    # No delegate
    a = Delegator()
    assert a.operation() == 'default implementation'

    # With a delegate that does not implement "thing"
    a.delegate = 'A delegate may be any object'
    assert a.operation() == 'default implementation'

    # With delegate that implements "thing"
    a.delegate = Delegate()
    assert a.operation() == 'delegate implementation'
```

Racket[edit]

Python: Delete_a_file[edit]

```
import os
# current directory
os.remove("output.txt")
os.rmdir("docs")
# root directory
os.remove("/output.txt")
os.rmdir("/docs")
```

Python: Deming's_Funnel[edit]

Translation of: Racket

```
import math

dxs = [-0.533, 0.27, 0.859, -0.043, -0.205, -0.127, -0.071, 0.275, 1.251,
```

```
       -0.231, -0.401, 0.269, 0.491, 0.951, 1.15, 0.001, -0.382, 0.161, 0.9
       2.08, -2.337, 0.034, -0.126, 0.014, 0.709, 0.129, -1.093, -0.483, -1
       0.02, -0.051, 0.047, -0.095, 0.695, 0.34, -0.182, 0.287, 0.213, -0.4
       -0.021, -0.134, 1.798, 0.021, -1.099, -0.361, 1.636, -1.134, 1.315, 
       0.034, 0.097, -0.17, 0.054, -0.553, -0.024, -0.181, -0.7, -0.361, -0
       0.279, -0.174, -0.009, -0.323, -0.658, 0.348, -0.528, 0.881, 0.021, 
       0.157, 0.648, 1.774, -1.043, 0.051, 0.021, 0.247, -0.31, 0.171, 0.0,
       0.024, -0.386, 0.962, 0.765, -0.125, -0.289, 0.521, 0.017, 0.281, -0
       -0.149, -2.436, -0.909, 0.394, -0.113, -0.598, 0.443, -0.521, -0.799
       0.087]

dys = [0.136, 0.717, 0.459, -0.225, 1.392, 0.385, 0.121, -0.395, 0.49, -0.6
       -0.065, 0.242, -0.288, 0.658, 0.459, 0.0, 0.426, 0.205, -0.765, -2.1
       -0.742, -0.01, 0.089, 0.208, 0.585, 0.633, -0.444, -0.351, -1.087, 0
       0.701, 0.096, -0.025, -0.868, 1.051, 0.157, 0.216, 0.162, 0.249, -0.
       0.009, 0.508, -0.79, 0.723, 0.881, -0.508, 0.393, -0.226, 0.71, 0.03
       -0.217, 0.831, 0.48, 0.407, 0.447, -0.295, 1.126, 0.38, 0.549, -0.44
       -0.046, 0.428, -0.074, 0.217, -0.822, 0.491, 1.347, -0.141, 1.23, -0
       0.079, 0.219, 0.698, 0.275, 0.056, 0.031, 0.421, 0.064, 0.721, 0.104
       -0.729, 0.65, -1.103, 0.154, -1.72, 0.051, -0.385, 0.477, 1.537, -0.
       0.939, -0.411, 0.341, -0.411, 0.106, 0.224, -0.947, -1.424, -0.542, 

def funnel(dxs, rule):
    x, rxs = 0, []
    for dx in dxs:
        rxs.append(x + dx)
        x = rule(x, dx)
    return rxs

def mean(xs): return sum(xs) / len(xs)

def stddev(xs):
    m = mean(xs)
    return math.sqrt(sum((x-m)**2 for x in xs) / len(xs))

def experiment(label, rule):
    rxs, rys = funnel(dxs, rule), funnel(dys, rule)
    print label
    print 'Mean x, y    : %.4f, %.4f' % (mean(rxs), mean(rys))
    print 'Std dev x, y : %.4f, %.4f' % (stddev(rxs), stddev(rys))
    print


experiment('Rule 1:', lambda z, dz: 0)
experiment('Rule 2:', lambda z, dz: -dz)
experiment('Rule 3:', lambda z, dz: -(z+dz))
experiment('Rule 4:', lambda z, dz: z+dz)
```

Output:

Python: Detect_division_by_zero[edit]

```
def div_check(x, y):
  try:
    x / y
  except ZeroDivisionError:
    return True
  else:
    return False
```

R[edit]

Python: Determine_if_only_one_instance_is_running

Linux (including cygwin) and Mac OSX Leopard[edit]

Works with: Python version 2.6

Must be run from an application, not the interpreter.

```
import __main__, os

def isOnlyInstance():
    # Determine if there are more than the current instance of the applicat
    # running at the current time.
    return os.system("(( $(ps -ef | grep python | grep '[" +
                     __main__.__file__[0] + "]" + __main__.__file__[1:] +
                     "' | wc -l) > 1 ))") != 0
```

This is not a solution - one can run the same app by copying the code to an

Python: Dice_game_probabilities[edit]

```
from itertools import product

def gen_dict(n_faces, n_dice):
```

```
    counts = [0] * ((n_faces + 1) * n_dice)
    for t in product(range(1, n_faces + 1), repeat=n_dice):
        counts[sum(t)] += 1
    return counts, n_faces ** n_dice

def beating_probability(n_sides1, n_dice1, n_sides2, n_dice2):
    c1, p1 = gen_dict(n_sides1, n_dice1)
    c2, p2 = gen_dict(n_sides2, n_dice2)
    p12 = float(p1 * p2)

    return sum(p[1] * q[1] / p12
               for p, q in product(enumerate(c1), enumerate(c2))
               if p[0] > q[0])

print beating_probability(4, 9, 6, 6)
print beating_probability(10, 5, 7, 6)
```

Output:

```
0.573144076783
0.642788628718
```

To handle larger number of dice (and faster in general):

Python: Dijkstra's_algorithm[edit]

Starts from the wp:Dijkstra's_algorithm#Pseudocode recognising that their f

Note: q could be changed to be a priority queue instead of a set as mentione

```
from collections import namedtuple, queue
from pprint import pprint as pp


inf = float('inf')
Edge = namedtuple('Edge', 'start, end, cost')

class Graph():
    def __init__(self, edges):
        self.edges = edges2 = [Edge(*edge) for edge in edges]
        self.vertices = set(sum(([e.start, e.end] for e in edges2), []))

    def dijkstra(self, source, dest):
```

```
        assert source in self.vertices
        dist = {vertex: inf for vertex in self.vertices}
        previous = {vertex: None for vertex in self.vertices}
        dist[source] = 0
        q = self.vertices.copy()
        neighbours = {vertex: set() for vertex in self.vertices}
        for start, end, cost in self.edges:
            neighbours[start].add((end, cost))
        #pp(neighbours)

        while q:
            u = min(q, key=lambda vertex: dist[vertex])
            q.remove(u)
            if dist[u] == inf or u == dest:
                break
            for v, cost in neighbours[u]:
                alt = dist[u] + cost
                if alt < dist[v]:                                 # Relax
                    dist[v] = alt
                    previous[v] = u
        #pp(previous)
        s, u = deque(), dest
        while previous[u]:
            s.pushleft(u)
            u = previous[u]
        s.pushleft(u)
        return s


graph = Graph([("a", "b", 7),  ("a", "c", 9),  ("a", "f", 14), ("b", "c", 1
               ("b", "d", 15), ("c", "d", 11), ("c", "f", 2),  ("d", "e", 6
               ("e", "f", 9)])
pp(graph.dijkstra("a", "e"))
```

Output:

```
['a', 'c', 'd', 'e']
```

Python: Dining_philosophers[edit]

This solution avoids deadlock by never waiting for a fork while having one
a philosopher acquires one fork but can't acquire the second, he releases t
before waiting to acquire the other (which then becomes the first fork acqu

```
import threading
```

```
import random
import time

# Dining philosophers, 5 Phillies with 5 forks. Must have two forks to eat.
#
# Deadlock is avoided by never waiting for a fork while holding a fork (loc
# Procedure is to do block while waiting to get first fork, and a nonblocki
# acquire of second fork.  If failed to get second fork, release first fork
# swap which fork is first and which is second and retry until getting both
#
# See discussion page note about 'live lock'.

class Philosopher(threading.Thread):

    running = True

    def __init__(self, xname, forkOnLeft, forkOnRight):
        threading.Thread.__init__(self)
        self.name = xname
        self.forkOnLeft = forkOnLeft
        self.forkOnRight = forkOnRight

    def run(self):
        while(self.running):
            #  Philosopher is thinking (but really is sleeping).
            time.sleep( random.uniform(3,13))
            print '%s is hungry.' % self.name
            self.dine()

    def dine(self):
        fork1, fork2 = self.forkOnLeft, self.forkOnRight

        while self.running:
            fork1.acquire(True)
            locked = fork2.acquire(False)
            if locked: break
            fork1.release()
            print '%s swaps forks' % self.name
            fork1, fork2 = fork2, fork1
        else:
            return

        self.dining()
        fork2.release()
        fork1.release()

    def dining(self):
        print '%s starts eating '% self.name
        time.sleep(random.uniform(1,10))
        print '%s finishes eating and leaves to think.' % self.name

def DiningPhilosophers():
    forks = [threading.Lock() for n in range(5)]
    philosopherNames = ('Aristotle','Kant','Buddha','Marx', 'Russel')

    philosophers= [Philosopher(philosopherNames[i], forks[i%5], forks[(i+1)%
```

```
            for i in range(5)]

    random.seed(507129)
    Philosopher.running = True
    for p in philosophers: p.start()
    time.sleep(100)
    Philosopher.running = False
    print ("Now we're finishing.")

DiningPhilosophers()
```

Racket[edit]

Python: Discordian_date[edit]

```
import datetime, calendar

DISCORDIAN_SEASONS = ["Chaos", "Discord", "Confusion", "Bureaucracy", "The A

def ddate(year, month, day):
    today = datetime.date(year, month, day)
    is_leap_year = calendar.isleap(year)
    if is_leap_year and month == 2 and day == 29:
        return "St. Tib's Day, YOLD " + (year + 1166)

    day_of_year = today.timetuple().tm_yday - 1

    if is_leap_year and day_of_year >= 60:
        day_of_year -= 1 # Compensate for St. Tib's Day

    season, dday = divmod(day_of_year, 73)
    return "%s %d, YOLD %d" % (DISCORDIAN_SEASONS[season], dday + 1, year +
```

Python: Discover_the_Hostname[edit]

Works with: Python version 2.5

Python: Display_a_linear_combination[edit]

```
def linear(x):
    return ' + '.join(['{}e({})'.format('-' if v == -1 else '' if v == 1 el
        for i, v in enumerate(x) if v] or ['0']).replace(' + -', ' - ')

list(map(lambda x: print(linear(x)), [[1, 2, 3], [0, 1, 2, 3], [1, 0, 3, 4]
        [0, 0, 0], [0], [1, 1, 1], [-1, -1, -1], [-1, -2, 0, 3], [-1]]))
```

Python: Distributed_program.1[edit]

Works with: Python version 2.4 and 2.6

XML-RPC[edit]

Protocol: XML-RPC

Server[edit]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import SimpleXMLRPCServer

class MyHandlerInstance:
    def echo(self, data):
        '''Method for returning data got from client'''
        return 'Server responded: %s' % data

    def div(self, num1, num2):
        '''Method for divide 2 numbers'''
        return num1/num2

def foo_function():
    '''A function (not an instance method)'''
    return True

HOST = "localhost"
PORT = 8000

server = SimpleXMLRPCServer.SimpleXMLRPCServer((HOST, PORT))

# register built-in system.* functions.
server.register_introspection_functions()
```

```
# register our instance
server.register_instance(MyHandlerInstance())

# register our function as well
server.register_function(foo_function)

try:
    # serve forever
    server.serve_forever()
except KeyboardInterrupt:
    print 'Exiting...'
    server.server_close()
```

Client[edit]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import xmlrpclib

HOST = "localhost"
PORT = 8000

rpc = xmlrpclib.ServerProxy("http://%s:%d" % (HOST, PORT))

# print what functions does server support
print 'Server supports these functions:',
print ' '.join(rpc.system.listMethods())

# echo something
rpc.echo("We sent this data to server")

# div numbers
print 'Server says: 8 / 4 is: %d' % rpc.div(8, 4)

# control if foo_function returns True
if rpc.foo_function():
    print 'Server says: foo_function returned True'
```

HTTP[edit]

Protocol: HTTP

Server[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import BaseHTTPServer

HOST = "localhost"
PORT = 8000

# we just want to write own class, we replace do_GET method. This could be e
# see; http://docs.python.org/lib/module-BaseHTTPServer.html
class MyHTTPHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self):
        # send 200 (OK) message
        self.send_response(200)
        # send header
        self.send_header("Content-type", "text/html")
        self.end_headers()

        # send context
        self.wfile.write("<html><head><title>Our Web Title</title></head>")
        self.wfile.write("<body><p>This is our body. You wanted to visit <b
        self.wfile.write("</html>")

if __name__ == '__main__':
    server = BaseHTTPServer.HTTPServer((HOST, PORT), MyHTTPHandler)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print 'Exiting...'
        server.server_close()
```

Client[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import httplib

HOST = "localhost"
PORT = 8000

conn = httplib.HTTPConnection(HOST, PORT)
conn.request("GET", "/somefile")

response = conn.getresponse()
print 'Server Status: %d' % response.status

print 'Server Message: %s' % response.read()
```

Python: Distributed_programming[edit]

Works with: Python version 2.4 and 2.6

XML-RPC[edit]

Protocol: XML-RPC

Server[edit]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import SimpleXMLRPCServer

class MyHandlerInstance:
    def echo(self, data):
        '''Method for returning data got from client'''
        return 'Server responded: %s' % data

    def div(self, num1, num2):
        '''Method for divide 2 numbers'''
        return num1/num2

def foo_function():
    '''A function (not an instance method)'''
    return True

HOST = "localhost"
PORT = 8000

server = SimpleXMLRPCServer.SimpleXMLRPCServer((HOST, PORT))

# register built-in system.* functions.
server.register_introspection_functions()

# register our instance
server.register_instance(MyHandlerInstance())

# register our function as well
server.register_function(foo_function)

try:
    # serve forever
    server.serve_forever()
except KeyboardInterrupt:
    print 'Exiting...'
```

```
    server.server_close()
```

Client[edit]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import xmlrpclib

HOST = "localhost"
PORT = 8000

rpc = xmlrpclib.ServerProxy("http://%s:%d" % (HOST, PORT))

# print what functions does server support
print 'Server supports these functions:',
print ' '.join(rpc.system.listMethods())

# echo something
rpc.echo("We sent this data to server")

# div numbers
print 'Server says: 8 / 4 is: %d' % rpc.div(8, 4)

# control if foo_function returns True
if rpc.foo_function():
    print 'Server says: foo_function returned True'
```

HTTP[edit]

Protocol: HTTP

Server[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import BaseHTTPServer

HOST = "localhost"
PORT = 8000

# we just want to write own class, we replace do_GET method. This could be e
# see; http://docs.python.org/lib/module-BaseHTTPServer.html
```

```
class MyHTTPHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self):
        # send 200 (OK) message
        self.send_response(200)
        # send header
        self.send_header("Content-type", "text/html")
        self.end_headers()

        # send context
        self.wfile.write("<html><head><title>Our Web Title</title></head>")
        self.wfile.write("<body><p>This is our body. You wanted to visit <b
        self.wfile.write("</html>")

if __name__ == '__main__':
    server = BaseHTTPServer.HTTPServer((HOST, PORT), MyHTTPHandler)
    try:
        server.serve_forever()
    except KeyboardInterrupt:
        print 'Exiting...'
        server.server_close()
```

Client[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import httplib

HOST = "localhost"
PORT = 8000

conn = httplib.HTTPConnection(HOST, PORT)
conn.request("GET", "/somefile")

response = conn.getresponse()
print 'Server Status: %d' % response.status

print 'Server Message: %s' % response.read()
```

Socket, Pickle format[edit]

Protocol: raw socket / pickle format

This example builds a very basic RPC mechanism on top of sockets and the pi

Server[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import SocketServer
import pickle

HOST = "localhost"
PORT = 8000

class RPCServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    # The object_to_proxy member should be set to the object we want
    # methods called on. Unfortunately, we can't do this in the constructor
    # because the constructor should not be overridden in TCPServer...

    daemon_threads = True

class RPCHandler(SocketServer.StreamRequestHandler):
    def handle(self):
        in_channel = pickle.Unpickler(self.rfile)
        out_channel = pickle.Pickler(self.wfile, protocol=2)
        while True:
            try:
                name, args, kwargs = in_channel.load()
                print 'got %s %s %s' % (name, args, kwargs)
            except EOFError:
                # EOF means we're done with this request.
                # Catching this exception to detect EOF is a bit hackish,
                # but will work for a quick demo like this
                break
            try:
                method = getattr(self.server.object_to_proxy, name)
                result = method(*args, **kwargs)
            except Exception, e:
                out_channel.dump(('Error',e))
            else:
                out_channel.dump(('OK',result))

class MyHandlerInstance(object):
    def echo(self, data):
        '''Method for returning data got from client'''
        return 'Server responded: %s' % data

    def div(self, dividend, divisor):
        '''Method to divide 2 numbers'''
        return dividend/divisor

    def is_computer_on(self):
        return True

if __name__ == '__main__':
    rpcserver = RPCServer((HOST, PORT), RPCHandler)
    rpcserver.object_to_proxy = MyHandlerInstance()
    try:
        rpcserver.serve_forever()
```

```
    except KeyboardInterrupt:
        print 'Exiting...'
        rpcserver.server_close()
```

Client[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import socket
import pickle

HOST = "localhost"
PORT = 8000

class RPCClient(object):
    def __init__(self, host, port):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((host, port))
        self.rfile = self.socket.makefile('rb')
        self.wfile = self.socket.makefile('wb')
        self.in_channel = pickle.Unpickler(self.rfile)
        self.out_channel = pickle.Pickler(self.wfile, protocol=2)

    def _close(self):
        self.socket.close()
        self.rfile.close()
        self.wfile.close()

    # Make calling remote methods easy by overriding attribute access.
    # Accessing any attribute on our instances will give a proxy method tha
    # calls the method with the same name on the remote machine.
    def __getattr__(self, name):
        def proxy(*args, **kwargs):
            self.out_channel.dump((name, args, kwargs))
            self.wfile.flush() # to make sure the server won't wait forever
            status, result = self.in_channel.load()
            if status == 'OK':
                return result
            else:
                raise result

        return proxy

if __name__ == '__main__':
    # connect to server and send data
    rpcclient = RPCClient(HOST, PORT)

    print 'Testing the echo() method:'
    print rpcclient.echo('Hello world!')
    print
```

```
    print 'Calculating 42/2 on the remote machine:'
    print rpcclient.div(42, 2)
    print
    print 'is_computer_on on the remote machine returns:'
    print rpcclient.is_computer_on()
    print
    print 'Testing keyword args:'
    print '42/2 is:', rpcclient.div(divisor=2, dividend=42)
    rpcclient._close()
    del rpcclient
```

Pyro[edit]

Note: You should install Pyro (http://pyro.sourceforge.net) first and run **p**

Server[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import Pyro.core
import Pyro.naming

# create instance that will return upper case
class StringInstance(Pyro.core.ObjBase):
    def makeUpper(self, data):
        return data.upper()

class MathInstance(Pyro.core.ObjBase):
    def div(self, num1, num2):
        return num1/num2

if __name__ == '__main__':
    server = Pyro.core.Daemon()
    name_server = Pyro.naming.NameServerLocator().getNS()
    server.useNameServer(name_server)
    server.connect(StringInstance(), 'string')
    server.connect(MathInstance(), 'math')
    try:
        server.requestLoop()
    except KeyboardInterrupt:
        print 'Exiting...'
        server.shutdown()
```

Client[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import Pyro.core

DATA = "my name is eren"
NUM1 = 10
NUM2 = 5

string = Pyro.core.getProxyForURI("PYRONAME://string")
math = Pyro.core.getProxyForURI("PYRONAME://math")

print 'We sent: %s' % DATA
print 'Server responded: %s\n' % string.makeUpper(DATA)

print 'We sent two numbers to divide: %d and %d' % (NUM1, NUM2)
print 'Server responded the result: %s' % math.div(NUM1, NUM2)
```

Spread[edit]

Note: You should install Spread (http://www.spread.org) and its python bind

Server[edit]

You don't need any code for server. You should start "spread" daemon by typ

After starting daemon, if you want to make sure that it is running, enter **s**

Client (Listener)[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import spread

PORT = '4803'

# connect spread daemon
conn = spread.connect(PORT)
# join the room
conn.join('test')
```

```
print 'Waiting for messages... If you want to stop this script, please stop
while True:
    recv = conn.receive()
    if hasattr(recv, 'sender') and hasattr(recv, 'message'):
        print 'Sender: %s' % recv.sender
        print 'Message: %s' % recv.message
```

Client (Sender)[edit]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import spread

PORT = '4803'

conn = spread.connect(PORT)
conn.join('test')

conn.multicast(spread.RELIABLE_MESS, 'test', 'hello, this is message sent f
conn.disconnect()
```

Python: Divisors_of_a_natural_number[edit]

Naive and slow but simplest (check all numbers from 1 to n):

```
>>> def factors(n):
      return [i for i in range(1, n + 1) if not n%i]
```

Slightly better (realize that there are no factors between n/2 and n):

```
>>> def factors(n):
      return [i for i in range(1, n//2 + 1) if not n%i] + [n]

>>> factors(45)
[1, 3, 5, 9, 15, 45]
```

Much better (realize that factors come in pairs, the smaller of which is no

```
>>> from math import sqrt
>>> def factor(n):
      factors = set()
      for x in range(1, int(sqrt(n)) + 1):
        if n % x == 0:
          factors.add(x)
          factors.add(n//x)
      return sorted(factors)

>>> for i in (45, 53, 64): print( "%i: factors: %s" % (i, factor(i)) )

45: factors: [1, 3, 5, 9, 15, 45]
53: factors: [1, 53]
64: factors: [1, 2, 4, 8, 16, 32, 64]
```

More efficient when factoring many numbers:

```
from itertools import chain, cycle, accumulate # last of which is Python 3 

def factors(n):
    def prime_powers(n):
        # c goes through 2, 3, 5, then the infinite (6n+1, 6n+5) series
        for c in accumulate(chain([2, 1, 2], cycle([2,4]))):
            if c*c > n: break
            if n%c: continue
            d,p = (), c
            while not n%c:
                n,p,d = n//c, p*c, d + (p,)
            yield(d)
        if n > 1: yield((n,))

    r = [1]
    for e in prime_powers(n):
        r += [a*b for a in r for b in e]
    return r
```

R[edit]

Python: Documentation[edit]

A string literal which is the first expression in a class, function, or mod

```
class Doc(object):
```

```
    """
    This is a class docstring. Traditionally triple-quoted strings are used
    they can span multiple lines and you can include quotation marks without
    """
    def method(self, num):
        """This is a method docstring."""
        pass
```

pydoc, a module of the Python standard library, can automatically generate

The built-in help() function uses the pydoc module to display docstring inf

```
>>> def somefunction():
        "takes no args and returns None after doing not a lot"


>>> help(somefunction)
Help on function somefunction in module __main__:

somefunction()
    takes no args and returns None after doing not a lot

>>>
```

Sphinx[edit]

The Sphinx Documentation generator suite is used to generate the new Python

R[edit]

Python: Dot_product[edit]

```
def dotp(a,b):
    assert len(a) == len(b), 'Vector sizes must match'
    return sum(aterm * bterm for aterm,bterm in zip(a, b))

if __name__ == '__main__':
    a, b = [1, 3, -5], [4, -2, -1]
    assert dotp(a,b) == 3
```

Python: Doubly-Linked_List[edit]

In the high level language Python, its list native datatype should be used.

Python: Doubly-Linked_List_(element)[edit]

```
class Node(object):
     def __init__(self, data = None, prev = None, next = None):
         self.prev = prev
         self.next = next
         self.data = data
     def __str__(self):
         return str(self.data)
     def __repr__(self):
         return repr(self.data)
     def iter_forward(self):
         c = self
         while c != None:
             yield c
             c = c.next
     def iter_backward(self):
         c = self
         while c != None:
             yield c
             c = c.prev
```

Racket[edit]

Python: Doubly-Linked_List_(element_insertion)[ed

```
def insert(anchor, new):
    new.next = anchor.next
    new.prev = anchor
    anchor.next.prev = new
    anchor.next = new
```

Python: Doubly-Linked_List_(traversal)[edit]

This provides two solutions. One that explicitly builds a linked list and t

Python: Draw_a_clock[edit]

Think Geek Binary Clock

Works with: Python version 2.6+, 3.0+

Textmode[edit]

```
import time

def chunks(l, n=5):
    return [l[i:i+n] for i in range(0, len(l), n)]

def binary(n, digits=8):
    n=int(n)
    return '{0:0{1}b}'.format(n, digits)

def secs(n):
    n=int(n)
    h='x' * n
    return "|".join(chunks(h))

def bin_bit(h):
    h=h.replace("1","x")
    h=h.replace("0"," ")
    return "|".join(list(h))


x=str(time.ctime()).split()
y=x[3].split(":")

s=y[-1]
y=map(binary,y[:-1])

print bin_bit(y[0])
print
print bin_bit(y[1])
print
print secs(s)
```

Library: VPython
[edit]

There is a 3D analog clock in the
VPython contributed section

Python: Draw_a_cuboid[edit]

Ascii-Art[edit]

```
def _pr(t, x, y, z):
    txt = '\n'.join(''.join(t[(n,m)] for n in range(3+x+z)).rstrip()
                    for m in reversed(range(3+y+z)))
    return txt

def cuboid(x,y,z):
    t = {(n,m):' ' for n in range(3+x+z) for m in range(3+y+z)}
    xrow = ['+'] + ['%i' % (i % 10) for i in range(x)] + ['+']
    for i,ch in enumerate(xrow):
        t[(i,0)] = t[(i,1+y)] = t[(1+z+i,2+y+z)] = ch
    if _debug: print(_pr(t, x, y, z))
    ycol = ['+'] + ['%i' % (j % 10) for j in range(y)] + ['+']
    for j,ch in enumerate(ycol):
        t[(0,j)] = t[(x+1,j)] = t[(2+x+z,1+z+j)] = ch
    zdepth = ['+'] + ['%i' % (k % 10) for k in range(z)] + ['+']
    if _debug: print(_pr(t, x, y, z))
    for k,ch in enumerate(zdepth):
        t[(k,1+y+k)] = t[(1+x+k,1+y+k)] = t[(1+x+k,k)] = ch

    return _pr(t, x, y, z)


_debug = False
if __name__ == '__main__':
    for dim in ((2,3,4), (3,4,2), (4,2,3)):
        print("CUBOID%r" % (dim,), cuboid(*dim), sep='\n')
```

Output:

```
CUBOID(2, 3, 4)
     +01+
    3  32
   2  2 1
  1  1  0
```

Python: Draw_a_rotating_cube[edit]

Library: VPython
[edit]

Works with: Python version 2.7.9

See also: Draw_a_cuboid

Short version[edit]

```
from visual import *
scene.title = "VPython: Draw a rotating cube"

scene.range = 2
scene.autocenter = True

print "Drag with right mousebutton to rotate view."
print "Drag up+down with middle mousebutton to zoom."

deg45 = math.radians(45.0)  # 0.785398163397

cube = box()    # using defaults, see http://www.vpython.org/contents/docs/
cube.rotate( angle=deg45, axis=(1,0,0) )
cube.rotate( angle=deg45, axis=(0,0,1) )

while True:                 # Animation-loop
    rate(50)
    cube.rotate( angle=0.005, axis=(0,1,0) )
```

Racket[edit]

Python: Draw_a_sphere[edit]

Ascii-Art[edit]

Translation of: C

```
import math

shades = ('.',':','!','*','o','e','&','#','%','@')

def normalize(v):
        len = math.sqrt(v[0]**2 + v[1]**2 + v[2]**2)
        return (v[0]/len, v[1]/len, v[2]/len)

def dot(x,y):
        d = x[0]*y[0] + x[1]*y[1] + x[2]*y[2]
        return -d if d < 0 else 0

def draw_sphere(r, k, ambient, light):
        for i in range(int(math.floor(-r)),int(math.ceil(r)+1)):
                x = i + 0.5
                line = ''

                for j in range(int(math.floor(-2*r)),int(math.ceil(2*r)+1))
                        y = j/2 + 0.5
                        if x*x + y*y <= r*r:
                                vec = normalize((x,y,math.sqrt(r*r - x*x - 
                                b = dot(light,vec)**k + ambient
                                intensity = int((1-b)*(len(shades)-1))
                                line += shades[intensity] if 0 <= intensity
                        else:
                                line += ' '

                print(line)

light = normalize((30,30,-50))
draw_sphere(20,4,0.1, light)
draw_sphere(10,2,0.4, light)
```

Output:

```
                                &&&&&&&&&&######
                        &&eeeeeeeeeeeeeeee&&&&&&######%%
                    &&oooo********oooooooeeeeee&&&&########%%
```

Python: Dynamic_variable_names[edit]

Works with: Python version 2.x

```
>>> name = raw_input("Enter a variable name: ")
Enter a variable name: X
>>> globals()[name] = 42
>>> X
```

```
42
```

Works with: Python version 3.x

```
>>> name = input("Enter a variable name: ")
Enter a variable name: X
>>> globals()[name] = 42
>>> X
42
```

Note: most of the time when people ask how to do this on newsgroups and oth

Python: Echo_Server[edit]

Works with: Python version 2.3 or above

```
import SocketServer

HOST = "localhost"
PORT = 12321

# this server uses ThreadingMixIn - one thread per connection
# replace with ForkMixIn to spawn a new process per connection

class EchoServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):
    # no need to override anything - default behavior is just fine
    pass

class EchoRequestHandler(SocketServer.StreamRequestHandler):
    """
    Handles one connection to the client.
    """
    def handle(self):
        print "connection from %s" % self.client_address[0]
        while True:
            line = self.rfile.readline()
            if not line: break
            print "%s wrote: %s" % (self.client_address[0], line.rstrip())
            self.wfile.write(line)
        print "%s disconnected" % self.client_address[0]


# Create the server
server = EchoServer((HOST, PORT), EchoRequestHandler)

# Activate the server; this will keep running until you
# interrupt the program with Ctrl-C
print "server listening on %s:%s" % server.server_address
```

```
server.serve_forever()
```

Python: Egyptian_fractions[edit]

```
from fractions import Fraction
from math import ceil

class Fr(Fraction):
    def __repr__(self):
        return '%s/%s' % (self.numerator, self.denominator)

def ef(fr):
    ans = []
    if fr >= 1:
        if fr.denominator == 1:
            return [[int(fr)], Fr(0, 1)]
        intfr = int(fr)
        ans, fr = [[intfr]], fr - intfr
    x, y = fr.numerator, fr.denominator
    while x != 1:
        ans.append(Fr(1, ceil(1/fr)))
        fr = Fr(-y % x, y* ceil(1/fr))
        x, y = fr.numerator, fr.denominator
    ans.append(fr)
    return ans

if __name__ == '__main__':
    for fr in [Fr(43, 48), Fr(5, 121), Fr(2014, 59)]:
        print('%r ─► %s' % (fr, ' '.join(str(x) for x in ef(fr))))
    lenmax = denommax = (0, None)
    for fr in set(Fr(a, b) for a in range(1,100) for b in range(1, 100)):
        e = ef(fr)
        #assert sum((f[0] if type(f) is list else f) for f in e) == fr, 'Wh
        elen, edenom = len(e), e[-1].denominator
        if elen > lenmax[0]:
            lenmax = (elen, fr, e)
        if edenom > denommax[0]:
            denommax = (edenom, fr, e)
    print('Term max is %r with %i terms' % (lenmax[1], lenmax[0]))
    dstr = str(denommax[0])
    print('Denominator max is %r with %i digits %s...%s' %
          (denommax[1], len(dstr), dstr[:5], dstr[-5:]))
```

Output:

```
43/48 ─► 1/2 1/3 1/16
5/121 ─► 1/25 1/757 1/763309 1/873960180913 1/1527612795642093418846225
```

```
2014/59 → [34] 1/8 1/95 1/14947 1/670223480
Term max is 97/53 with 9 terms
Denominator max is 8/97 with 150 digits 57950...89665
```

Python: Element-wise_operations[edit]

```
>>> import random
>>> from operator import add, sub, mul, floordiv
>>> from pprint import pprint as pp
>>>
>>> def ewise(matrix1, matrix2, op):
        return [[op(e1,e2) for e1,e2 in zip(row1, row2)] for row1,row2 in z

>>> m,n = 3,4   # array dimensions
>>> a0 = [[random.randint(1,9) for y in range(n)] for x in range(m)]
>>> a1 = [[random.randint(1,9) for y in range(n)] for x in range(m)]
>>> pp(a0); pp(a1)
[[7, 8, 7, 4], [4, 9, 4, 1], [2, 3, 6, 4]]
[[4, 5, 1, 6], [6, 8, 3, 4], [2, 2, 6, 3]]
>>> pp(ewise(a0, a1, add))
[[11, 13, 8, 10], [10, 17, 7, 5], [4, 5, 12, 7]]
>>> pp(ewise(a0, a1, sub))
[[3, 3, 6, -2], [-2, 1, 1, -3], [0, 1, 0, 1]]
>>> pp(ewise(a0, a1, mul))
[[28, 40, 7, 24], [24, 72, 12, 4], [4, 6, 36, 12]]
>>> pp(ewise(a0, a1, floordiv))
[[1, 1, 7, 0], [0, 1, 1, 0], [1, 1, 1, 1]]
>>> pp(ewise(a0, a1, pow))
[[2401, 32768, 7, 4096], [4096, 43046721, 64, 1], [4, 9, 46656, 64]]
>>> pp(ewise(a0, a1, lambda x, y:2*x - y))
[[10, 11, 13, 2], [2, 10, 5, -2], [2, 4, 6, 5]]
>>>
>>> def s_ewise(scalar1, matrix1, op):
        return [[op(scalar1, e1) for e1 in row1] for row1 in matrix1]

>>> scalar = 10
>>> a0
[[7, 8, 7, 4], [4, 9, 4, 1], [2, 3, 6, 4]]
>>> for op in ( add, sub, mul, floordiv, pow, lambda x, y:2*x - y ):
        print("%10s :" % op.__name__, s_ewise(scalar, a0, op))


       add : [[17, 18, 17, 14], [14, 19, 14, 11], [12, 13, 16, 14]]
       sub : [[3, 2, 3, 6], [6, 1, 6, 9], [8, 7, 4, 6]]
       mul : [[70, 80, 70, 40], [40, 90, 40, 10], [20, 30, 60, 40]]
  floordiv : [[1, 1, 1, 2], [2, 1, 2, 10], [5, 3, 1, 2]]
       pow : [[10000000, 100000000, 10000000, 10000], [10000, 1000000000, 1
  <lambda> : [[13, 12, 13, 16], [16, 11, 16, 19], [18, 17, 14, 16]]
>>>
```

Python: Elliptic_curve_arithmetic[edit]

Translation of: C

```
#!/usr/bin/env python3

class Point:
    b = 7
    def __init__(self, x=float('inf'), y=float('inf')):
        self.x = x
        self.y = y

    def copy(self):
        return Point(self.x, self.y)

    def is_zero(self):
        return self.x > 1e20 or self.x < -1e20

    def neg(self):
        return Point(self.x, -self.y)

    def dbl(self):
        if self.is_zero():
            return self.copy()
        try:
            L = (3 * self.x * self.x) / (2 * self.y)
        except ZeroDivisionError:
            return Point()
        x = L * L - 2 * self.x
        return Point(x, L * (self.x - x) - self.y)

    def add(self, q):
        if self.x == q.x and self.y == q.y:
            return self.dbl()
        if self.is_zero():
            return q.copy()
        if q.is_zero():
            return self.copy()
        try:
            L = (q.y - self.y) / (q.x - self.x)
        except ZeroDivisionError:
            return Point()
        x = L * L - self.x - q.x
        return Point(x, L * (self.x - x) - self.y)

    def mul(self, n):
        p = self.copy()
        r = Point()
        i = 1
        while i <= n:
            if i&n:
                r = r.add(p)
            p = p.dbl()
```

```
            i <<= 1
        return r

    def __str__(self):
        return "({:.3f}, {:.3f})".format(self.x, self.y)

def show(s, p):
    print(s, "Zero" if p.is_zero() else p)

def from_y(y):
    n = y * y - Point.b
    x = n**(1./3) if n>=0 else -((-n)**(1./3))
    return Point(x, y)

# demonstrate
a = from_y(1)
b = from_y(2)
show("a =", a)
show("b =", b)
c = a.add(b)
show("c = a + b =", c)
d = c.neg()
show("d = -c =", d)
show("c + d =", c.add(d))
show("a + b + d =", a.add(b.add(d)))
show("a * 12345 =", a.mul(12345))
```

Output:

```
a = (-1.817, 1.000)
b = (-1.442, 2.000)
c = a + b = (10.375, -33.525)
d = -c = (10.375, 33.525)
c + d = Zero
a + b + d = Zero
a * 12345 = (10.759, 35.387)
```

Python: Emirp_primes[edit]

This uses Prime_decomposition#Python:_Using_Croft_Spiral_sieve and so the p

There is no explicit hard-coded ceiling added to the code for the prime gen

```
from __future__ import print_function
from prime_decomposition import primes, is_prime
```

```
from heapq import *
from itertools import islice

def emirp():
    largest = set()
    emirps = []
    heapify(emirps)
    for pr in primes():
        while emirps and pr > emirps[0]:
            yield heappop(emirps)
        if pr in largest:
            yield pr
        else:
            rp = int(str(pr)[::-1])
            if rp > pr and is_prime(rp):
                heappush(emirps, pr)
                largest.add(rp)

print('First 20:\n  ', list(islice(emirp(), 20)))
print('Between 7700 and 8000:\n  [', end='')
for pr in emirp():
    if pr >= 8000: break
    if pr >= 7700: print(pr, end=', ')
print(']')
print('10000th:\n  ', list(islice(emirp(), 10000-1, 10000)))
```

Output:

```
First 20:
   [13, 17, 31, 37, 71, 73, 79, 97, 107, 113, 149, 157, 167, 179, 199, 311,
Between 7700 and 8000:
  [7717, 7757, 7817, 7841, 7867, 7879, 7901, 7927, 7949, 7951, 7963, ]
10000th:
   [948349]
```

Python: Empty_Program[edit]

An empty text file is a correct Python program that does nothing.

QUACKASM[edit]

Python: Empty_directory[edit]

Works with: Python version 2.x

```
import os;
if os.listdir(raw_input("directory")):
    print "not empty"
else:
    print "empty"
```

Python: Empty_program[edit]

An empty text file is a correct Python program that does nothing.

Python: Empty_string[edit]

The empty string is printed by Python REPL as '', and is treated as boolean

Python: Enforced_immutability[edit]

Some datatypes such as strings are immutable:

```
>>> s = "Hello"
>>> s[0] = "h"

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    s[0] = "h"
TypeError: 'str' object does not support item assignment
```

Python: Ensure_that_a_file_exists[edit]

Works with: Python version 2.5

The *os.path.exists* method will return True if a path exists False if it doe

```
import os

os.path.exists("input.txt")
os.path.exists("/input.txt")
os.path.exists("docs")
os.path.exists("/docs")
```

R[edit]

Python: Environment_variables[edit]

The os.environ dictionary maps environmental variable names to their values

```
import os
os.environ['HOME']
```

Python: Equilibrium_index[edit]

Two Pass[edit]

Uses an initial summation of the whole list then visits each item of the li

Python: Euler's_sum_of_powers_conjecture[edit]

```
def eulers_sum_of_powers():
    max_n = 250
    pow_5 = [n**5 for n in range(max_n)]
    pow5_to_n = {n**5: n for n in range(max_n)}
    for x0 in range(1, max_n):
        for x1 in range(1, x0):
```

```
            for x2 in range(1, x1):
                for x3 in range(1, x2):
                    pow_5_sum = sum(pow_5[i] for i in (x0, x1, x2, x3))
                    if pow_5_sum in pow5_to_n:
                        y = pow5_to_n[pow_5_sum]
                        return (x0, x1, x2, x3, y)

print("%i**5 + %i**5 + %i**5 + %i**5 == %i**5" % eulers_sum_of_powers())
```

Output:

```
133**5 + 110**5 + 84**5 + 27**5 == 144**5
```

The above can be written as:

Works with: Python version 2.6+

```
from itertools import combinations

def eulers_sum_of_powers():
    max_n = 250
    pow_5 = [n**5 for n in range(max_n)]
    pow5_to_n = {n**5: n for n in range(max_n)}
    for x0, x1, x2, x3 in combinations(range(1, max_n), 4):
        pow_5_sum = sum(pow_5[i] for i in (x0, x1, x2, x3))
        if pow_5_sum in pow5_to_n:
            y = pow5_to_n[pow_5_sum]
            return (x0, x1, x2, x3, y)

print("%i**5 + %i**5 + %i**5 + %i**5 == %i**5" % eulers_sum_of_powers())
```

Output:

```
27**5 + 84**5 + 110**5 + 133**5 == 144**5
```

It's much faster to cache and look up sums of two fifth powers, due to the

```
MAX = 250
p5, sum2 = {}, {}
```

```
for i in range(1, MAX):
        p5[i**5] = i
        for j in range(i, MAX):
                sum2[i**5 + j**5] = (i, j)

sk = sorted(sum2.keys())
for p in sorted(p5.keys()):
        for s in sk:
                if p <= s: break
                if p - s in sum2:
                        print(p5[p], sum2[s] + sum2[p-s])
                        exit()
```

Output:

```
144 (27, 84, 110, 133)
```

Python: Euler_method[edit]

Translation of: Common Lisp

```
def euler(f,y0,a,b,h):
        t,y = a,y0
        while t <= b:
                print "%6.3f %6.3f" % (t,y)
                t += h
                y += h * f(t,y)

def newtoncooling(time, temp):
        return -0.07 * (temp - 20)

euler(newtoncooling,100,0,100,10)
```

Output:

```
  0.000 100.000
 10.000 44.000
 20.000 27.200
 30.000 22.160
 40.000 20.648
 50.000 20.194
 60.000 20.058
```

```
70.000 20.017
80.000 20.005
90.000 20.002
```

R[edit]

Python: Eval[edit]

Works with: Python version 2.x

The exec statement allows the optional passing in of global and local names

```
>>> exec '''
x = sum([1,2,3,4])
print x
'''
10
```

Works with: Python version 3.x

Note that in Python 3.x exec is a function:

```
>>> exec('''
x = sum([1,2,3,4])
print(x)
''')
10
```

Python: Eval_in_environment[edit]

```
>>> def eval_with_x(code, a, b):
        return eval(code, {'x':b}) - eval(code, {'x':a})

>>> eval_with_x('2 ** x', 3, 5)
24
```

A slight change allows the evaluation to take multiple names:

```
>>> def eval_with_args(code, **kwordargs):
        return eval(code, kwordargs)

>>> code = '2 ** x'
>>> eval_with_args(code, x=5) - eval_with_args(code, x=3)
24
>>> code = '3 * x + y'
>>> eval_with_args(code, x=5, y=2) - eval_with_args(code, x=3, y=1)
7
```

R[edit]

Python: Evaluate_binomial_coefficients[edit]

Straight-forward implementation:

```
def binomialCoeff(n, k):
    result = 1
    for i in range(1, k+1):
        result = result * (n-i+1) / i
    return result

if __name__ == "__main__":
    print(binomialCoeff(5, 3))
```

Python: Evolutionary_algorithm[edit]

Using lists instead of strings for easier manipulation, and a mutation rate

```
from string import letters
from random import choice, random

target  = list("METHINKS IT IS LIKE A WEASEL")
charset = letters + ' '
parent  = [choice(charset) for _ in range(len(target))]
minmutaterate  = .09
C = range(100)
```

```
perfectfitness = float(len(target))

def fitness(trial):
    'Sum of matching chars by position'
    return sum(t==h for t,h in zip(trial, target))

def mutaterate():
    'Less mutation the closer the fit of the parent'
    return 1-((perfectfitness - fitness(parent)) / perfectfitness * (1 - mi

def mutate(parent, rate):
    return [(ch if random() <= rate else choice(charset)) for ch in parent]

def que():
    '(from the favourite saying of Manuel in Fawlty Towers)'
    print ("#%-4i, fitness: %4.1f%%, '%s'" %
           (iterations, fitness(parent)*100./perfectfitness, ''.join(parent

def mate(a, b):
    place = 0
    if choice(xrange(10)) < 7:
        place = choice(xrange(len(target)))
    else:
        return a, b

    return a, b, a[:place] + b[place:], b[:place] + a[place:]

iterations = 0
center = len(C)/2
while parent != target:
    rate = mutaterate()
    iterations += 1
    if iterations % 100 == 0: que()
    copies = [ mutate(parent, rate) for _ in C ]  + [parent]
    parent1 = max(copies[:center], key=fitness)
    parent2 = max(copies[center:], key=fitness)
    parent = max(mate(parent1, parent2), key=fitness)
que()
```

Sample output

```
#100 , fitness: 50.0%, 'DVTAIKKS OZ IAPYIKWXALWE CEL'
#200 , fitness: 60.7%, 'MHUBINKMEIG IS LIZEVA WEOPOL'
#300 , fitness: 71.4%, 'MEYHINKS ID SS LIJF A KEKUEL'

#378 , fitness: 100.0%, 'METHINKS IT IS LIKE A WEASEL'
```

A simpler Python version that converges in less steps:

```
from random import choice, random

target  = list("METHINKS IT IS LIKE A WEASEL")
alphabet = " ABCDEFGHIJLKLMNOPQRSTUVWXYZ"
p = 0.05 # mutation probability
c = 100  # number of children in each generation

def neg_fitness(trial):
    return sum(t != h for t,h in zip(trial, target))

def mutate(parent):
    return [(choice(alphabet) if random() < p else ch) for ch in parent]

parent = [choice(alphabet) for _ in xrange(len(target))]
i = 0
print "%3d" % i, "".join(parent)
while parent != target:
    copies = (mutate(parent) for _ in xrange(c))
    parent = min(copies, key=neg_fitness)
    print "%3d" % i, "".join(parent)
    i += 1
```

Python: Exceptions_Through_Nested_Calls[edit]

There is no extra syntax to add to functions and/or methods such as *bar*, to say what exceptions they may raise or pass through them:

```
class U0(Exception): pass
class U1(Exception): pass

def foo():
    for i in range(2):
        try:
            bar(i)
        except U0:
            print("Function foo caught exception U0")

def bar(i):
    baz(i) # Nest those calls

def baz(i):
    raise U1 if i else U0

foo()
```

Output:

```
Function foo caught exception U0

Traceback (most recent call last):
  File "C:/Paddy3118/Exceptions_Through_Nested_Calls.py", line 17, in <modu
    foo()
  File "C:/Paddy3118/Exceptions_Through_Nested_Calls.py", line 7, in foo
```

Python: Executable_library[edit]

Executable libraries are common in Python. The Python entry for Hailstone s

The entry is copied below and, for this task needs to be in a file called h

```
def hailstone(n):
    seq = [n]
    while n>1:
        n = 3*n + 1 if n & 1 else n//2
        seq.append(n)
    return seq

if __name__ == '__main__':
    h = hailstone(27)
    assert len(h)==112 and h[:4]==[27, 82, 41, 124] and h[-4:]==[8, 4, 2, 1
    print("Maximum length %i was found for hailstone(%i) for numbers <100,0
          max((len(hailstone(i)), i) for i in range(1,100000)))
```

In the case of the Python language the interpreter maintains a module level

If the same file hailstone.py is *run*, (as maybe python hailstone.py; or may

Library importing executable

The second executable is the file common_hailstone_length.py with this conte

```
from collections import Counter

def function_length_frequency(func, hrange):
    return Counter(len(func(n)) for n in hrange).most_common()

if __name__ == '__main__':
    from executable_hailstone_library import hailstone

    upto = 100000
    hlen, freq = function_length_frequency(hailstone, range(1, upto))[0]
    print("The length of hailstone sequence that is most common for\n"
          "hailstone(n) where 1<=n<%i, is %i. It occurs %i times."
```

```
        % (upto, hlen, freq))
```

Both files could be in the same directory. (That is the easiest way to make

Output:

On executing the file common_hailstone_length.py it loads the library and p

```
The length of hailstone sequence that is most common for
hailstone(n) where 1<=n<100000, is 72. It occurs 1467 times
```

Note that the file common_hailstone_length.py is itself written as an execu

Other examples[edit]

- The Python Prime decomposition entry of Least common multiple employs
- Names_to_numbers#Python uses Number_names#Python as an executable lib

Racket[edit]

Python: Execute_a_Markov_algorithm[edit]

The example uses a regexp to parse the syntax of the grammar. This regexp i

The example gains flexibility by not being tied to specific files. The func

```
import re

def extractreplacements(grammar):
    return [ (matchobj.group('pat'), matchobj.group('repl'), bool(matchobj.
                for matchobj in re.finditer(syntaxre, grammar)
                if matchobj.group('rule')]
```

```
def replace(text, replacements):
    while True:
        for pat, repl, term in replacements:
            if pat in text:
                text = text.replace(pat, repl, 1)
                if term:
                    return text
                break
        else:
            return text

syntaxre = r"""(?mx)
^(?:
  (?: (?P<comment> \# .* ) ) |
  (?: (?P<blank>   \s*  ) (?: \n | $ )  ) |
  (?: (?P<rule>    (?P<pat> .+? ) \s+ -> \s+ (?P<term> \.)? (?P<repl> .+) ) )
)$
"""

grammar1 = """\
# This rules file is extracted from Wikipedia:
# http://en.wikipedia.org/wiki/Markov_Algorithm
A -> apple
B -> bag
S -> shop
T -> the
the shop -> my brother
a never used -> .terminating rule
"""

grammar2 = '''\
# Slightly modified from the rules on Wikipedia
A -> apple
B -> bag
S -> .shop
T -> the
the shop -> my brother
a never used -> .terminating rule
'''

grammar3 = '''\
# BNF Syntax testing rules
A -> apple
WWWW -> with
Bgage -> ->.*
B -> bag
->.* -> money
W -> WW
S -> .shop
T -> the
the shop -> my brother
a never used -> .terminating rule
'''

grammar4 = '''\
```

```
### Unary Multiplication Engine, for testing Markov Algorithm implementatio
### By Donal Fellows.
# Unary addition engine
_+1 -> _1+
1+1 -> 11+
# Pass for converting from the splitting of multiplication into ordinary
# addition
1! -> !1
,! -> !+
_! -> _
# Unary multiplication by duplicating left side, right side times
1*1 -> x,@y
1x -> xX
X, -> 1,1
X1 -> 1X
_x -> _X
,x -> ,X
y1 -> 1y
y_ -> _
# Next phase of applying
1@1 -> x,@y
1@_ -> @_
,@_ -> !_
++ -> +
# Termination cleanup for addition
_1 -> 1
1+_ -> 1
_+_ ->
'''


grammar5 = '''\
# Turing machine: three-state busy beaver
#
# state A, symbol 0 => write 1, move right, new state B
A0 -> 1B
# state A, symbol 1 => write 1, move left, new state C
0A1 -> C01
1A1 -> C11
# state B, symbol 0 => write 1, move left, new state A
0B0 -> A01
1B0 -> A11
# state B, symbol 1 => write 1, move right, new state B
B1 -> 1B
# state C, symbol 0 => write 1, move left, new state B
0C0 -> B01
1C0 -> B11
# state C, symbol 1 => write 1, move left, halt
0C1 -> H01
1C1 -> H11
'''

text1 = "I bought a B of As from T S."

text2 = "I bought a B of As W my Bgage from T S."

text3 = '_1111*11111_'
```

```
text4 = '000000A000000'


if __name__ == '__main__':
    assert replace(text1, extractreplacements(grammar1)) \
           == 'I bought a bag of apples from my brother.'
    assert replace(text1, extractreplacements(grammar2)) \
           == 'I bought a bag of apples from T shop.'
    # Stretch goals
    assert replace(text2, extractreplacements(grammar3)) \
           == 'I bought a bag of apples with my money from T shop.'
    assert replace(text3, extractreplacements(grammar4)) \
           == '11111111111111111111'
    assert replace(text4, extractreplacements(grammar5)) \
           == '00011H1111000'
```

Racket[edit]

Python: Execute_a_System_Command[edit]

```
import os
exit_code = os.system('ls')       # Just execute the command, return a succ
output    = os.popen('ls').read() # If you want to get the output data. Dep
```

or

Works with: Python version 2.7 (and above)

Python: Execute_a_system_command[edit]

```
import os
exit_code = os.system('ls')       # Just execute the command, return a succ
output    = os.popen('ls').read() # If you want to get the output data. Dep
```

or

Works with: Python version 2.7 (and above)

```
import subprocess
# if the exit code was non-zero these commands raise a CalledProcessError
exit_code = subprocess.check_call(['ls', '-l'])   # Python 2.5+
assert exit_code == 0
output    = subprocess.check_output(['ls', '-l']) # Python 2.7+
```

or

Works with: Python version 2.4 (and above)

```
from subprocess import PIPE, Popen, STDOUT
p = Popen('ls', stdout=PIPE, stderr=STDOUT)
print p.communicate()[0]
```

Note: The latter is the preferred method for calling external processes, al

or

Works with: Python version 2.2 (and above)

```
import commands
stat, out = commands.getstatusoutput('ls')
if not stat:
    print out
```

Python: Exponentiation_operator[edit]

```
MULTIPLY = lambda x, y: x*y

class num(float):
    # the following method has complexity O(b)
    # rather than O(log b) via the rapid exponentiation
    def __pow__(self, b):
        return reduce(MULTIPLY, [self]*b, 1)

# works with ints as function or operator
print num(2).__pow__(3)
print num(2) ** 3

# works with floats as function or operator
print num(2.3).__pow__(8)
print num(2.3) ** 8
```

[R](#)[[edit](#)]

[Python: Exponentiation_order](#)[[edit](#)]

```
>>> 5**3**2
1953125
>>> (5**3)**2
15625
>>> 5**(3**2)
1953125
>>> # The following is not normally done
>>> try: from functools import reduce # Py3K
except: pass

>>> reduce(pow, (5, 3, 2))
15625
>>>
```

[Python: Extend_your_language](#)[[edit](#)]

Macro programming is heavily discouraged in the Python community. One of th

[Python: Extensible_prime_generator](#)[[edit](#)]

The Croft spiral sieve prime generator from the [Prime decomposition](#) task is

```
islice(count(7), 0, None, 2)
```

The call to count(7) is to a generator of integers that counts from 7 upwar

The definition croft is a generator of primes and is used to generate as ma

[Python: Extract_file_extension](#)[[edit](#)]

Uses [os.path.splitext](#) and the extended tests from the Go example above.

```
Python 3.5.0a1 (v3.5.0a1:5d4b6a57d5fd, Feb  7 2015, 17:58:38) [MSC v.1900 3
Type "copyright", "credits" or "license()" for more information.
```

```
>>> import os
>>> tests = ["picture.jpg",
                "http://mywebsite.com/picture/image.png",
                "myuniquefile.longextension",
                "IAmAFileWithoutExtension",
                "/path/to.my/file",
                "file.odd_one",
                # Extra, with unicode
                "café.png",
                "file.resumé",
                # with unicode combining characters
                "cafe\u0301.png",
                "file.resume\u0301"]
>>> for path in tests:
    print("Path: %r -> Extension: %r" % (path, os.path.splitext(path)[-1]))


Path: 'picture.jpg' -> Extension: '.jpg'
Path: 'http://mywebsite.com/picture/image.png' -> Extension: '.png'
Path: 'myuniquefile.longextension' -> Extension: '.longextension'
Path: 'IAmAFileWithoutExtension' -> Extension: ''
Path: '/path/to.my/file' -> Extension: ''
Path: 'file.odd_one' -> Extension: '.odd_one'
Path: 'café.png' -> Extension: '.png'
Path: 'file.resumé' -> Extension: '.resumé'
Path: 'café.png' -> Extension: '.png'
Path: 'file.resumé' -> Extension: '.resumé'
>>>
```

Racket[edit]

Python: Extreme_floating_point_values[edit]

```
>>> # Extreme values from expressions
>>> inf = 1e234 * 1e234
>>> _inf = 1e234 * -1e234
>>> _zero = 1 / _inf
>>> nan = inf + _inf
>>> inf, _inf, _zero, nan
(inf, -inf, -0.0, nan)
>>> # Print
>>> for value in (inf, _inf, _zero, nan): print (value)

inf
-inf
-0.0
nan
>>> # Extreme values from other means
```

```
>>> float('nan')
nan
>>> float('inf')
inf
>>> float('-inf')
-inf
>>> -0.
-0.0
>>> # Some arithmetic
>>> nan == nan
False
>>> nan is nan
True
>>> 0. == -0.
True
>>> 0. is -0.
False
>>> inf + _inf
nan
>>> 0.0 * nan
nan
>>> nan * 0.0
nan
>>> 0.0 * inf
nan
>>> inf * 0.0
nan
```

```
>>> # But note!
>>> 1 / -0.0

Traceback (most recent call last):
  File "<pyshell#106>", line 1, in <module>
    1 / -0.0
ZeroDivisionError: float division by zero
>>> # (Not minus infinity)
```

R[edit]

Python: FASTA_format[edit]

```
I use a string to mimic an input file.
If it was an input file, then the file is read line-by-line
and I use a generator expression yielding key, value pairs
as soon as they are read, keeping the minimum in memory.
```

```
import io

FASTA='''\
>Rosetta_Example_1
THERECANBENOSPACE
>Rosetta_Example_2
THERECANBESEVERAL
LINESBUTTHEYALLMUST
BECONCATENATED'''

infile = io.StringIO(FASTA)

def fasta_parse(infile):
    key = ''
    for line in infile:
        if line.startswith('>'):
            if key:
                yield key, val
            key, val = line[1:].rstrip().split()[0], ''
        elif key:
            val += line.rstrip()
    if key:
        yield key, val

print('\n'.join('%s: %s' % keyval for keyval in fasta_parse(infile)))
```

Output:

```
Rosetta_Example_1: THERECANBENOSPACE
Rosetta_Example_2: THERECANBESEVERALLINESBUTTHEYALLMUSTBECONCATENATED
```

Python: FIFO[edit]

A python list can be used as a simple FIFO by simply using only it's *.appen*

To encapsulate this behavior into a class and provide the task's specific A

```
class FIFO(object):
    def __init__(self, *args):
        self.contents = list(args)
    def __call__(self):
        return self.pop()
    def __len__(self):
        return len(self.contents)
    def pop(self):
```

```
            return self.contents.pop(0)
        def push(self, item):
            self.contents.append(item)
        def extend(self,*itemlist):
            self.contents += itemlist
        def empty(self):
            return bool(self.contents)
        def __iter__(self):
            return self
        def next(self):
            if self.empty():
                raise StopIteration
            return self.pop()

if __name__ == "__main__":
    # Sample usage:
    f = FIFO()
    f.push(3)
    f.push(2)
    f.push(1)
    while not f.empty():
        print f.pop(),
    # >>> 3 2 1
    # Another simple example gives the same results:
    f = FIFO(3,2,1)
    while not f.empty():
        print f(),
    # Another using the default "truth" value of the object
    # (implicitly calls on the length() of the object after
    # checking for a __nonzero__ method
    f = FIFO(3,2,1)
    while f:
        print f(),
    # Yet another, using more Pythonic iteration:
    f = FIFO(3,2,1)
    for i in f:
        print i,
```

This example does add to a couple of features which are easy in Python and a

These additional methods could be omitted and some could have been dispatche

That sort of wrapper looks like:

```
class FIFO:  ## NOT a new-style class, must not derive from "object"
   def __init__(self,*args):
       self.contents = list(args)
   def __call__(self):
       return self.pop()
   def empty(self):
       return bool(self.contents)
   def pop(self):
```

```
        return self.contents.pop(0)
    def __getattr__(self, attr):
        return getattr(self.contents,attr)
    def next(self):
        if not self:
            raise StopIteration
        return self.pop()
```

As noted in the contents this must NOT be a new-style class, it must NOT bu

Works with: Python version 2.4+

Python 2.4 and later includes a deque class, supporting thread-safe, memory

```
from collections import deque
fifo = deque()
fifo. appendleft(value) # push
value = fifo.pop()
not fifo # empty
fifo.pop() # raises IndexError when empty
```

Python: FIFO_(usage)[edit]

```
import Queue
my_queue = Queue.Queue()
my_queue.put("foo")
my_queue.put("bar")
my_queue.put("baz")
print my_queue.get()  # foo
print my_queue.get()  # bar
print my_queue.get()  # baz
```

Racket[edit]

Python: FTP[edit]

Works with: Python version 2.7.10

Python: Factorial_function[edit]

Library[edit]

Works with: Python version 2.6+, 3.x

Python: Factors_of_a_Mersenne_number[edit]

```
def is_prime(number):
    return True # code omitted - see Primality by Trial Division

def m_factor(p):
    max_k = 16384 / p # arbitrary limit; since Python automatically uses lo
    for k in xrange(max_k):
        q = 2*p*k + 1
        if not is_prime(q):
            continue
        elif q % 8 != 1 and q % 8 != 7:
            continue
        elif pow(2, p, q) == 1:
            return q
    return None

if __name__ == '__main__':
    exponent = int(raw_input("Enter exponent of Mersenne number: "))
    if not is_prime(exponent):
        print "Exponent is not prime: %d" % exponent
    else:
        factor = m_factor(exponent)
        if not factor:
            print "No factor found for M%d" % exponent
        else:
            print "M%d has a factor: %d" % (exponent, factor)
```

Example:

```
Enter exponent of Mersenne number: 929
M929 has a factor: 13007
```

Racket[edit]

Python: Factors_of_an_integer[edit]

Naive and slow but simplest (check all numbers from 1 to n):

```
>>> def factors(n):
      return [i for i in range(1, n + 1) if not n%i]
```

Slightly better (realize that there are no factors between n/2 and n):

```
>>> def factors(n):
      return [i for i in range(1, n//2 + 1) if not n%i] + [n]

>>> factors(45)
[1, 3, 5, 9, 15, 45]
```

Much better (realize that factors come in pairs, the smaller of which is no

Python: Farey_sequence[edit]

```
from fractions import Fraction


class Fr(Fraction):
    def __repr__(self):
        return '(%s/%s)' % (self.numerator, self.denominator)


def farey(n, length=False):
    if not length:
        return [Fr(0, 1)] + sorted({Fr(m, k) for k in range(1, n+1) for m i
    else:
        #return 1         +    len({Fr(m, k) for k in range(1, n+1) for m i
        return  (n*(n+3))//2 - sum(farey(n//k, True) for k in range(2, n+1)

if __name__ == '__main__':
    print('Farey sequence for order 1 through 11 (inclusive):')
    for n in range(1, 12):
        print(farey(n))
    print('Number of fractions in the Farey sequence for order 100 through
    print([farey(i, length=True) for i in range(100, 1001, 100)])
```

Output:

```
Farey sequence for order 1 through 11 (inclusive):
[(0/1), (1/1)]
[(0/1), (1/2), (1/1)]
[(0/1), (1/3), (1/2), (2/3), (1/1)]
[(0/1), (1/4), (1/3), (1/2), (2/3), (3/4), (1/1)]
[(0/1), (1/5), (1/4), (1/3), (2/5), (1/2), (3/5), (2/3), (3/4), (4/5), (1/1
[(0/1), (1/6), (1/5), (1/4), (1/3), (2/5), (1/2), (3/5), (2/3), (3/4), (4/5
[(0/1), (1/7), (1/6), (1/5), (1/4), (2/7), (1/3), (2/5), (3/7), (1/2), (4/7
[(0/1), (1/8), (1/7), (1/6), (1/5), (1/4), (2/7), (1/3), (3/8), (2/5), (3/7
[(0/1), (1/9), (1/8), (1/7), (1/6), (1/5), (2/9), (1/4), (2/7), (1/3), (3/8
[(0/1), (1/10), (1/9), (1/8), (1/7), (1/6), (1/5), (2/9), (1/4), (2/7), (3/
[(0/1), (1/11), (1/10), (1/9), (1/8), (1/7), (1/6), (2/11), (1/5), (2/9), (
Number of fractions in the Farey sequence for order 100 through 1,000 (incl
[3045, 12233, 27399, 48679, 76117, 109501, 149019, 194751, 246327, 304193]
```

Python: Fibonacci_sequence[edit]

Analytic[edit]

```
from math import *

def analytic_fibonacci(n):
  sqrt_5 = sqrt(5);
  p = (1 + sqrt_5) / 2;
  q = 1/p;
  return int( (p**n + q**n) / sqrt_5 + 0.5 )

for i in range(1,31):
  print analytic_fibonacci(i),
```

Output:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17
```

Iterative[edit]

```
def fibIter(n):
    if n < 2:
        return n
    fibPrev = 1
    fib = 1
    for num in xrange(2, n):
        fibPrev, fib = fib, fib + fibPrev
    return fib
```

Recursive[edit]

```
def fibRec(n):
    if n < 2:
        return n
    else:
        return fibRec(n-1) + fibRec(n-2)
```

Recursive with Memoization[edit]

```
def fibMemo():
    pad = {0:0, 1:1}
    def func(n):
        if n not in pad:
            pad[n] = func(n-1) + func(n-2)
        return pad[n]
    return func

fm = fibMemo()
for i in range(1,31):
    print fm(i),
```

Output:

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17
```

Better Recursive doesn't need Memoization[edit]

The recursive code as written two sections above is incredibly slow and ine

```
def fibFastRec(n):
    def fib(prvprv, prv, c):
        if c < 1: return prvprv
        else: return fib(prv, prvprv + prv, c - 1)
    return fib(0, 1, n)
```

However, although much faster and not requiring memory, the above code can

Generative[edit]

```
def fibGen(n,a=0,b=1):
    while n>0:
        yield a
        a,b,n = b,a+b,n-1
```

Example use[edit]

```
>>> [i for i in fibGen(11)]

[0,1,1,2,3,5,8,13,21,34,55]
```

Matrix-Based[edit]

Translation of the matrix-based approach used in F#.

```
def prevPowTwo(n):
    'Gets the power of two that is less than or equal to the given input'
    if ((n & -n) == n):
        return n
    else:
        n -= 1
        n |= n >> 1
        n |= n >> 2
        n |= n >> 4
        n |= n >> 8
        n |= n >> 16
        n += 1
```

```
        return (n/2)

def crazyFib(n):
    'Crazy fast fibonacci number calculation'
    powTwo = prevPowTwo(n)

    q = r = i = 1
    s = 0

    while(i < powTwo):
        i *= 2
        q, r, s = q*q + r*r, r * (q + s), (r*r + s*s)

    while(i < n):
        i += 1
        q, r, s = q+r, q, r

    return q
```

Large step recurrence[edit]

This is much faster for a single, large value of n:

```
def fib(n, c={0:1, 1:1}):
    if n not in c:
        x = n // 2
        c[n] = fib(x-1) * fib(n-x-1) + fib(x) * fib(n - x)
    return c[n]

fib(10000000)  # calculating it takes a few seconds, printing it takes eons
```

Generative with Recursion[edit]

This can get very slow and uses a lot of memory. Can be sped up by caching

```
def fib():
    """Yield fib[n+1] + fib[n]"""
    yield 1  # have to start somewhere
    lhs, rhs = fib(), fib()
    yield next(lhs) # move lhs one iteration ahead
    while True:
        yield next(lhs)+next(rhs)
```

```
f=fib()
print [next(f) for _ in range(9)]
```

Output:

```
[1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Python: File_IO[edit]

The following use of the standard libraries shutil.copyfile is to be prefer

```
import shutil
shutil.copyfile('input.txt', 'output.txt')
```

However the following example shows how one would do file I/O of other sort

```
infile = open('input.txt', 'r')
outfile = open('output.txt', 'w')
for line in infile:
   outfile.write(line)
outfile.close()
infile.close()
```

This does no error checking. A more robust program would wrap each open wit

```
import sys
try:
    infile = open('input.txt', 'r')
except IOError:
    print >> sys.stderr, "Unable to open input.txt for input"
    sys.exit(1)
try:
    outfile = open('output.txt', 'w')
except IOError:
    print >> sys.stderr, "Unable to open output.txt for output"
    sys.exit(1)
try:  # for finally
    try: # for I/O
        for line in infile:
```

```
            outfile.write(line)
    except IOError, e:
        print >> sys.stderr, "Some I/O Error occurred (reading from input.t
finally:
    infile.close()
    outfile.close()
```

In Python 2.6 (or 2.5 if we use *from __future__ import with_statement*) we c

```
import sys
try:
    with open('input.txt') as infile:
        with open('output.txt', 'w') as outfile:
            for line in infile:
                outfile.write(line)
except IOError:
    print >> sys.stderr, "Some I/O Error occurred"
    sys.exit(1)
```

The files will automatically be closed on exit of their *with:* blocks. (Thus

Python: File_Size[edit]

```
import os

size = os.path.getsize('input.txt')
size = os.path.getsize('/input.txt')
```

R[edit]

Python: File_extension_is_in_extensions_list[edit

```
import os

def isExt(filename, extensions):
    return os.path.splitext(filename.lower())[-1] in [e.lower() for e in ex
```

Python: File_modification_time[edit]

```
import os

#Get modification time:
modtime = os.path.getmtime('filename')

#Set the access and modification times:
os.utime('path', (actime, mtime))

#Set just the modification time:
os.utime('path', (os.path.getatime('path'), mtime))

#Set the access and modification times to the current time:
os.utime('path', None)
```

Python: File_size[edit]

```
import os

size = os.path.getsize('input.txt')
size = os.path.getsize('/input.txt')
```

Python: Filter[edit]

Works with: Python version 2.4

```
values = range(10)
evens = [x for x in values if not x & 1]
ievens = (x for x in values if not x & 1) # lazy
# alternately but less idiomatic:
evens = filter(lambda x: not x & 1, values)
```

Alternative using the slice syntax with its optional "stride" expression:

```
values = range(10)
evens = values[::2]
```

This works for all versions of Python (at least as far back as 1.5). Lists

Since strings in Python can be treated as a sort of immutable list of chara

One can also assign to a slice (of a list or other mutable indexed object.

```
values = range(10)
values[::2] = [11,13,15,17,19]
print values
11, 1, 13, 3, 15, 5, 17, 7, 19, 9
```

Python: Find_Common_Directory_Path[edit]

The Python os.path.commonprefix function is broken as it returns common cha

```
>>> import os
>>> os.path.commonprefix(['/home/user1/tmp/coverage/test',
                          '/home/user1/tmp/covert/operator', '/home/user1/t
'/home/user1/tmp/cove'
```

This result can be fixed:

```
>>> def commonprefix(args, sep='/'):
        return os.path.commonprefix(args).rpartition(sep)[0]

>>> commonprefix(['/home/user1/tmp/coverage/test',
                  '/home/user1/tmp/covert/operator', '/home/user1/tmp/coven
'/home/user1/tmp'
```

But it may be better to not rely on the faulty implementation at all:

```
>>> from itertools import takewhile
>>> def allnamesequal(name):
        return all(n==name[0] for n in name[1:])

>>> def commonprefix(paths, sep='/'):
        bydirectorylevels = zip(*[p.split(sep) for p in paths])
        return sep.join(x[0] for x in takewhile(allnamesequal, bydirectoryl

>>> commonprefix(['/home/user1/tmp/coverage/test',
                  '/home/user1/tmp/covert/operator', '/home/user1/tmp/coven
```

```
'/home/user1/tmp'
>>> # And also
>>> commonprefix(['/home/user1/tmp', '/home/user1/tmp/coverage/test',
                  '/home/user1/tmp/covert/operator', '/home/user1/tmp/coven
'/home/user1/tmp'
>>>
```

R[edit]

Python: Find_common_directory_path[edit]

The Python os.path.commonprefix function is broken as it returns common cha

```
>>> import os
>>> os.path.commonprefix(['/home/user1/tmp/coverage/test',
                          '/home/user1/tmp/covert/operator', '/home/user1/tr
'/home/user1/tmp/cove'
```

This result can be fixed:

```
>>> def commonprefix(args, sep='/'):
        return os.path.commonprefix(args).rpartition(sep)[0]

>>> commonprefix(['/home/user1/tmp/coverage/test',
                  '/home/user1/tmp/covert/operator', '/home/user1/tmp/coven
'/home/user1/tmp'
```

But it may be better to not rely on the faulty implementation at all:

```
>>> from itertools import takewhile
>>> def allnamesequal(name):
        return all(n==name[0] for n in name[1:])

>>> def commonprefix(paths, sep='/'):
        bydirectorylevels = zip(*[p.split(sep) for p in paths])
        return sep.join(x[0] for x in takewhile(allnamesequal, bydirectoryle

>>> commonprefix(['/home/user1/tmp/coverage/test',
                  '/home/user1/tmp/covert/operator', '/home/user1/tmp/coven
'/home/user1/tmp'
```

```
>>> # And also
>>> commonprefix(['/home/user1/tmp', '/home/user1/tmp/coverage/test',
                  '/home/user1/tmp/covert/operator', '/home/user1/tmp/coven
'/home/user1/tmp'
>>>
```

Python: Find_first_and_last_set_bit_of_a_long_int

Works with: Python version 2.7+ and 3.1+

```
def msb(x):
    return x.bit_length() - 1

def lsb(x):
    return msb(x & -x)

for i in range(6):
    x = 42 ** i
    print("%10d MSB: %2d LSB: %2d" % (x, msb(x), lsb(x)))

for i in range(6):
    x = 1302 ** i
    print("%20d MSB: %2d LSB: %2d" % (x, msb(x), lsb(x)))
```

Output:

```
         1 MSB:  0 LSB:  0
        42 MSB:  5 LSB:  1
      1764 MSB: 10 LSB:  2
     74088 MSB: 16 LSB:  3
   3111696 MSB: 21 LSB:  4
 130691232 MSB: 26 LSB:  5
                   1 MSB:  0 LSB:  0
                1302 MSB: 10 LSB:  1
             1695204 MSB: 20 LSB:  2
          2207155608 MSB: 31 LSB:  3
       2873716601616 MSB: 41 LSB:  4
    3741579015304032 MSB: 51 LSB:  5
```

Python: Find_largest_left_truncatable_prime_in_a_

```
import random
```

```
def is_probable_prime(n,k):
    #this uses the miller-rabin primality test found from rosetta code
    if n==0 or n==1:
        return False
    if n==2:
        return True
    if n % 2 == 0:
        return False
    s = 0
    d = n-1

    while True:
        quotient, remainder = divmod(d, 2)
        if remainder == 1:
            break
        s += 1
        d = quotient

    def try_composite(a):
        if pow(a, d, n) == 1:
            return False
        for i in range(s):
            if pow(a, 2**i * d, n) == n-1:
                return False
        return True # n is definitely composite

    for i in range(k):
        a = random.randrange(2, n)
        if try_composite(a):
            return False

    return True # no base tested showed n as composite


def largest_left_truncatable_prime(base):
    radix = 0
    candidates = [0]
    while True:
        new_candidates=[]
        multiplier = base**radix
        for i in range(1,base):
            new_candidates += [x+i*multiplier for x in candidates if is_pro
        if len(new_candidates)==0:
            return max(candidates)
        candidates = new_candidates
        radix += 1

for b in range(3,24):
    print("%d:%d\n" % (b,largest_left_truncatable_prime(b)))
```

Output:

3:23

4:4091

5:7817

6:4836525320399

7:817337

8:14005650767869

9:1676456897

10:357686312646216567629137

Python: Find_last_sunday_of_each_month[edit]

```
#!/usr/bin/python3

'''
    Output:
    2013-Jan-27
    2013-Feb-24
    2013-Mar-31
    2013-Apr-28
    2013-May-26
    2013-Jun-30
    2013-Jul-28
    2013-Aug-25
    2013-Sep-29
    2013-Oct-27
    2013-Nov-24
    2013-Dec-29
'''

import sys
import calendar

YEAR = sys.argv[-1]
try:
    year = int(YEAR)
except:
    year = 2013
    YEAR = str(year)

c = calendar.Calendar(firstweekday = 0) # Sunday is day 6.

result = []
for month in range(0+1,12+1):
```

```
    MON = calendar.month_abbr[month]
    # list of weeks of tuples has too much structure
    # Use the overloaded list.__add__ operator to remove the week structure
    flatter = sum(c.monthdays2calendar(year, month), [])
    # make a dictionary keyed by number of day of week,
    # successively overwriting values.
    SUNDAY = {b: a for (a, b) in flatter if a}[6]
    result.append('{}-{}-{:2}'.format(YEAR, MON, SUNDAY))

print('\n'.join(result))
```

Racket[edit]

Python: Find_limit_of_recursion[edit]

```
import sys
print(sys.getrecursionlimit())
```

Python: Find_palindromic_numbers_in_both_binary_a

```
from itertools import islice

digits = "0123456789abcdefghijklmnopqrstuvwxyz"

def baseN(num,b):
  if num == 0: return "0"
  result = ""
  while num != 0:
    num, d = divmod(num, b)
    result += digits[d]
  return result[::-1] # reverse

def pal2(num):
    if num == 0 or num == 1: return True
    based = bin(num)[2:]
    return based == based[::-1]

def pal_23():
    yield 0
    yield 1
    n = 1
    while True:
        n += 1
```

```
        b = baseN(n, 3)
        revb = b[::-1]
        #if len(b) > 12: break
        for trial in ('{0}{1}'.format(b, revb), '{0}0{1}'.format(b, revb),
                      '{0}1{1}'.format(b, revb), '{0}2{1}'.format(b, revb)):
            t = int(trial, 3)
            if pal2(t):
                yield t

for pal23 in islice(pal_23(), 6):
    print(pal23, baseN(pal23, 3), baseN(pal23, 2))
```

```
Output:
```

```
0 0 0
1 1 1
6643 100010001 1100111110011
1422773 2200021200022 101011011010110110101
5415589 101012010210101 10100101010001010100101
90396755477 2212202222010222202122 1010100001100000100010000011000010101
```

Racket[edit]

Python: Find_the_last_Sunday_of_each_month[edit]

```
#!/usr/bin/python3

'''
    Output:
    2013-Jan-27
    2013-Feb-24
    2013-Mar-31
    2013-Apr-28
    2013-May-26
    2013-Jun-30
    2013-Jul-28
    2013-Aug-25
    2013-Sep-29
    2013-Oct-27
    2013-Nov-24
    2013-Dec-29
'''
```

```
import sys
import calendar

YEAR = sys.argv[-1]
try:
    year = int(YEAR)
except:
    year = 2013
    YEAR = str(year)

c = calendar.Calendar(firstweekday = 0) # Sunday is day 6.

result = []
for month in range(0+1,12+1):
    MON = calendar.month_abbr[month]
    # list of weeks of tuples has too much structure
    # Use the overloaded list.__add__ operator to remove the week structure
    flatter = sum(c.monthdays2calendar(year, month), [])
    # make a dictionary keyed by number of day of week,
    # successively overwriting values.
    SUNDAY = {b: a for (a, b) in flatter if a}[6]
    result.append('{}-{}-{:2}'.format(YEAR, MON, SUNDAY))

print('\n'.join(result))
```

Racket[edit]

Python: Find_unimplemented_tasks[edit]

Using XML.

```
import xml.dom.minidom
import urllib, sys

def findrc(category):
    name = "http://www.rosettacode.org/w/api.php?action=query&list=category
    cmcontinue, titles = '', []
    while True:
        u = urllib.urlopen(name + cmcontinue)
        xmldata = u.read()
        u.close()
        x = xml.dom.minidom.parseString(xmldata)
        titles += [i.getAttribute("title") for i in x.getElementsByTagName(
        cmcontinue = filter( None,
                             (urllib.quote(i.getAttribute("cmcontinue"))
                              for i in x.getElementsByTagName("categorymemb
```

```
        if cmcontinue:
            cmcontinue = '&cmcontinue=' + cmcontinue[0]
        else:
            break
    return titles

alltasks = findrc("Programming_Tasks")
lang = findrc(sys.argv[1])

for i in [i for i in alltasks if i not in lang]:
    print i
```

Python: First-class_Numbers[edit]

This new task:

```
IDLE 2.6.1
>>> # Number literals
>>> x,xi, y,yi = 2.0,0.5, 4.0,0.25
>>> # Numbers from calculation
>>> z  = x + y
>>> zi = 1.0 / (x + y)
>>> # The multiplier function is similar to 'compose' but with numbers
>>> multiplier = lambda n1, n2: (lambda m: n1 * n2 * m)
>>> # Numbers as members of collections
>>> numlist = [x, y, z]
>>> numlisti = [xi, yi, zi]
>>> # Apply numbers from list
>>> [multiplier(inversen, n)(.5) for n, inversen in zip(numlist, numlisti)]
[0.5, 0.5, 0.5]
>>>
```

The Python solution to First-class functions for comparison:

```
>>> # Some built in functions and their inverses
>>> from math import sin, cos, acos, asin
>>> # Add a user defined function and its inverse
>>> cube = lambda x: x * x * x
>>> croot = lambda x: x ** (1/3.0)
>>> # First class functions allow run-time creation of functions from funct
>>> # return function compose(f,g)(x) == f(g(x))
>>> compose = lambda f1, f2: ( lambda x: f1(f2(x)) )
>>> # first class functions should be able to be members of collection type
>>> funclist = [sin, cos, cube]
>>> funclisti = [asin, acos, croot]
>>> # Apply functions from lists as easily as integers
```

```
>>> [compose(inversef, f)(.5) for f, inversef in zip(funclist, funclisti)]
[0.5, 0.4999999999999999, 0.5]
>>>
```

As can be see, the treatment of functions is very close to the treatment of

Python: First_class_environments[edit]

In Python, name bindings are held in dicts, one for global scope and anothe

```
environments = [{'cnt':0, 'seq':i+1} for i in range(12)]

code = '''
print('% 4d' % seq, end='')
if seq != 1:
    cnt += 1
    seq = 3 * seq + 1 if seq & 1 else seq // 2
'''

while any(env['seq'] > 1 for env in environments):
    for env in environments:
        exec(code, globals(), env)
    print()

print('Counts')
for env in environments:
    print('% 4d' % env['cnt'], end='')
print()
```

Output:

```
   1   2   3   4   5   6   7   8   9  10  11  12
   1   1  10   2  16   3  22   4  28   5  34   6
   1   1   5   1   8  10  11   2  14  16  17   3
```

Python: Five_weekends[edit]

```
from datetime import timedelta, date

DAY     = timedelta(days=1)
START, STOP = date(1900, 1, 1), date(2101, 1, 1)
```

```
WEEKEND = {6, 5, 4}     # Sunday is day 6
FMT     = '%Y %m(%B)'

def fiveweekendspermonth(start=START, stop=STOP):
    'Compute months with five weekends between dates'

    when = start
    lastmonth = weekenddays = 0
    fiveweekends = []
    while when < stop:
        year, mon, _mday, _h, _m, _s, wday, _yday, _isdst = when.timetuple(
        if mon != lastmonth:
            if weekenddays >= 15:
                fiveweekends.append(when - DAY)
            weekenddays = 0
            lastmonth = mon
        if wday in WEEKEND:
            weekenddays += 1
        when += DAY
    return fiveweekends

dates = fiveweekendspermonth()
indent = '  '
print('There are %s months of which the first and last five are:' % len(date
print(indent +('\n'+indent).join(d.strftime(FMT) for d in dates[:5]))
print(indent +'...')
print(indent +('\n'+indent).join(d.strftime(FMT) for d in dates[-5:]))

print('\nThere are %i years in the range that do not have months with five w
      % len(set(range(START.year, STOP.year)) - {d.year for d in dates}))
```

Alternate Algorithm

The condition is equivalent to having a thirty-one day month in which the l

```
LONGMONTHS = (1, 3, 5, 7, 8, 10, 12) # Jan Mar May Jul Aug Oct Dec
def fiveweekendspermonth2(start=START, stop=STOP):
    return [date(yr, month, 31)
            for yr in range(START.year, STOP.year)
            for month in LONGMONTHS
            if date(yr, month, 31).timetuple()[6] == 6 # Sunday
            ]

dates2 = fiveweekendspermonth2()
assert dates2 == dates
```

Sample output:

```
There are 201 months of which the first and last five are:
  1901 03(March)
  1902 08(August)
  1903 05(May)
  1904 01(January)
  1904 07(July)
  ...
  2097 03(March)
  2098 08(August)
  2099 05(May)
  2100 01(January)
  2100 10(October)

There are 29 years in the range that do not have months with five weekends
```

Python: Flatten_a_list[edit]

Recursive[edit]

```
>>> def flatten(lst):
        return sum( ([x] if not isinstance(x, list) else flatten(x)
                     for x in lst), [] )

>>> lst = [[1], 2, [[3,4], 5], [[[]]], [[[6]]], 7, 8, []]
>>> flatten(lst)
[1, 2, 3, 4, 5, 6, 7, 8]
```

Non-recursive[edit]

Function flat is iterative and flattens the list in-place. It follows the P

```
>>> def flat(lst):
    i=0
    while i<len(lst):
        while True:
            try:
                lst[i:i+1] = lst[i]
            except (TypeError, IndexError):
                break
        i += 1

>>> lst = [[1], 2, [[3,4], 5], [[[]]], [[[6]]], 7, 8, []]
>>> flat(lst)
```

```
>>> lst
[1, 2, 3, 4, 5, 6, 7, 8]
```

Generative[edit]

This method shows a solution using Python generators.

flatten is a generator that yields the non-list values of its input in orde
In this case, the generator is converted back to a list before printing.

```
>>> def flatten(lst):
     for x in lst:
         if isinstance(x, list):
             for x in flatten(x):
                 yield x
         else:
             yield x


>>> lst = [[1], 2, [[3,4], 5], [[[]]], [[[6]]], 7, 8, []]
>>> print list(flatten(lst))
[1, 2, 3, 4, 5, 6, 7, 8]
```

R[edit]

Python: Flipping_bits_game[edit]

```
"""
Given a %i by %i sqare array of zeroes or ones in an initial
configuration, and a target configuration of zeroes and ones
The task is to transform one to the other in as few moves as
possible by inverting whole numbered rows or whole lettered
columns at once.
In an inversion any 1 becomes 0 and any 0 becomes 1 for that
whole row or column.

"""

from random import randrange
from copy import deepcopy
from string import ascii_lowercase
```

```
try:    # 2to3 fix
    input = raw_input
except:
    pass

N = 3   # N x N Square arrray

board  = [[0]* N for i in range(N)]

def setbits(board, count=1):
    for i in range(count):
        board[randrange(N)][randrange(N)] ^= 1

def shuffle(board, count=1):
    for i in range(count):
        if randrange(0, 2):
            fliprow(randrange(N))
        else:
            flipcol(randrange(N))


def pr(board, comment=''):
    print(str(comment))
    print('     ' + ' '.join(ascii_lowercase[i] for i in range(N)))
    print('  ' + '\n  '.join(' '.join(['%2s' % j] + [str(i) for i in line])
                             for j, line in enumerate(board, 1)))

def init(board):
    setbits(board, count=randrange(N)+1)
    target = deepcopy(board)
    while board == target:
        shuffle(board, count=2 * N)
    prompt = '  X, T, or 1-%i / %s-%s to flip: ' % (N, ascii_lowercase[0],
                                                     ascii_lowercase[N-1])
    return target, prompt

def fliprow(i):
    board[i-1][:] = [x ^ 1 for x in board[i-1] ]

def flipcol(i):
    for row in board:
        row[i] ^= 1

if __name__ == '__main__':
    print(__doc__ % (N, N))
    target, prompt = init(board)
    pr(target, 'Target configuration is:')
    print('')
    turns = 0
    while board != target:
        turns += 1
        pr(board, '%i:' % turns)
        ans = input(prompt).strip()
        if (len(ans) == 1
            and ans in ascii_lowercase and ascii_lowercase.index(ans) < N):
            flipcol(ascii_lowercase.index(ans))
```

```
        elif ans and all(ch in '0123456789' for ch in ans) and 1 <= int(ans
            fliprow(int(ans))
        elif ans == 'T':
            pr(target, 'Target configuration is:')
            turns -= 1
        elif ans == 'X':
            break
        else:
            print("  I don't understand %r... Try again. "
                  "(X to exit or T to show target)\n" % ans[:9])
            turns -= 1
    else:
        print('\nWell done!\nBye.')
```

Output:

```
Given a 3 by 3 sqare array of zeroes or ones in an initial
configuration, and a target configuration of zeroes and ones
The task is to transform one to the other in as few moves as
possible by inverting whole numbered rows or whole lettered
columns at once.
In an inversion any 1 becomes 0 and any 0 becomes 1 for that
whole row or column.
```

Python: Flood_fill[edit]

```
import Image
def FloodFill( fileName, initNode, targetColor, replaceColor ):
   img = Image.open( fileName )
   pix = img.load()
   xsize, ysize = img.size
   Q = []
   if pix[ initNode[0], initNode[1] ] != targetColor:
      return img
   Q.append( initNode )
   while Q != []:
      node = Q.pop(0)
      if pix[ node[0], node[1] ] == targetColor:
         W = list( node )
         if node[0] + 1 < xsize:
            E = list( [ node[0] + 1, node[1] ] )
         else:
            E = list( node )
      # Move west until color of node does not match targetColor
      while pix[ W[0], W[1] ] == targetColor:
         pix[ W[0], W[1] ] = replaceColor
```

```
         if W[1] + 1 < ysize:
            if pix[ W[0], W[1] + 1 ] == targetColor:
               Q.append( [ W[0], W[1] + 1 ] )
         if W[1] - 1 >= 0:
            if pix[ W[0], W[1] - 1 ] == targetColor:
               Q.append( [ W[0], W[1] - 1 ] )
         if W[0] - 1 >= 0:
            W[0] = W[0] - 1
         else:
            break
      # Move east until color of node does not match targetColor
      while pix[ E[0], E[1] ] == targetColor:
         pix[ E[0], E[1] ] = replaceColor
         if E[1] + 1 < ysize:
            if pix[ E[0], E[1] + 1 ] == targetColor:
               Q.append( [ E[0], E[1] + 1 ] )
         if E[1] - 1 >= 0:
            if pix[ E[0], E[1] - 1 ] == targetColor:
               Q.append( [ E[0], E[1] -1 ] )
         if E[0] + 1 < xsize:
            E[0] = E[0] + 1
         else:
            break
      return img
```

Usage example[edit]

```
# "FloodFillClean.png" is name of input file
# [55,55] the x,y coordinate where fill starts
# (0,0,0,255) the target colour being filled( black in this example )
# (255,255,255,255) the final colour ( white in this case )
img = FloodFill( "FloodFillClean.png", [55,55], (0,0,0,255), (255,255,255,2
#The resulting image is saved as Filled.png
img.save( "Filled.png" )
```

Python: Flow-control_structures[edit]

Loops[edit]

Python supports *break* and *continue* to exit from a loop early or short circu

```
# Search for an odd factor of a using brute force:
for i in range(n):
    if (n%2) == 0:
        continue
    if (n%i) == 0:
        result = i
        break
else:
    result = None
    print "No odd factors found"
```

In addition, as shown in the foregoing example, Python loops support an *els*

Python: Floyd's_triangle[edit]

```
>>> def floyd(rowcount=5):
        rows = [[1]]
        while len(rows) < rowcount:
                n = rows[-1][-1] + 1
                rows.append(list(range(n, n + len(rows[-1]) + 1)))
        return rows

>>> floyd()
[[1], [2, 3], [4, 5, 6], [7, 8, 9, 10], [11, 12, 13, 14, 15]]
>>> def pfloyd(rows=[[1], [2, 3], [4, 5, 6], [7, 8, 9, 10]]):
        colspace = [len(str(n)) for n in rows[-1]]
        for row in rows:
                print( ' '.join('%*i' % space_n for space_n in zip(colspace
```

```
>>> pfloyd()
1
2 3
4 5 6
7 8 9 10
>>> pfloyd(floyd(5))
 1
 2  3
 4  5  6
 7  8  9 10
11 12 13 14 15
>>> pfloyd(floyd(14))
 1
 2  3
 4  5  6
 7  8  9 10
```

```
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44  45
46 47 48 49 50 51 52 53  54  55
56 57 58 59 60 61 62 63  64  65  66
67 68 69 70 71 72 73 74  75  76  77  78
79 80 81 82 83 84 85 86  87  88  89  90  91
92 93 94 95 96 97 98 99 100 101 102 103 104 105
>>>
```

Alternately (using the mathematical formula for each row directly):

```
def floyd(rowcount=5):
    return [list(range(i*(i-1)//2+1, i*(i+1)//2+1))
            for i in range(1, rowcount+1)]
```

Racket[edit]

Python: Foreach[edit]

```
for i in collection:
   print i
```

Note: The Python for statement is always a "*foreach*" ... and the range() an

For example:

```
lines = words = characters = 0
f = open('somefile','r')
for eachline in f:
    lines += 1
    for eachword in eachline.split():
        words += 1
        for eachchar in eachword:
            characters += 1

print lines, words, characters
```

Whether for loops over the elements of the collection in order depends on t

One can loop over the key/value pairs of a dictionary in *alphabetic* or *nume*

```
d = {3: "Earth", 1: "Mercury", 4: "Mars", 2: "Venus"}
for k in sorted(d):
    print("%i: %s" % (k, d[k]))

d = {"London": "United Kingdom", "Berlin": "Germany", "Rome": "Italy", "Par
for k in sorted(d):
    print("%s: %s" % (k, d[k]))
```

Works with: Python version 2.x

```
d = {"fortytwo": 42, 3.14159: "pi", 23: "twentythree", "zero": 0, 13: "thir
for k in sorted(d):
    print("%s: %s" % (k, d[k]))
```

Python: Forest_fire[edit]

Just hit return to advance the simulation, or enter an integer to advance t
Entering 'p' will print the grid, and 'q' will quit. A summary of the grids

```
'''
Forest-Fire Cellular automation
 See: http://en.wikipedia.org/wiki/Forest-fire_model
'''

L = 15
# d = 2 # Fixed
initial_trees = 0.55
p = 0.01
f = 0.001

try:
    raw_input
except:
    raw_input = input

import random


tree, burning, space = 'TB.'
hood = ((-1,-1), (-1,0), (-1,1),
        (0,-1),          (0, 1),
        (1,-1),  (1,0),  (1,1))
```

```
def initialise():
    grid = {(x,y): (tree if random.random()<= initial_trees else space)
            for x in range(L)
            for y in range(L) }
    return grid

def gprint(grid):
    txt = '\n'.join(''.join(grid[(x,y)] for x in range(L))
                    for y in range(L))
    print(txt)

def quickprint(grid):
    t = b = 0
    ll = L * L
    for x in range(L):
        for y in range(L):
            if grid[(x,y)] in (tree, burning):
                t += 1
                if grid[(x,y)] == burning:
                    b += 1
    print(('Of %6i cells, %6i are trees of which %6i are currently burning.
          + ' (%6.3f%%, %6.3f%%)')
          % (ll, t, b, 100. * t / ll, 100. * b / ll))


def gnew(grid):
    newgrid = {}
    for x in range(L):
        for y in range(L):
            if grid[(x,y)] == burning:
                newgrid[(x,y)] = space
            elif grid[(x,y)] == space:
                newgrid[(x,y)] = tree if random.random()<= p else space
            elif grid[(x,y)] == tree:
                newgrid[(x,y)] = (burning
                                   if any(grid.get((x+dx,y+dy),space) == bu
                                           for dx,dy in hood)
                                        or random.random()<= f
                                   else tree)
    return newgrid

if __name__ == '__main__':
    grid = initialise()
    iter = 0
    while True:
        quickprint(grid)
        inp = raw_input('Print/Quit/<int>/<return> %6i: ' % iter).lower().s
        if inp:
            if inp[0] == 'p':
                gprint(grid)
            elif inp.isdigit():
                for i in range(int(inp)):
                    iter +=1
                    grid = gnew(grid)
                    quickprint(grid)
```

```
            elif inp[0] == 'q':
                break
        grid = gnew(grid)
        iter +=1
```

Sample output

```
Of    225 cells,    108 are trees of which      0 are currently burning. (4
Print/Quit/<int>/<return>      0:
Of    225 cells,    114 are trees of which      1 are currently burning. (5
Print/Quit/<int>/<return>      1: p
.TTT.T.T.TTTT.T
T.T.T.TT..T.T..
TT.TTTT...T.TT.
TTT..TTTTT.T..T
.T.TTT....TT.TT
...T..TTT.TT.T.
.TT.TT...TT..TT
.TT.T.T..T.T.T.
..TTT.TT.T..T..
.T....T.....TTT
T..TTT..T..T...
TTT....TTTTTT.T
......TBTTT...T
..T....TTTTTTTT
.T.T.T....TT...
Of    225 cells,    115 are trees of which      6 are currently burning. (5
Print/Quit/<int>/<return>      2: p
.TTT.TTT.TTTT.T
T.T.T.TT..T.T..
TT.TTTT...T.TT.
TTT..TTTTT.T..T
.T.TTT....TT.TT
...T..TTT.TT.T.
.TT.TT...TT..TT
.TT.T.T..T.T.T.
..TTT.TT.T..T..
.T....T.....TTT
T..TTT..T..T...
TTT....BBTTTT.T
....T.B.BTT...T
..T....BBTTTTTT
.T.T.T....TT...
Of    225 cells,    113 are trees of which      4 are currently burning. (5
Print/Quit/<int>/<return>      3: p
.TTT.TTT.TTTT.T
T.T.T.TT..T.T..
TT.TTTT...T.TT.
TTT..TTTTT.T..T
.T.TTT...TTT.TT
...T..TTT.TTTTT
.TT.TT...TT..TT
```

```
.TT.T.T..T.T.T.
..TTT.TT.T..T..
.T.T..T.....TTT
T..TTT..B..T...
TTT......BTTT.T
....T....BT...T
..T......BTTTTT
.T.T.T....TT...
Of    225 cells,    110 are trees of which      4 are currently burning. (4
Print/Quit/<int>/<return>      4:
```

Python: Fork[edit]

Works with: Python version 2.5

```
import os

pid = os.fork()
if pid > 0:
 # parent code
else:
 # child code
```

Racket[edit]

Python: Fork_Process[edit]

Works with: Python version 2.5

```
import os

pid = os.fork()
if pid > 0:
 # parent code
else:
 # child code
```

Python: Formatted_numeric_output[edit]

Works with: Python version 2.5

Python has 3 different floating point formatting methods: "%e","%f" & "%g".

Python: Forward_difference[edit]

```
>>> dif = lambda s: [x-s[i] for i,x in enumerate(s[1:])]
>>> # or, dif = lambda s: [x-y for x,y in zip(s[1:],s)]
>>> difn = lambda s, n: difn(dif(s), n-1) if n else s

>>> s = [90, 47, 58, 29, 22, 32, 55, 5, 55, 73]
>>> difn(s, 0)
[90, 47, 58, 29, 22, 32, 55, 5, 55, 73]
>>> difn(s, 1)
[-43, 11, -29, -7, 10, 23, -50, 50, 18]
>>> difn(s, 2)
[54, -40, 22, 17, 13, -73, 100, -32]

>>> from pprint import pprint
>>> pprint( [difn(s, i) for i in xrange(10)] )
[[90, 47, 58, 29, 22, 32, 55, 5, 55, 73],
 [-43, 11, -29, -7, 10, 23, -50, 50, 18],
 [54, -40, 22, 17, 13, -73, 100, -32],
 [-94, 62, -5, -4, -86, 173, -132],
 [156, -67, 1, -82, 259, -305],
 [-223, 68, -83, 341, -564],
 [291, -151, 424, -905],
 [-442, 575, -1329],
 [1017, -1904],
 [-2921]]
```

Python: Four_bits_adder[edit]

Individual boolean bits are represented by either 1, 0, True (interchangeab

Python: Fractal_tree[edit]

File:Fractal-tree-python.png

Library: pygame

```
import pygame, math

pygame.init()
window = pygame.display.set_mode((600, 600))
pygame.display.set_caption("Fractal Tree")
```

```
screen = pygame.display.get_surface()

def drawTree(x1, y1, angle, depth):
    if depth:
        x2 = x1 + int(math.cos(math.radians(angle)) * depth * 10.0)
        y2 = y1 + int(math.sin(math.radians(angle)) * depth * 10.0)
        pygame.draw.line(screen, (255,255,255), (x1, y1), (x2, y2), 2)
        drawTree(x2, y2, angle - 20, depth - 1)
        drawTree(x2, y2, angle + 20, depth - 1)

def input(event):
    if event.type == pygame.QUIT:
        exit(0)

drawTree(300, 550, -90, 9)
pygame.display.flip()
while True:
    input(pygame.event.wait())
```

Python: Free_polyominoes_enumeration[edit]

Translation of: Haskell

```
from itertools import imap, imap, groupby, chain, imap
from operator import itemgetter
from sys import argv
from array import array

def concat_map(func, it):
    return list(chain.from_iterable(imap(func, it)))

def minima(poly):
    """Finds the min x and y coordiate of a Polyomino."""
    return (min(pt[0] for pt in poly), min(pt[1] for pt in poly))

def translate_to_origin(poly):
    (minx, miny) = minima(poly)
    return [(x - minx, y - miny) for (x, y) in poly]

rotate90   = lambda (x, y): ( y, -x)
rotate180  = lambda (x, y): (-x, -y)
rotate270  = lambda (x, y): (-y,  x)
reflect    = lambda (x, y): (-x,  y)

def rotations_and_reflections(poly):
    """All the plane symmetries of a rectangular region."""
    return (poly,
            map(rotate90, poly),
            map(rotate180, poly),
            map(rotate270, poly),
            map(reflect, poly),
```

```
            [reflect(rotate90(pt)) for pt in poly],
            [reflect(rotate180(pt)) for pt in poly],
            [reflect(rotate270(pt)) for pt in poly])

def canonical(poly):
    return min(sorted(translate_to_origin(pl)) for pl in rotations_and_refle

def unique(lst):
    lst.sort()
    return map(next, imap(itemgetter(1), groupby(lst)))

# All four points in Von Neumann neighborhood.
contiguous = lambda (x, y): [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)

def new_points(poly):
    """Finds all distinct points that can be added to a Polyomino."""
    return unique([pt for pt in concat_map(contiguous, poly) if pt not in po

def new_polys(poly):
    return unique([canonical(poly + [pt]) for pt in new_points(poly)])

monomino = [(0, 0)]
monominoes = [monomino]

def rank(n):
    """Generates polyominoes of rank n recursively."""
    assert n >= 0
    if n == 0: return []
    if n == 1: return monominoes
    return unique(concat_map(new_polys, rank(n - 1)))

def text_representation(poly):
    """Generates a textual representation of a Polyomino."""
    min_pt = minima(poly)
    max_pt = (max(p[0] for p in poly), max(p[1] for p in poly))
    table = [array('c', ' ') * (max_pt[1] - min_pt[1] + 1)
             for _ in xrange(max_pt[0] - min_pt[0] + 1)]
    for pt in poly:
        table[pt[0] - min_pt[0]][pt[1] - min_pt[1]] = '#'
    return "\n".join(row.tostring() for row in table)

def main():
    print [len(rank(n)) for n in xrange(1, 11)]

    n = int(argv[1]) if (len(argv) == 2) else 5
    print "\nAll free polyominoes of rank %d:" % n

    for poly in rank(n):
        print text_representation(poly), "\n"

main()
```

Output:

```
[1, 1, 2, 5, 12, 35, 108, 369, 1285, 4655]

All free polyominoes of rank 5:
#####

####
#

####
 #

###
##

###
# #

###
#
#

###
 #
 #

###
  ##

##
 ##
 #

##
 ##
  #

##
 #
 ##

 #
###
 #
```

Python: Function_as_an_Argument[edit]

Works with: Python version 2.5

```
def first(function):
```

```
    return function()

def second():
    return "second"

result = first(second)
```

or

```
  result = first(lambda: "second")
```

Functions are first class objects in Python. They can be bound to names ("a

Q[edit]

Python: Function_composition[edit]

```
compose = lambda f, g: lambda x: f( g(x) )
```

Example use:

```
>>> compose = lambda f, g: lambda x: f( g(x) )
>>> from math import sin, asin
>>> sin_asin = compose(sin, asin)
>>> sin_asin(0.5)
0.5
>>>
```

Qi[edit]

Python: Function_definition[edit]

Function definition:

```
def multiply(a, b):
    return a * b
```

Lambda function definition:

Python: Function_frequency[edit]

Works with: Python version 3.x

This code parses a Python source file using the built-in **ast** module and cou

```
import ast

class CallCountingVisitor(ast.NodeVisitor):

    def __init__(self):
        self.calls = {}

    def visit_Call(self, node):
        if isinstance(node.func, ast.Name):
            fun_name = node.func.id
            call_count = self.calls.get(fun_name, 0)
            self.calls[fun_name] = call_count + 1
        self.generic_visit(node)

filename = input('Enter a filename to parse: ')
with open(filename, encoding='utf-8') as f:
    contents = f.read()
root = ast.parse(contents, filename=filename) #NOTE: this will throw a Synt
visitor = CallCountingVisitor()
visitor.visit(root)
top10 = sorted(visitor.calls.items(), key=lambda x: x[1], reverse=True)[:10
for name, count in top10:
    print(name,'called',count,'times')
```

The result of running the program on the ftplib module of Python 3.2:

```
Enter a filename to parse: c:\Python32\Lib\ftplib.py
error_reply called 10 times
print called 10 times
error_proto called 8 times
```

```
callback called 8 times
```

[Python: Functional_Composition][edit]

```
compose = lambda f, g: lambda x: f( g(x) )
```

Example use:

```
>>> compose = lambda f, g: lambda x: f( g(x) )
>>> from math import sin, asin
>>> sin_asin = compose(sin, asin)
>>> sin_asin(0.5)
0.5
>>>
```

[Python: GUI_component_interaction][edit]

Library: Tkinter

```
import random
from Tkinter import *
import tkMessageBox


class Application(Frame):
    def __init__(self, master):
        Frame.__init__(self, master)
        self.counter = 0
        self.contents = StringVar()
        self.contents.set(str(self.counter))
        self.pack(expand=True, fill='both', padx=10, pady=15)
        self.create_widgets()

    def increment(self, *args):
        self.counter += 1
        self.update_entry()

    def random(self):
        if tkMessageBox.askyesno("Confirmation", "Reset to random value ?")
            self.counter = random.randint(0, 5000)
            self.update_entry()

    def entry_updated(self, event, *args):
        if not event.char:
            return 'break'
```

```
        if not event.char.isdigit():
            tkMessageBox.showerror('Error', 'Invalid input !')
            return 'break'
        self.counter = int('%s%s' % (self.contents.get(), event.char))

    def update_entry(self):
        self.contents.set(str(self.counter))
        self.entry['textvariable'] = self.contents

    def create_widgets(self):
        options = {'expand': True, 'fill': 'x', 'side': 'left', 'padx': 5}
        self.entry = Entry(self)
        self.entry.bind('<Key>', self.entry_updated)
        self.entry.pack(**options)
        self.update_entry()
        self.increment_button = Button(self, text='Increment', command=self
        self.increment_button.pack(**options)
        self.random_button = Button(self, text='Random', command=self.rando
        self.random_button.pack(**options)


if __name__ == '__main__':
    root = Tk()
    try:
        app = Application(master=root)
        app.master.title("Rosetta code")
        app.mainloop()
    except KeyboardInterrupt:
        root.destroy()
```

File:GuiInterationPython.png

Python: Gale-Shapely_algorithm[edit]

```
import copy

guyprefers = {
 'abe':  ['abi', 'eve', 'cath', 'ivy', 'jan', 'dee', 'fay', 'bea', 'hope',
 'bob':  ['cath', 'hope', 'abi', 'dee', 'eve', 'fay', 'bea', 'jan', 'ivy',
 'col':  ['hope', 'eve', 'abi', 'dee', 'bea', 'fay', 'ivy', 'gay', 'cath',
 'dan':  ['ivy', 'fay', 'dee', 'gay', 'hope', 'eve', 'jan', 'bea', 'cath',
 'ed':   ['jan', 'dee', 'bea', 'cath', 'fay', 'eve', 'abi', 'ivy', 'hope',
 'fred': ['bea', 'abi', 'dee', 'gay', 'eve', 'ivy', 'cath', 'jan', 'hope',
 'gav':  ['gay', 'eve', 'ivy', 'bea', 'cath', 'abi', 'dee', 'hope', 'jan',
 'hal':  ['abi', 'eve', 'hope', 'fay', 'ivy', 'cath', 'jan', 'bea', 'gay',
 'ian':  ['hope', 'cath', 'dee', 'gay', 'bea', 'abi', 'fay', 'ivy', 'jan',
 'jon':  ['abi', 'fay', 'jan', 'gay', 'eve', 'bea', 'dee', 'cath', 'ivy', '
galprefers = {
 'abi':  ['bob', 'fred', 'jon', 'gav', 'ian', 'abe', 'dan', 'ed', 'col', 'h
```

```
 'bea':  ['bob', 'abe', 'col', 'fred', 'gav', 'dan', 'ian', 'ed', 'jon', 'h
 'cath': ['fred', 'bob', 'ed', 'gav', 'hal', 'col', 'ian', 'abe', 'dan', 'j
 'dee':  ['fred', 'jon', 'col', 'abe', 'ian', 'hal', 'gav', 'dan', 'bob', '
 'eve':  ['jon', 'hal', 'fred', 'dan', 'abe', 'gav', 'col', 'ed', 'ian', 'b
 'fay':  ['bob', 'abe', 'ed', 'ian', 'jon', 'dan', 'fred', 'gav', 'col', 'h
 'gay':  ['jon', 'gav', 'hal', 'fred', 'bob', 'abe', 'col', 'ed', 'dan', 'i
 'hope': ['gav', 'jon', 'bob', 'abe', 'ian', 'dan', 'hal', 'ed', 'col', 'fr
 'ivy':  ['ian', 'col', 'hal', 'gav', 'fred', 'bob', 'abe', 'ed', 'jon', 'd
 'jan':  ['ed', 'hal', 'gav', 'abe', 'bob', 'jon', 'col', 'ian', 'fred', 'd

guys = sorted(guyprefers.keys())
gals = sorted(galprefers.keys())


def check(engaged):
    inverseengaged = dict((v,k) for k,v in engaged.items())
    for she, he in engaged.items():
        shelikes = galprefers[she]
        shelikesbetter = shelikes[:shelikes.index(he)]
        helikes = guyprefers[he]
        helikesbetter = helikes[:helikes.index(she)]
        for guy in shelikesbetter:
            guysgirl = inverseengaged[guy]
            guylikes = guyprefers[guy]
            if guylikes.index(guysgirl) > guylikes.index(she):
                print("%s and %s like each other better than "
                      "their present partners: %s and %s, respectively"
                      % (she, guy, he, guysgirl))
                return False
        for gal in helikesbetter:
            girlsguy = engaged[gal]
            gallikes = galprefers[gal]
            if gallikes.index(girlsguy) > gallikes.index(he):
                print("%s and %s like each other better than "
                      "their present partners: %s and %s, respectively"
                      % (he, gal, she, girlsguy))
                return False
    return True

def matchmaker():
    guysfree = guys[:]
    engaged  = {}
    guyprefers2 = copy.deepcopy(guyprefers)
    galprefers2 = copy.deepcopy(galprefers)
    while guysfree:
        guy = guysfree.pop(0)
        guyslist = guyprefers2[guy]
        gal = guyslist.pop(0)
        fiance = engaged.get(gal)
        if not fiance:
            # She's free
            engaged[gal] = guy
            print("  %s and %s" % (guy, gal))
        else:
            # The bounder proposes to an engaged lass!
            galslist = galprefers2[gal]
```

```
            if galslist.index(fiance) > galslist.index(guy):
                # She prefers new guy
                engaged[gal] = guy
                print("  %s dumped %s for %s" % (gal, fiance, guy))
                if guyprefers2[fiance]:
                    # Ex has more girls to try
                    guysfree.append(fiance)
            else:
                # She is faithful to old fiance
                if guyslist:
                    # Look again
                    guysfree.append(guy)
    return engaged


print('\nEngagements:')
engaged = matchmaker()

print('\nCouples:')
print('  ' + ',\n  '.join('%s is engaged to %s' % couple
                          for couple in sorted(engaged.items())))
print()
print('Engagement stability check PASSED'
      if check(engaged) else 'Engagement stability check FAILED')

print('\n\nSwapping two fiances to introduce an error')
engaged[gals[0]], engaged[gals[1]] = engaged[gals[1]], engaged[gals[0]]
for gal in gals[:2]:
    print('  %s is now engaged to %s' % (gal, engaged[gal]))
print()
print('Engagement stability check PASSED'
      if check(engaged) else 'Engagement stability check FAILED')
```

Output:

```
Engagements:
  abe and abi
  bob and cath
  col and hope
  dan and ivy
  ed and jan
  fred and bea
  gav and gay
  hope dumped col for ian
  abi dumped abe for jon
  hal and eve
  col and dee
  ivy dumped dan for abe
  dan and fay

Couples:
```

```
  abi is engaged to jon,
  bea is engaged to fred,
  cath is engaged to bob,
  dee is engaged to col,
  eve is engaged to hal,
  fay is engaged to dan,
  gay is engaged to gav,
  hope is engaged to ian,
  ivy is engaged to abe,
  jan is engaged to ed

Engagement stability check PASSED


Swapping two fiances to introduce an error
  abi is now engaged to fred
  bea is now engaged to jon

fay and jon like each other better than their present partners: dan and bea
Engagement stability check FAILED
```

Python: Gamma_function[edit]

Translation of: Ada

```
_a =    ( 1.00000000000000000000, 0.57721566490153286061, -0.65587807152025
         -0.04200263503409523553, 0.16653861138229148950, -0.04219773455554
         -0.00962197152787697356, 0.00721894324666309954, -0.00116516759185
         -0.00021524167411495097, 0.00012805028238811619, -0.00002013485478
         -0.00000125049348214267, 0.00000113302723198170, -0.00000020563384
          0.00000000611609510448, 0.00000000500200764447, -0.00000000118127
          0.00000000010434267117, 0.00000000000778226344, -0.00000000000369
          0.00000000000051003703, -0.00000000000002058326, -0.0000000000000
          0.00000000000000122678, -0.00000000000000011813, 0.0000000000000
          0.00000000000000000141, -0.00000000000000000023, 0.0000000000000
        )
def gamma (x):
   y  = float(x) - 1.0;
   sm = _a[-1];
   for an in _a[-2::-1]:
      sm = sm * y + an;
   return 1.0 / sm;


if __name__ == '__main__':
    for i in range(1,11):
        print "  %20.14e" % gamma(i/3.0)
```

Output:

```
  2.67893853470775e+00
  1.35411793942640e+00
  1.00000000000000e+00
  8.92979511569249e-01
  9.02745292950934e-01
  1.00000000000000e+00
  1.19063934875900e+00
  1.50457548825154e+00
  1.99999999999397e+00
  2.77815847933857e+00
```

Python: Gaussian_elimination[edit]

```
# The 'gauss' function takes two matrices, 'a' and 'b', with 'a' square, an
# If 'b' is the identity, then 'x' is the inverse of 'a'.

import copy
from fractions import Fraction

def gauss(a, b):
    a = copy.deepcopy(a)
    b = copy.deepcopy(b)
    n = len(a)
    p = len(b[0])
    det = 1
    for i in range(n - 1):
        k = i
        for j in range(i + 1, n):
            if abs(a[j][i]) > abs(a[k][i]):
                k = j
        if k != i:
            a[i], a[k] = a[k], a[i]
            b[i], b[k] = b[k], b[i]
            det = -det

        for j in range(i + 1, n):
            t = a[j][i]/a[i][i]
            for k in range(i + 1, n):
                a[j][k] -= t*a[i][k]
            for k in range(p):
                b[j][k] -= t*b[i][k]

    for i in range(n - 1, -1, -1):
        for j in range(i + 1, n):
            t = a[i][j]
            for k in range(p):
                b[i][k] -= t*b[j][k]
```

```
        t = 1/a[i][i]
        det *= a[i][i]
        for j in range(p):
            b[i][j] *= t
    return det, b

def zeromat(p, q):
    return [[0]*q for i in range(p)]

def matmul(a, b):
    n, p = len(a), len(a[0])
    p1, q = len(b), len(b[0])
    if p != p1:
        raise ValueError("Incompatible dimensions")
    c = zeromat(n, q)
    for i in range(n):
        for j in range(q):
                c[i][j] = sum(a[i][k]*b[k][j] for k in range(p))
    return c


def mapmat(f, a):
    return [list(map(f, v)) for v in a]

def ratmat(a):
    return mapmat(Fraction, a)

# As an example, compute the determinant and inverse of 3x3 magic square

a = [[2, 9, 4], [7, 5, 3], [6, 1, 8]]
b = [[1, 0, 0], [0, 1, 0], [0, 0, 1]]
det, c = gauss(a, b)

det
-360.0

c
[[-0.10277777777777776, 0.18888888888888888, -0.019444444444444438],
[0.10555555555555554, 0.0222222222222223, -0.061111111111111116],
[0.0638888888888889, -0.14444444444444446, 0.1472222222222223]]

# Check product
matmul(a, c)
[[1.0, 0.0, 0.0], [5.551115123125783e-17, 1.0, 0.0],
[1.1102230246251565e-16, -2.220446049250313e-16, 1.0]]

# Same with fractions, so the result is exact

det, c = gauss(ratmat(a), ratmat(b))

det
Fraction(-360, 1)

c
[[Fraction(-37, 360), Fraction(17, 90), Fraction(-7, 360)],
[Fraction(19, 180), Fraction(1, 45), Fraction(-11, 180)],
```

```
[Fraction(23, 360), Fraction(-13, 90), Fraction(53, 360)]]

matmul(a, c)
[[Fraction(1, 1), Fraction(0, 1), Fraction(0, 1)],
[Fraction(0, 1), Fraction(1, 1), Fraction(0, 1)],
[Fraction(0, 1), Fraction(0, 1), Fraction(1, 1)]]
```

Python: General_FizzBuzz[edit]

```
def genfizzbuzz(factorwords, numbers):
    factorwords.sort(key=lambda p: p[0])
    lines = []
    for num in numbers:
        words = ''.join(wrd for fact, wrd in factorwords if (num % fact) ==
        lines.append(words if words else str(num))
    return '\n'.join(lines)

if __name__ == '__main__':
    print(genfizzbuzz([(5, 'Buzz'), (3, 'Fizz'), (7, 'Baxx')], range(1, 21)
```

Output:

```
1
2
Fizz
4
Buzz
Fizz
Baxx
8
Fizz
Buzz
11
Fizz
13
Baxx
FizzBuzz
16
17
Fizz
19
Buzz
```

Python: Generate_lower_case_ASCII_alphabet[edit]

```
# From the standard library:
from string import ascii_lowercase

# Generation:
lower = [chr(i) for i in range(ord('a'), ord('z') + 1)]
```

Python: Generator.1[edit]

In Python, any function that contains a yield statement becomes a generator

Works with: Python version 2.6+ and 3.x
(in versions prior to 2.6, replace next(something) with something.next())

```
from itertools import islice, count

def powers(m):
    for n in count():
        yield n ** m

def filtered(s1, s2):
    v, f = next(s1), next(s2)
    while True:
        if v > f:
            f = next(s2)
            continue
        elif v < f:
            yield v
        v = next(s1)

squares, cubes = powers(2), powers(3)
f = filtered(squares, cubes)
print(list(islice(f, 20, 30)))
```

Output:

```
[529, 576, 625, 676, 784, 841, 900, 961, 1024, 1089]
```

Python: Generic_swap[edit]

Python has support for swapping built in:

```
a, b = b, a
```

But the task calls for a "generic swap method" to be written, so here it is

```
def swap(a, b):
    return b, a
```

Note that tuples are immutable in Python. This function doesn't mutate anyt

Python: Get_system_command_output[edit]

```
>>> import subprocess
>>> returned_text = subprocess.check_output("dir", shell=True, universal_ne
>>> type(returned_text)
<class 'str'>
>>> print(returned_text)
 Volume in drive C is Windows
 Volume Serial Number is 44X7-73CE

 Directory of C:\Python33

04/07/2013  06:40    <DIR>          .
04/07/2013  06:40    <DIR>          ..
27/05/2013  07:10    <DIR>          DLLs
27/05/2013  07:10    <DIR>          Doc
27/05/2013  07:10    <DIR>          include
27/05/2013  07:10    <DIR>          Lib
27/05/2013  07:10    <DIR>          libs
16/05/2013  00:15            33,326 LICENSE.txt
15/05/2013  22:49           214,554 NEWS.txt
16/05/2013  00:03            26,624 python.exe
16/05/2013  00:03            27,136 pythonw.exe
15/05/2013  22:49             6,701 README.txt
27/05/2013  07:10    <DIR>          tcl
27/05/2013  07:10    <DIR>          Tools
16/05/2013  00:02            43,008 w9xpopen.exe
               6 File(s)        351,349 bytes
               9 Dir(s)  46,326,947,840 bytes free

>>> # Ref: https://docs.python.org/3/library/subprocess.html
```

Python: Globally_replace_text_in_several_files[ed

From Python docs. (Note: in-place editing does not work for MS-DOS 8+3 file

```
import fileinput

for line in fileinput.input(inplace=True):
    print(line.replace('Goodbye London!', 'Hello New York!'), end='')
```

Python: Gnome_sort[edit]

```
>>> def gnomesort(a):
        i,j,size = 1,2,len(a)
        while i < size:
                if a[i-1] <= a[i]:
                        i,j = j, j+1
                else:
                        a[i-1],a[i] = a[i],a[i-1]
                        i -= 1
                        if i == 0:
                                i,j = j, j+1
        return a

>>> gnomesort([3,4,2,5,1,6])
[1, 2, 3, 4, 5, 6]
>>>
```

Python: Gray_code[edit]

This example works with lists of discrete binary digits.

First some int<>bin conversion routines

```
>>> def int2bin(n):
        'From positive integer to list of binary bits, msb at index 0'
```

```
        if n:
                bits = []
                while n:
                        n,remainder = divmod(n, 2)
                        bits.insert(0, remainder)
                return bits
        else: return [0]
```

```
>>> def bin2int(bits):
        'From binary bits, msb at index 0 to integer'
        i = 0
        for bit in bits:
                i = i * 2 + bit
        return i
```

Now the bin<>gray converters.

These follow closely the methods in the animation seen here: [Converting Betw

```
>>> def bin2gray(bits):
        return bits[:1] + [i ^ ishift for i, ishift in zip(bits[:-1], bits[

>>> def gray2bin(bits):
        b = [bits[0]]
        for nextb in bits[1:]: b.append(b[-1] ^ nextb)
        return b
```

Sample output

```
>>> for i in range(16):
        print('int:%2i -> bin:%12r -> gray:%12r -> bin:%12r -> int:%2i' %
              ( i,
                int2bin(i),
                bin2gray(int2bin(i)),
                gray2bin(bin2gray(int2bin(i))),
                bin2int(gray2bin(bin2gray(int2bin(i))))
              ))
```

```
int: 0 -> bin:          [0] -> gray:          [0] -> bin:          [0] -> int:
int: 1 -> bin:          [1] -> gray:          [1] -> bin:          [1] -> int:
int: 2 -> bin:       [1, 0] -> gray:       [1, 1] -> bin:       [1, 0] -> int:
int: 3 -> bin:       [1, 1] -> gray:       [1, 0] -> bin:       [1, 1] -> int:
int: 4 -> bin:    [1, 0, 0] -> gray:    [1, 1, 0] -> bin:    [1, 0, 0] -> int:
int: 5 -> bin:    [1, 0, 1] -> gray:    [1, 1, 1] -> bin:    [1, 0, 1] -> int:
int: 6 -> bin:    [1, 1, 0] -> gray:    [1, 0, 1] -> bin:    [1, 1, 0] -> int:
```

```
int: 7 -> bin:   [1, 1, 1] -> gray:   [1, 0, 0] -> bin:   [1, 1, 1] -> int:
int: 8 -> bin:[1, 0, 0, 0] -> gray:[1, 1, 0, 0] -> bin:[1, 0, 0, 0] -> int:
int: 9 -> bin:[1, 0, 0, 1] -> gray:[1, 1, 0, 1] -> bin:[1, 0, 0, 1] -> int:
int:10 -> bin:[1, 0, 1, 0] -> gray:[1, 1, 1, 1] -> bin:[1, 0, 1, 0] -> int:
int:11 -> bin:[1, 0, 1, 1] -> gray:[1, 1, 1, 0] -> bin:[1, 0, 1, 1] -> int:
int:12 -> bin:[1, 1, 0, 0] -> gray:[1, 0, 1, 0] -> bin:[1, 1, 0, 0] -> int:
int:13 -> bin:[1, 1, 0, 1] -> gray:[1, 0, 1, 1] -> bin:[1, 1, 0, 1] -> int:
int:14 -> bin:[1, 1, 1, 0] -> gray:[1, 0, 0, 1] -> bin:[1, 1, 1, 0] -> int:
int:15 -> bin:[1, 1, 1, 1] -> gray:[1, 0, 0, 0] -> bin:[1, 1, 1, 1] -> int:
>>>
```

Python: Grayscale_image[edit]

Works with: Python version 3.1

Extending the example given here

```
# String masquerading as ppm file (version P3)
import io
ppmfileout = io.StringIO('')

def togreyscale(self):
    for h in range(self.height):
        for w in range(self.width):
            r, g, b = self.get(w, h)
            l = int(0.2126 * r + 0.7152 * g + 0.0722 * b)
            self.set(w, h, Colour(l, l, l))

Bitmap.togreyscale = togreyscale


# Draw something simple
bitmap = Bitmap(4, 4, white)
bitmap.fillrect(1, 0, 1, 2, Colour(127, 0, 63))
bitmap.set(3, 3, Colour(0, 127, 31))
print('Colour:')
# Write to the open 'file' handle
bitmap.writeppmp3(ppmfileout)
print(ppmfileout.getvalue())
print('Grey:')
bitmap.togreyscale()
ppmfileout = io.StringIO('')
bitmap.writeppmp3(ppmfileout)
print(ppmfileout.getvalue())


'''
The print statement above produces the following output :
```

```
Colour:
P3
# generated from Bitmap.writeppmp3
4 4
255
   255 255 255   255 255 255   255 255 255     0 127  31
   255 255 255   255 255 255   255 255 255   255 255 255
   255 255 255   127   0  63   255 255 255   255 255 255
   255 255 255   127   0  63   255 255 255   255 255 255

Grey:
P3
# generated from Bitmap.writeppmp3
4 4
254
   254 254 254   254 254 254   254 254 254    93  93  93
   254 254 254   254 254 254   254 254 254   254 254 254
   254 254 254    31  31  31   254 254 254   254 254 254
   254 254 254    31  31  31   254 254 254   254 254 254

'''
```

R[edit]

Python: Greatest_common_divisor[edit]

Built-in[edit]

Works with: Python version 2.6+

```
from fractions import gcd
```

Iterative Euclid algorithm[edit]

Python: Greatest_element_of_a_list[edit]

The built-in Python function max() already does this.

```
max(values)
```

Of course this assumes we have a list or tuple (or other sequence like obje

If we truly were receiving a stream of data then in Python, such streams ar

max(), (and min()), can take iterables and a key argument which takes a fun

```
>>> floatstrings = ['1\n', ' 2.3\n', '4.5e-1\n', '0.01e4\n', '-1.2']
>>> max(floatstrings, key = float)
'0.01e4\n'
>>>
```

Normally we would want the converted form as the maximum and we could just

```
>>> max(float(x) for x in floatstrings)
100.0
>>>
```

Or you can write your own functional version, of the maximum function, usin

```
>>> mylist = [47, 11, 42, 102, 13]
>>> reduce(lambda a,b: a if (a > b) else b, mylist)
102
```

Q[edit]

Python: Greatest_subsequential_sum[edit]

Naive, inefficient but really simple solution which tests all possible subs

```
def maxsubseq(seq):
  return max((seq[begin:end] for begin in xrange(len(seq)+1)
                             for end in xrange(begin, len(seq)+1)),
             key=sum)
```

Classic linear-time constant-space solution based on algorithm from "Progra

```
def maxsum(sequence):
    """Return maximum sum."""
    maxsofar, maxendinghere = 0, 0
    for x in sequence:
        # invariant: ``maxendinghere`` and ``maxsofar`` are accurate for ``:
        maxendinghere = max(maxendinghere + x, 0)
        maxsofar = max(maxsofar, maxendinghere)
    return maxsofar
```

Adapt the above-mentioned solution to return maximizing subsequence. See ht

```
def maxsumseq(sequence):
    start, end, sum_start = -1, -1, -1
    maxsum_, sum_ = 0, 0
    for i, x in enumerate(sequence):
        sum_ += x
        if maxsum_ < sum_: # found maximal subsequence so far
            maxsum_ = sum_
            start, end = sum_start, i
        elif sum_ < 0: # start new sequence
            sum_ = 0
            sum_start = i
    assert maxsum_ == maxsum(sequence)
    assert maxsum_ == sum(sequence[start + 1:end + 1])
    return sequence[start + 1:end + 1]
```

Modify ``maxsumseq()`` to allow any iterable not just sequences.

```
def maxsumit(iterable):
    maxseq = seq = []
    start, end, sum_start = -1, -1, -1
    maxsum_, sum_ = 0, 0
    for i, x in enumerate(iterable):
        seq.append(x); sum_ += x
        if maxsum_ < sum_:
            maxseq = seq; maxsum_ = sum_
            start, end = sum_start, i
        elif sum_ < 0:
            seq = []; sum_ = 0
            sum_start = i
    assert maxsum_ == sum(maxseq[:end - start])
    return maxseq[:end - start]
```

Elementary tests:

Python: HTTPS_Request[edit]

Python's **urllib.request** library, (**urllib2** in Python2.x) has support for SSL

Python 3.x:

```
from urllib.request import urlopen
print(urlopen('https://sourceforge.net/').read())
```

(Python 2.x)

```
from urllib2 import urlopen
print urlopen('https://sourceforge.net/').read()
```

Python: HTTPS_request[edit]

Python's **urllib.request** library, (**urllib2** in Python2.x) has support for SSL

Python 3.x:

```
from urllib.request import urlopen
print(urlopen('https://sourceforge.net/').read())
```

(Python 2.x)

```
from urllib2 import urlopen
print urlopen('https://sourceforge.net/').read()
```

Python: HTTPS_request_with_authentication[edit]

Works with: Python version 2.4 and 2.6

Note: You should install ***mechanize*** to run code below. Visit: http://wwwsear

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from mechanize import Browser

USER_AGENT = "Mozilla/5.0 (X11; U; Linux i686; tr-TR; rv:1.8.1.9) Gecko/200

br = Browser()
br.addheaders = [("User-agent", USER_AGENT)]

# remove comment if you get debug output
# br.set_debug_redirects(True)
# br.set_debug_responses(True)
# br.set_debug_http(True)

br.open("https://www.facebook.com")

br.select_form("loginform")
br['email'] = "xxxxxxx@xxxxx.com"
br['pass'] = "xxxxxxxxx"
br['persistent'] = ["1"]

response = br.submit()
print response.read()
```

Racket[edit]

Python: HTTP_Request[edit]

Python 3

Using the urllib.request module.

```
import urllib.request
print(urllib.request.urlopen("http://rosettacode.org").read())
```

Python: Hailstone_sequence[edit]

```
def hailstone(n):
    seq = [n]
    while n>1:
        n = 3*n + 1 if n & 1 else n//2
        seq.append(n)
    return seq

if __name__ == '__main__':
    h = hailstone(27)
    assert len(h)==112 and h[:4]==[27, 82, 41, 124] and h[-4:]==[8, 4, 2, 1
    print("Maximum length %i was found for hailstone(%i) for numbers <100,0
          max((len(hailstone(i)), i) for i in range(1,100000)))
```

```
Output:
```

```
Maximum length 351 was found for hailstone(77031) for numbers <100,000
```

R[edit]

Python: Hamming_numbers[edit]

Version based on example from Dr. Dobb's CodeTalk[edit]

```
from itertools import islice

def hamming2():
    '''\
    This version is based on a snippet from:
```

```
        http://dobbscodetalk.com/index.php?option=com_content&task=view&id=

        When expressed in some imaginary pseudo-C with automatic
        unlimited storage allocation and BIGNUM arithmetics, it can be
        expressed as:
            hamming = h where
              array h;
              n=0; h[0]=1; i=0; j=0; k=0;
              x2=2*h[ i ]; x3=3*h[j]; x5=5*h[k];
              repeat:
                h[++n] = min(x2,x3,x5);
                if (x2==h[n]) { x2=2*h[++i]; }
                if (x3==h[n]) { x3=3*h[++j]; }
                if (x5==h[n]) { x5=5*h[++k]; }
    '''
    h = 1
    _h=[h]    # memoized
    multipliers  = (2, 3, 5)
    multindeces  = [0 for i in multipliers] # index into _h for multipliers
    multvalues   = [x * _h[i] for x,i in zip(multipliers, multindeces)]
    yield h
    while True:
        h = min(multvalues)
        _h.append(h)
        for (n,(v,x,i)) in enumerate(zip(multvalues, multipliers, multindec
            if v == h:
                i += 1
                multindeces[n] = i
                multvalues[n]  = x * _h[i]
        # cap the memoization
        mini = min(multindeces)
        if mini >= 1000:
            del _h[:mini]
            multindeces = [i - mini for i in multindeces]
        #
        yield h
```

Output:

Python: Handle_a_signal[edit]

Simple version

```
import time

def counter():
    n = 0
```

```
    t1 = time.time()
    while True:
        try:
            time.sleep(0.5)
            n += 1
            print n
        except KeyboardInterrupt, e:
            print 'Program has run for %5.3f seconds.' % (time.time() - t1)
            break

counter()
```

The following example should work on all platforms.

```
import time

def intrptWIN():
   procDone = False
   n = 0

   while not procDone:
      try:
         time.sleep(0.5)
         n += 1
         print n
      except KeyboardInterrupt, e:
         procDone = True

t1 = time.time()
intrptWIN()
tdelt = time.time() - t1
print 'Program has run for %5.3f seconds.' % tdelt
```

There is a signal module in the standard distribution
that accomodates the UNIX type signal mechanism.
However the pause() mechanism is not implemented on Windows versions.

```
import signal, time, threading
done = False
n = 0

def counter():
   global n, timer
   n += 1
   print n
   timer = threading.Timer(0.5, counter)
   timer.start()

def sigIntHandler(signum, frame):
```

```
    global done
    timer.cancel()
    done = True

def intrptUNIX():
    global timer
    signal.signal(signal.SIGINT, sigIntHandler)

    timer = threading.Timer(0.5, counter)
    timer.start()
    while not done:
       signal.pause()

t1 = time.time()
intrptUNIX()
tdelt = time.time() - t1
print 'Program has run for %5.3f seconds.' % tdelt
```

How about this one? It should work on all platforms;
and it does show how to install a signal handler:

```
import time, signal

class WeAreDoneException(Exception):
    pass

def sigIntHandler(signum, frame):
    signal.signal(signal.SIGINT, signal.SIG_DFL) # resets to default handle
    raise WeAreDoneException

t1 = time.time()

try:
    signal.signal(signal.SIGINT, sigIntHandler)
    n = 0
    while True:
        time.sleep(0.5)
        n += 1
        print n
except WeAreDoneException:
    pass

tdelt = time.time() - t1
print 'Program has run for %5.3f seconds.' % tdelt
```

Python: Happy_Number[edit]

```
>>> def happy(n):
```

```
    past = set()
    while n != 1:
        n = sum(int(i)**2 for i in str(n))
        if n in past:
            return False
        past.add(n)
    return True

>>> [x for x in xrange(500) if happy(x)][:8]
[1, 7, 10, 13, 19, 23, 28, 31]
```

R[edit]

Python: Happy_numbers[edit]

```
>>> def happy(n):
    past = set()
    while n != 1:
        n = sum(int(i)**2 for i in str(n))
        if n in past:
            return False
        past.add(n)
    return True

>>> [x for x in xrange(500) if happy(x)][:8]
[1, 7, 10, 13, 19, 23, 28, 31]
```

Python: Hash_from_two_arrays[edit]

Works with: Python version 3.0+ and 2.7

Shows off the dict comprehensions in Python 3 (that were back-ported to 2.7

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
hash = {key: value for key, value in zip(keys, values)}
```

Works with: Python version 2.2+

```
keys = ['a', 'b', 'c']
```

```
values = [1, 2, 3]
hash = dict(zip(keys, values))

# Lazily, Python 2.3+, not 3.x:
from itertools import izip
hash = dict(izip(keys, values))
```

Works with: Python version 2.0+

```
keys = ['a', 'b', 'c']
values = [1, 2, 3]
hash = {}
for k,v in zip(keys, values):
    hash[k] = v
```

The original (Ruby) example uses a range of different types as keys. Here i

```
>>> class Hashable(object):
        def __hash__(self):
                return id(self) ^ 0xBEEF


>>> my_inst = Hashable()
>>> my_int = 1
>>> my_complex = 0 + 1j
>>> my_float = 1.2
>>> my_string = "Spam"
>>> my_bool = True
>>> my_unicode = u'Ham'
>>> my_list = ['a', 7]
>>> my_tuple = ( 0.0, 1.4 )
>>> my_set = set(my_list)
>>> def my_func():
        pass

>>> class my_class(object):
        pass

>>> keys = [my_inst, my_tuple, my_int, my_complex, my_float, my_string,
        my_bool, my_unicode, frozenset(my_set), tuple(my_list),
        my_func, my_class]
>>> values = range(12)
>>> d = dict(zip(keys, values))
>>> for key, value in d.items(): print key, ":", value

1 : 6
1j : 3
Ham : 7
Spam : 5
(0.0, 1.3999999999999999) : 1
```

```
frozenset(['a', 7]) : 8
1.2 : 4
('a', 7) : 9
<function my_func at 0x0128E7B0> : 10
<class '__main__.my_class'> : 11
<__main__.Hashable object at 0x012AFC50> : 0
>>> # Notice that the key "True" disappeared, and its value got associated w
>>> # This is because 1 == True in Python, and dictionaries cannot have two
```

Python: Hash_join[edit]

```
from collections import defaultdict

def hashJoin(table1, index1, table2, index2):
    h = defaultdict(list)
    # hash phase
    for s in table1:
        h[s[index1]].append(s)
    # join phase
    return [(s, r) for r in table2 for s in h[r[index2]]]

table1 = [(27, "Jonah"),
          (18, "Alan"),
          (28, "Glory"),
          (18, "Popeye"),
          (28, "Alan")]
table2 = [("Jonah", "Whales"),
          ("Jonah", "Spiders"),
          ("Alan", "Ghosts"),
          ("Alan", "Zombies"),
          ("Glory", "Buffy")]

for row in hashJoin(table1, 1, table2, 0):
    print(row)
```

Output:

```
((27, 'Jonah'), ('Jonah', 'Whales'))
((27, 'Jonah'), ('Jonah', 'Spiders'))
((18, 'Alan'), ('Alan', 'Ghosts'))
((28, 'Alan'), ('Alan', 'Ghosts'))
((18, 'Alan'), ('Alan', 'Zombies'))
((28, 'Alan'), ('Alan', 'Zombies'))
((28, 'Glory'), ('Glory', 'Buffy'))
```

Python: Haversine_formula[edit]

```
from math import radians, sin, cos, sqrt, asin

def haversine(lat1, lon1, lat2, lon2):

  R = 6372.8 # Earth radius in kilometers

  dLat = radians(lat2 - lat1)
  dLon = radians(lon2 - lon1)
  lat1 = radians(lat1)
  lat2 = radians(lat2)

  a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2
  c = 2*asin(sqrt(a))

  return R * c

>>> haversine(36.12, -86.67, 33.94, -118.40)
2887.2599506071106
>>>
```

R[edit]

Python: Heapsort[edit]

```
def heapsort(lst):
  ''' Heapsort. Note: this function sorts in-place (it mutates the list). '

  # in pseudo-code, heapify only called once, so inline it here
  for start in range((len(lst)-2)/2, -1, -1):
    siftdown(lst, start, len(lst)-1)

  for end in range(len(lst)-1, 0, -1):
    lst[end], lst[0] = lst[0], lst[end]
    siftdown(lst, 0, end - 1)
  return lst

def siftdown(lst, start, end):
  root = start
  while True:
    child = root * 2 + 1
    if child > end: break
    if child + 1 <= end and lst[child] < lst[child + 1]:
      child += 1
    if lst[root] < lst[child]:
      lst[root], lst[child] = lst[child], lst[root]
      root = child
    else:
```

```
      break
```

Testing:

```
>>> ary = [7, 6, 5, 9, 8, 4, 3, 1, 2, 0]
>>> heapsort(ary)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Python: Hello_world.1[edit]

Works with: Python version 2.4

```
print "Hello world!"
```

The same using sys.stdout

```
import sys
sys.stdout.write("Hello world!\n")
```

In Python 3.0, print is changed from a statement to a function.

Python: Here_document[edit]

Python does not have here-docs.
It does however have triple-quoted strings which can be used similarly.

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

Racket[edit]

Python: Heronian_triangles[edit]

```
from __future__ import division, print_function
from math import sqrt
from fractions import gcd
from itertools import product


def hero(a, b, c):
    s = (a + b + c) / 2
    a2 = s*(s-a)*(s-b)*(s-c)
    return sqrt(a2) if a2 > 0 else 0


def is_heronian(a, b, c):
    a = hero(a, b, c)
    return a > 0 and a.is_integer()


def gcd3(x, y, z):
    return gcd(gcd(x, y), z)


if __name__ == '__main__':
    maxside = 200
    h = [(a, b, c) for a,b,c in product(range(1, maxside + 1), repeat=3)
         if a <= b <= c and a + b > c and gcd3(a, b, c) == 1 and is_heronia
    h.sort(key = lambda x: (hero(*x), sum(x), x[::-1]))   # By increasing a
    print('Primitive Heronian triangles with sides up to %i:' % maxside, le
    print('\nFirst ten when ordered by increasing area, then perimeter,then
    print('\n'.join('  %14r perim: %3i area: %i'
                    % (sides, sum(sides), hero(*sides)) for sides in h[:10]
    print('\nAll with area 210 subject to the previous ordering:')
    print('\n'.join('  %14r perim: %3i area: %i'
                    % (sides, sum(sides), hero(*sides)) for sides in h
                    if hero(*sides) == 210))
```

Output:

```
Primitive Heronian triangles with sides up to 200: 517
```

Python: Hickerson_series_of_almost_integers[edit]

This uses Pythons decimal module of fixed precision decimal floating point

```
from decimal import Decimal
import math

def h(n):
    'Simple, reduced precision calculation'
    return math.factorial(n) / (2 * math.log(2) ** (n + 1))

def h2(n):
    'Extended precision Hickerson function'
    return Decimal(math.factorial(n)) / (2 * Decimal(2).ln() ** (n + 1))

for n in range(18):
    x = h2(n)
    norm = str(x.normalize())
    almostinteger = (' Nearly integer'
                     if 'E' not in norm and ('.0' in norm or '.9' in norm)
                     else ' NOT nearly integer!')
    print('n:%2i h:%s%s' % (n, norm, almostinteger))
```

Output:

```
n: 0 h:0.7213475204444817036799623405 NOT nearly integer!
n: 1 h:1.040684490502803898934790802 Nearly integer
n: 2 h:3.002780707156905443499767406 Nearly integer
n: 3 h:12.99629050527696646222488454 Nearly integer
n: 4 h:74.99873544766160012763455035 Nearly integer
n: 5 h:541.0015185164235075692027746 Nearly integer
n: 6 h:4683.001247262257437180467151 Nearly integer
n: 7 h:47292.99873131462390482283547 Nearly integer
n: 8 h:545834.9979074851670672910395 Nearly integer
n: 9 h:7087261.001622899120979187513 Nearly integer
n:10 h:102247563.0052710420110883885 Nearly integer
n:11 h:1622632572.997550049852874859 Nearly integer
n:12 h:28091567594.98157244071518915 Nearly integer
n:13 h:526858348381.0012482861804887 Nearly integer
n:14 h:10641342970443.08453192709506 Nearly integer
n:15 h:230283190977853.0374360391257 Nearly integer
n:16 h:5315654681981354.513076743451 NOT nearly integer!
n:17 h:130370767029135900.4579853491 NOT nearly integer!
```

The range for should be reduced to be for this definition of almost integ

Python: Hidato[edit]

```
board = []
```

```
given = []
start = None

def setup(s):
    global board, given, start
    lines = s.splitlines()
    ncols = len(lines[0].split())
    nrows = len(lines)
    board = [[-1] * (ncols + 2) for _ in xrange(nrows + 2)]

    for r, row in enumerate(lines):
        for c, cell in enumerate(row.split()):
            if cell == "__" :
                board[r + 1][c + 1] = 0
                continue
            elif cell == ".":
                continue # -1
            else:
                val = int(cell)
                board[r + 1][c + 1] = val
                given.append(val)
                if val == 1:
                    start = (r + 1, c + 1)
    given.sort()

def solve(r, c, n, next=0):
    if n > given[-1]:
        return True
    if board[r][c] and board[r][c] != n:
        return False
    if board[r][c] == 0 and given[next] == n:
        return False

    back = 0
    if board[r][c] == n:
        next += 1
        back = n

    board[r][c] = n
    for i in xrange(-1, 2):
        for j in xrange(-1, 2):
            if solve(r + i, c + j, n + 1, next):
                return True
    board[r][c] = back
    return False

def print_board():
    d = {-1: "  ", 0: "__"}
    bmax = max(max(r) for r in board)
    form = "%" + str(len(str(bmax)) + 1) + "s"
    for r in board[1:-1]:
        print "".join(form % d.get(c, str(c)) for c in r[1:-1])

hi = """\
__ 33 35 __ __  .  .  .
__ __ 24 22 __  .  .  .
```

```
__ __ __ 21 __ __  .  .
__ 26 __ 13 40 11  .  .
27 __ __ __  9 __  1  .
 .  . __ __ 18 __ __  .
 .  .  .  . __  7 __ __
 .  .  .  .  .  .  5 __"""

setup(hi)
print_board()
solve(start[0], start[1], 1)
print
print_board()
```

Output:

```
 __ 33 35 __ __
 __ __ 24 22 __
 __ __ __ 21 __ __
 __ 26 __ 13 40 11
 27 __ __ __  9 __  1
       __ __ 18 __ __
             __  7 __ __
                   5 __

 32 33 35 36 37
 31 34 24 22 38
 30 25 23 21 12 39
 29 26 20 13 40 11
 27 28 14 19  9 10  1
       15 16 18  8  2
             17  7  6  3
                   5  4
```

Racket[edit]

Python: Higher-order_functions[edit]

Works with: Python version 2.5

```
def first(function):
    return function()

def second():
    return "second"
```

```
result = first(second)
```

or

```
  result = first(lambda: "second")
```

Functions are first class objects in Python. They can be bound to names ("a

Q[edit]

Python: History_variables[edit]

```
import sys

HIST = {}

def trace(frame, event, arg):
    for name,val in frame.f_locals.items():
        if name not in HIST:
            HIST[name] = []
        else:
            if HIST[name][-1] is val:
                continue
        HIST[name].append(val)
    return trace

def undo(name):
    HIST[name].pop(-1)
    return HIST[name][-1]

def main():
    a = 10
    a = 20

    for i in range(5):
        c = i

    print "c:", c, "-> undo x3 ->",
    c = undo('c')
    c = undo('c')
    c = undo('c')
    print c
```

```
    print 'HIST:', HIST

sys.settrace(trace)
main()
```

Output:

```
c: 4 -> undo x3 -> 1
HIST: {'a': [10, 20], 'i': [0, 1, 2, 3, 4], 'c': [0, 1], 'name': ['c']}
```

PicoLisp[edit]

Python: Hofstadter-Conway_$10,000_sequence[edit]

```
from __future__ import division

def maxandmallows(nmaxpower2=20):
    # Note: The first hc number is returned in hc[1];
    # hc[0] is not part of the series.
    nmax = 2**nmaxpower2
    hc, mallows, mx, mxpow2 = [None, 1, 1], None, (0.5, 2), []
    for n in range(2, nmax + 1):
        ratio = hc[n] / n
        if ratio > mx[0]: mx = (ratio, n)
        if ratio >= 0.55: mallows = n
        if ratio == 0.5:
            print("In the region %7i < n <= %7i: max a(n)/n = %s" %
                  ((n)//2, n, "%6.4f at  n = %i" % mx))
            mxpow2.append(mx[0])
            mx = (ratio, n)
        hc.append(hc[hc[n]] + hc[-hc[n]])
    return hc, mallows if mxpow2 and mxpow2[-1] < 0.55 and n > 4 else None

if __name__ == '__main__':
    hc, mallows = maxandmallows(20)
    if mallows:
        print("\nYou too might have won $1000 with the mallows number of %i
```

Sample output

Python: Hofstadter_Figure-Figure_sequences[edit]

```
def ffr(n):
    if n < 1 or type(n) != int: raise ValueError("n must be an int >= 1")
    try:
        return ffr.r[n]
    except IndexError:
        r, s = ffr.r, ffs.s
        ffr_n_1 = ffr(n-1)
        lastr = r[-1]
        # extend s up to, and one past, last r
        s += list(range(s[-1] + 1, lastr))
        if s[-1] < lastr: s += [lastr + 1]
        # access s[n-1] temporarily extending s if necessary
        len_s = len(s)
        ffs_n_1 = s[n-1] if len_s > n else (n - len_s) + s[-1]
        ans = ffr_n_1 + ffs_n_1
        r.append(ans)
        return ans
ffr.r = [None, 1]

def ffs(n):
    if n < 1 or type(n) != int: raise ValueError("n must be an int >= 1")
    try:
        return ffs.s[n]
    except IndexError:
        r, s = ffr.r, ffs.s
        for i in range(len(r), n+2):
            ffr(i)
            if len(s) > n:
                return s[n]
        raise Exception("Whoops!")
ffs.s = [None, 2]

if __name__ == '__main__':
    first10 = [ffr(i) for i in range(1,11)]
    assert first10 == [1, 3, 7, 12, 18, 26, 35, 45, 56, 69], "ffr() value e
    print("ffr(n) for n = [1..10] is", first10)
    #
    bin = [None] + [0]*1000
    for i in range(40, 0, -1):
        bin[ffr(i)] += 1
    for i in range(960, 0, -1):
        bin[ffs(i)] += 1
    if all(b == 1 for b in bin[1:1000]):
        print("All Integers 1..1000 found OK")
    else:
        print("All Integers 1..1000 NOT found only once: ERROR")
```

Output

```
ffr(n) for n = [1..10] is [1, 3, 7, 12, 18, 26, 35, 45, 56, 69]
All Integers 1..1000 found OK
```

Alternative[edit]

```
cR = [1]
cS = [2]

def extend_RS():
        x = cR[len(cR) - 1] + cS[len(cR) - 1]
        cR.append(x)
        cS += range(cS[-1] + 1, x)
        cS.append(x + 1)

def ff_R(n):
        assert(n > 0)
        while n > len(cR): extend_RS()
        return cR[n - 1]

def ff_S(n):
        assert(n > 0)
        while n > len(cS): extend_RS()
        return cS[n - 1]

# tests
print([ ff_R(i) for i in range(1, 11) ])

s = {}
for i in range(1, 1001): s[i] = 0
for i in range(1, 41):  del s[ff_R(i)]
for i in range(1, 961): del s[ff_S(i)]

# the fact that we got here without a key error
print("Ok")
```

output

```
[1, 3, 7, 12, 18, 26, 35, 45, 56, 69]
Ok
```

Using cyclic iterators[edit]

Translation of: Haskell

Defining R and S as mutually recursive generators. Follows directly from th

```
from itertools import islice

def R():
        n = 1
        yield n
        for s in S():
                n += s
                yield n;

def S():
        yield 2
        yield 4
        u = 5
        for r in R():
                if r <= u: continue;
                for x in range(u, r): yield x
                u = r + 1

def lst(s, n): return list(islice(s(), n))

print "R:", lst(R, 10)
print "S:", lst(S, 10)
print sorted(lst(R, 40) + lst(S, 960)) == list(range(1,1001))

# perf test case
# print sum(lst(R, 10000000))
```

Output:

```
R: [1, 3, 7, 12, 18, 26, 35, 45, 56, 69]
S: [2, 4, 5, 6, 8, 9, 10, 11, 13, 14]
True
```

Racket[edit]

Python: Hofstadter_Q_sequence[edit]

```
def q(n):
    if n < 1 or type(n) != int: raise ValueError("n must be an int >= 1")
    try:
        return q.seq[n]
    except IndexError:
        ans = q(n - q(n - 1)) + q(n - q(n - 2))
        q.seq.append(ans)
```

```
        return ans
q.seq = [None, 1, 1]

if __name__ == '__main__':
    first10 = [q(i) for i in range(1,11)]
    assert first10 == [1, 1, 2, 3, 3, 4, 5, 5, 6, 6], "Q() value error(s)"
    print("Q(n) for n = [1..10] is:", ', '.join(str(i) for i in first10))
    assert q(1000) == 502, "Q(1000) value error"
    print("Q(1000) =", q(1000))
```

Extra credit

If you try and initially compute larger values of n then you tend to hit th

The function q1 gets around this by calling function q to extend the Q seri

The following code is to be concatenated to the code above:

```
from sys import getrecursionlimit

def q1(n):
    if n < 1 or type(n) != int: raise ValueError("n must be an int >= 1")
    try:
        return q.seq[n]
    except IndexError:
        len_q, rlimit = len(q.seq), getrecursionlimit()
        if (n - len_q) > (rlimit // 5):
            for i in range(len_q, n, rlimit // 5):
                q(i)
        ans = q(n - q(n - 1)) + q(n - q(n - 2))
        q.seq.append(ans)
        return ans

if __name__ == '__main__':
    tmp = q1(100000)
    print("Q(i+1) < Q(i) for i [1..100000] is true %i times." %
          sum(k1 < k0 for k0, k1 in zip(q.seq[1:], q.seq[2:])))
```

Combined output:

```
Q(n) for n = [1..10] is: 1, 1, 2, 3, 3, 4, 5, 5, 6, 6
Q(1000) = 502
Q(i+1) < Q(i) for i [1..10000] is true 49798 times.
```

Alternative[edit]

```
def q(n):
    l = len(q.seq)
    while l <= n:
        q.seq.append(q.seq[l - q.seq[l - 1]] + q.seq[l - q.seq[l - 2]])
        l += 1
    return q.seq[n]
q.seq = [None, 1, 1]

print("Q(n) for n = [1..10] is:", [q(i) for i in range(1, 11)])
print("Q(1000) =", q(1000))
q(100000)
print("Q(i+1) < Q(i) for i [1..100000] is true %i times." %
      sum([q.seq[i] > q.seq[i + 1] for i in range(1, 100000)]))
```

Racket[edit]

Python: Holidays_related_to_Easter[edit]

Works with: Python version 2.6

Library: python-dateutil

Unfortunately, at present Python doesn't support date formatting for any da

```
from dateutil.easter import *
import datetime, calendar

class Holiday(object):
    def __init__(self, date, offset=0):
        self.holiday = date + datetime.timedelta(days=offset)

    def __str__(self):
        dayofweek = calendar.day_name[self.holiday.weekday()][0:3]
        month = calendar.month_name[self.holiday.month][0:3]
        return '{0} {1:2d} {2}'.format(dayofweek, self.holiday.day, month)

def get_holiday_values(year):
    holidays = {'year': year}
    easterDate = easter(year)
    holidays['easter'] = Holiday(easterDate)
    holidays['ascension'] = Holiday(easterDate, 39)
    holidays['pentecost'] = Holiday(easterDate, 49)
    holidays['trinity'] = Holiday(easterDate, 56)
```

```
    holidays['corpus'] = Holiday(easterDate, 60)
    return holidays

def print_holidays(holidays):
    print '{year:4d} Easter: {easter}, Ascension: {ascension}, Pentecost: {

if __name__ == "__main__":
    print "Christian holidays, related to Easter, for each centennial from 
    for year in range(400, 2200, 100):
        print_holidays(get_holiday_values(year))

    print ''
    print "Christian holidays, related to Easter, for years from 2010 to 20
    for year in range(2010, 2021):
        print_holidays(get_holiday_values(year))
```

Output:

```
Christian holidays, related to Easter, for each centennial from 400 to 2100
 400 Easter: Sun  2 Apr, Ascension: Thu 11 May, Pentecost: Sun 21 May, Trin
 500 Easter: Sun  4 Apr, Ascension: Thu 13 May, Pentecost: Sun 23 May, Trin
 600 Easter: Sun 13 Apr, Ascension: Thu 22 May, Pentecost: Sun  1 Jun, Trin
 700 Easter: Sun 15 Apr, Ascension: Thu 24 May, Pentecost: Sun  3 Jun, Trin
 800 Easter: Sun 23 Apr, Ascension: Thu  1 Jun, Pentecost: Sun 11 Jun, Trin
 900 Easter: Sun 28 Mar, Ascension: Thu  6 May, Pentecost: Sun 16 May, Trin
```

Python: Horizontal_sundial_calculations[edit]

Translation of: ALGOL 68

```
from __future__ import print_function
import math
try: raw_input
except: raw_input = input

lat = float(raw_input("Enter latitude       => "))
lng = float(raw_input("Enter longitude      => "))
ref = float(raw_input("Enter legal meridian => "))
print()

slat = math.sin(math.radians(lat))
print("    sine of latitude:   %.3f" % slat)
print("    diff longitude:     %.3f" % (lng-ref))
print()
print("Hour, sun hour angle, dial hour line angle from 6am to 6pm")

for h in range(-6, 7):
  hra = 15 * h
```

```
  hra -= lng - ref
  hla = math.degrees(math.atan(slat * math.tan(math.radians(hra))))
  print("HR=%3d; HRA=%7.3f; HLA=%7.3f" % (h, hra, hla))
```

Output:

```
Enter latitude       => -4.95
Enter longitude      => -150.5
Enter legal meridian => -150

    sine of latitude:   -0.086
    diff longitude:     -0.500

Hour, sun hour angle, dial hour line angle from 6am to 6pm
HR= -6; HRA=-89.500; HLA= 84.225
HR= -5; HRA=-74.500; HLA= 17.283
HR= -4; HRA=-59.500; HLA=  8.334
HR= -3; HRA=-44.500; HLA=  4.847
HR= -2; HRA=-29.500; HLA=  2.795
HR= -1; HRA=-14.500; HLA=  1.278
HR=  0; HRA=  0.500; HLA= -0.043
HR=  1; HRA= 15.500; HLA= -1.371
HR=  2; HRA= 30.500; HLA= -2.910
HR=  3; HRA= 45.500; HLA= -5.018
HR=  4; HRA= 60.500; HLA= -8.671
HR=  5; HRA= 75.500; HLA=-18.451
HR=  6; HRA= 90.500; HLA= 84.225
```

Racket[edit]

Python: Horner's_rule_for_polynomial_evaluation[e

```
>>> def horner(coeffs, x):
        acc = 0
        for c in reversed(coeffs):
                acc = acc * x + c
        return acc

>>> horner( (-19, 7, -4, 6), 3)
128
```

Functional version[edit]

```
>>> try: from functools import reduce
except: pass

>>> def horner(coeffs, x):
        return reduce(lambda acc, c: acc * x + c, reversed(coeffs), 0)

>>> horner( (-19, 7, -4, 6), 3)
128
```

Library: numpy
[edit]

```
>>> import numpy
>>> numpy.polynomial.polynomial.polyval(3, (-19, 7, -4, 6))
128.0
```

R[edit]

Python: Host_Introspection[edit]

```
>>> import platform, sys, socket
>>> platform.architecture()
('64bit', 'ELF')
>>> platform.machine()
'x86_64'
>>> platform.node()
'yourhostname'
>>> platform.system()
'Linux'
>>> sys.byteorder
little
>>> socket.gethostname()
'yourhostname'
>>>
```

R[edit]

Python: Host_introspection[edit]

```
>>> import platform, sys, socket
>>> platform.architecture()
('64bit', 'ELF')
>>> platform.machine()
'x86_64'
>>> platform.node()
'yourhostname'
>>> platform.system()
'Linux'
>>> sys.byteorder
little
>>> socket.gethostname()
'yourhostname'
>>>
```

Python: Hostname[edit]

Works with: Python version 2.5

Python: Hough_transform[edit]

Library: PIL

This is the classical Hough transform as described in wikipedia. The code d

Python: Huffman_codes[edit]

A slight modification of the method outlined in the task description allows

The output is sorted first on length of the code, then on the symbols.

```
from heapq import heappush, heappop, heapify
from collections import defaultdict

def encode(symb2freq):
    """Huffman encode the given dict mapping symbols to weights"""
    heap = [[wt, [sym, ""]] for sym, wt in symb2freq.items()]
    heapify(heap)
    while len(heap) > 1:
        lo = heappop(heap)
```

```
        hi = heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return sorted(heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

txt = "this is an example for huffman encoding"
symb2freq = defaultdict(int)
for ch in txt:
    symb2freq[ch] += 1
# in Python 3.1+:
# symb2freq = collections.Counter(txt)
huff = encode(symb2freq)
print "Symbol\tWeight\tHuffman Code"
for p in huff:
    print "%s\t%s\t%s" % (p[0], symb2freq[p[0]], p[1])
```

Python: Huffman_coding[edit]

A slight modification of the method outlined in the task description allows

The output is sorted first on length of the code, then on the symbols.

```
from heapq import heappush, heappop, heapify
from collections import defaultdict

def encode(symb2freq):
    """Huffman encode the given dict mapping symbols to weights"""
    heap = [[wt, [sym, ""]] for sym, wt in symb2freq.items()]
    heapify(heap)
    while len(heap) > 1:
        lo = heappop(heap)
        hi = heappop(heap)
        for pair in lo[1:]:
            pair[1] = '0' + pair[1]
        for pair in hi[1:]:
            pair[1] = '1' + pair[1]
        heappush(heap, [lo[0] + hi[0]] + lo[1:] + hi[1:])
    return sorted(heappop(heap)[1:], key=lambda p: (len(p[-1]), p))

txt = "this is an example for huffman encoding"
symb2freq = defaultdict(int)
for ch in txt:
    symb2freq[ch] += 1
# in Python 3.1+:
# symb2freq = collections.Counter(txt)
huff = encode(symb2freq)
print "Symbol\tWeight\tHuffman Code"
```

```
for p in huff:
    print "%s\t%s\t%s" % (p[0], symb2freq[p[0]], p[1])
```

Output:

```
Symbol  Weight  Huffman Code
    6   101
n   4   010
a   3   1001
e   3   1100
f   3   1101
h   2   0001
i   3   1110
m   2   0010
o   2   0011
s   2   0111
g   1   00000
l   1   00001
p   1   01100
r   1   01101
t   1   10000
u   1   10001
x   1   11110
c   1   111110
d   1   111111
```

An extension to the method outlined above is given here.

Python: I.Q._Puzzle[edit]

```
#
# Draw board triangle in ascii
#
def DrawBoard(board):
  peg = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
  for n in xrange(1,16):
    peg[n] = '.'
    if n in board:
      peg[n] = "%X" % n
  print "     %s" % peg[1]
  print "    %s %s" % (peg[2],peg[3])
  print "   %s %s %s" % (peg[4],peg[5],peg[6])
```

```
  print "  %s %s %s %s" % (peg[7],peg[8],peg[9],peg[10])
  print " %s %s %s %s %s" % (peg[11],peg[12],peg[13],peg[14],peg[15])
#

# remove peg n from board
def RemovePeg(board,n):
  board.remove(n)

# Add peg n on board
def AddPeg(board,n):
  board.append(n)

# return true if peg N is on board else false is empty position
def IsPeg(board,n):
  return n in board

# A dictionary of valid jump moves index by jumping peg
# then a list of moves where move has jumpOver and LandAt positions
JumpMoves = { 1: [ (2,4),(3,6) ],  # 1 can jump over 2 to land on 4, or jum
              2: [ (4,7),(5,9)  ],
              3: [ (5,8),(6,10) ],
              4: [ (2,1),(5,6),(7,11),(8,13) ],
              5: [ (8,12),(9,14) ],
              6: [ (3,1),(5,4),(9,13),(10,15) ],
              7: [ (4,2),(8,9)  ],
              8: [ (5,3),(9,10) ],
              9: [ (5,2),(8,7)  ],
             10: [ (9,8) ],
             11: [ (12,13) ],
             12: [ (8,5),(13,14) ],
             13: [ (8,4),(9,6),(12,11),(14,15) ],
             14: [ (9,5),(13,12)  ],
             15: [ (10,6),(14,13) ]
            }

Solution = []
#
# Recursively solve the problem
#
def Solve(board):
  #DrawBoard(board)
  if len(board) == 1:
    return board # Solved one peg left
  # try a move for each peg on the board
  for peg in xrange(1,16): # try in numeric order not board order
    if IsPeg(board,peg):
      movelist = JumpMoves[peg]
      for over,land in movelist:
        if IsPeg(board,over) and not IsPeg(board,land):
          saveboard = board[:] # for back tracking
          RemovePeg(board,peg)
          RemovePeg(board,over)
          AddPeg(board,land) # board order changes!

          Solution.append((peg,over,land))
```

```
          board = Solve(board)
          if len(board) == 1:
            return board
        ## undo move and back track when stuck!
          board = saveboard[:] # back track
          del Solution[-1] # remove last move
  return board

#
# Remove one peg and start solving
#
def InitSolve(empty):
  board = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
  RemovePeg(board,empty_start)
  Solve(board)

#
empty_start = 1
InitSolve(empty_start)

board = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
RemovePeg(board,empty_start)
for peg,over,land in Solution:
  RemovePeg(board,peg)
  RemovePeg(board,over)
  AddPeg(board,land) # board order changes!
  DrawBoard(board)
  print "Peg %X jumped over %X to land on %X\n" % (peg,over,land)
```

Output:

```
     1
    . 3
   . 5 6
  7 8 9 A
 B C D E F
Peg 4 jumped over 2 to land on 1

     1
    . 3
   4 . .
  7 8 9 A
 B C D E F
Peg 6 jumped over 5 to land on 4

     .
    . .
   4 . 6
  7 8 9 A
 B C D E F
Peg 1 jumped over 3 to land on 6
```

```
     .
    2 .
   . . 6
  . 8 9 A
 B C D E F
Peg 7 jumped over 4 to land on 2

     .
    2 .
   . 5 6
  . . 9 A
 B . D E F
Peg C jumped over 8 to land on 5

     .
    2 .
   . 5 6
  . . 9 A
 B C . . F
Peg E jumped over D to land on C

     .
    2 .
   . 5 .
  . . . A
 B C D . F
Peg 6 jumped over 9 to land on D

     .
    . .
   . . .
  . . 9 A
 B C D . F
Peg 2 jumped over 5 to land on 9

     .
    . .
   . . .
  . . 9 A
 B . . E F
Peg C jumped over D to land on E

     .
    . .
   . . 6
  . . 9 .
 B . . E .
Peg F jumped over A to land on 6

     .
    . .
   . . .
  . . . .
 B . D E .
Peg 6 jumped over 9 to land on D
```

```
     .
    . .
   . . .
  . . . .
 B C . . .
Peg E jumped over D to land on C

     .
    . .
   . . .
  . . . .
 . . D . .
Peg B jumped over C to land on D
```

Python: IBAN[edit]

Translation of: Ruby

```
import re

_country2length = dict(
    AL=28, AD=24, AT=20, AZ=28, BE=16, BH=22, BA=20, BR=29,
    BG=22, CR=21, HR=21, CY=28, CZ=24, DK=18, DO=28, EE=20,
    FO=18, FI=18, FR=27, GE=22, DE=22, GI=23, GR=27, GL=18,
    GT=28, HU=28, IS=26, IE=22, IL=23, IT=27, KZ=20, KW=30,
    LV=21, LB=28, LI=21, LT=20, LU=20, MK=19, MT=31, MR=27,
    MU=30, MC=27, MD=24, ME=22, NL=18, NO=15, PK=24, PS=29,
    PL=28, PT=25, RO=24, SM=27, SA=24, RS=22, SK=24, SI=19,
    ES=24, SE=24, CH=21, TN=24, TR=26, AE=23, GB=22, VG=24 )

def valid_iban(iban):
    # Ensure upper alphanumeric input.
    iban = iban.replace(' ','').replace('\t','')
    if not re.match(r'^[\dA-Z]+$', iban):
        return False
    # Validate country code against expected length.
    if len(iban) != _country2length[iban[:2]]:
        return False
    # Shift and convert.
    iban = iban[4:] + iban[:4]
    digits = int(''.join(str(int(ch, 36)) for ch in iban)) #BASE 36: 0..9,A
    return digits % 97 == 1

if __name__ == '__main__':
    for account in ["GB82 WEST 1234 5698 7654 32", "GB82 TEST 1234 5698 765
        print('%s validation is: %s' % (account, valid_iban(account)))
```

Output:

```
GB82 WEST 1234 5698 7654 32 validation is: True
GB82 TEST 1234 5698 7654 32 validation is: False
```

Racket[edit]

Python: Idiomatically_determine_all_the_lowercase

Python defines eleven string classes for the Unicode characters in the rang

```
classes = (str.isupper, str.islower, str.isalnum, str.isalpha, str.isdecima
           str.isdigit, str.isidentifier, str.isnumeric, str.isprintable,
           str.isspace, str.istitle)

for stringclass in classes:
    chars = ''.join(chr(i) for i in range(0x10FFFF+1) if stringclass(chr(i)
    print('\nString class %s has %i characters the first of which are:\n  %
          % (stringclass.__name__, len(chars), chars[:100]))
```

Output:

```
String class isupper has 1483 characters the first of which are:
  'ABCDEFGHIJKLMNOPQRSTUVWXYZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞĀĂĄĆĈĊČĎĐĒĔĖĘĚĜĞ

String class islower has 1934 characters the first of which are:
  'abcdefghijklmnopqrstuvwxyzªµºßàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþÿāăąćĉċčďđēĕ

String class isalnum has 102157 characters the first of which are:
  '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzª²³µ¹º¼½¾À

String class isalpha has 101013 characters the first of which are:
  'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzªµºÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐ

String class isdecimal has 460 characters the first of which are:
  '0123456789□□□□□□□□□□٠١٢٣٤٥٦٧٨٩۰۱۲۳۴۵۶۷۸۹०१२३४५६७८९০১২৩৪৫৬৭৮৯੦੧੨੩

String class isdigit has 588 characters the first of which are:
  '0123456789²³¹□□□□□□□□□□٠١٢٣٤٥٦٧٨٩۰۱۲۳۴۵۶۷۸۹०१२३४५६७८९০১২৩৪৫৬৭৮৯੦

String class isidentifier has 101218 characters the first of which are:
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyzªµºÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
```

```
String class isnumeric has 1225 characters the first of which are:
  '0123456789²³¹¼½¾□□□□□□□□□□□٠١٢٣٤٥٦٧٨٩۰۱۲۳۴۵۶۷۸۹߀߁߂߃߄߅߆߇߈߉०१२३

String class isprintable has 109958 characters the first of which are:
  ' !"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcde

String class isspace has 30 characters the first of which are:
  '\t\n\x0b\x0c\r\x1c\x1d\x1e\x1f \x85\xa0\u1680\u180e\u2000\u2001\u2002\u2

String class istitle has 1514 characters the first of which are:
  'ABCDEFGHIJKLMNOPQRSTUVWXYZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞĀĂĄĆĈĊČĎĐĒĔĖĘĚĜĞ
```

Python: Image_noise[edit]

Library: Tkinter

Library: PIL

```
import time
import random
import Tkinter
import Image, ImageTk # PIL libray

class App(object):
    def __init__(self, size, root):
        self.root = root
        self.root.title("Image Noise Test")

        self.img = Image.new("RGB", size)
        self.label = Tkinter.Label(root)
        self.label.pack()

        self.time = 0.0
        self.frames = 0
        self.size = size
        self.loop()

    def loop(self):
        self.ta = time.time()
        # 13 FPS boost. half integer idea from C#.
        rnd = random.random
        white = (255, 255, 255)
        black = (0, 0, 0)
        npixels = self.size[0] * self.size[1]
        data = [white if rnd() > 0.5 else black for i in xrange(npixels)]
        self.img.putdata(data)
        self.pimg = ImageTk.PhotoImage(self.img)
        self.label["image"] = self.pimg
        self.tb = time.time()
```

```
        self.time += (self.tb - self.ta)
        self.frames += 1

        if self.frames == 30:
            try:
                self.fps = self.frames / self.time
            except:
                self.fps = "INSTANT"
            print ("%d frames in %3.2f seconds (%s FPS)" %
                  (self.frames, self.time, self.fps))
            self.time = 0
            self.frames = 0

        self.root.after(1, self.loop)

def main():
    root = Tkinter.Tk()
    app = App((320, 240), root)
    root.mainloop()

main()
```

About 28 FPS max, Python 2.6.6.

Python: Implicit_type_conversion[edit]

Python does do some automatic conversions between different types but is st

```
from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 60
from itertools import product

casting_functions = [int, float, complex,   # Numbers
                     Fraction, Decimal,     # Numbers
                     hex, oct, bin,         # Int representations - not str
                     bool,                  # Boolean/integer Number
                     iter,                  # Iterator type
                     list, tuple, range,    # Sequence types
                     str, bytes,            # Strings, byte strings
                     bytearray,             # Mutable bytes
                     set, frozenset,        # Set, hashable set
                     dict,                  # hash mapping dictionary
                    ]

examples_of_types = [0, 42,
                     0.0 -0.0, 12.34, 56.0,
                     (0+0j), (1+2j), (1+0j), (78.9+0j), (0+1.2j),
```

```
                     Fraction(0, 1), Fraction(22, 7), Fraction(4, 2),
                     Decimal('0'),
                     Decimal('3.14159265358979323846264338327950288419716939
                     Decimal('1'), Decimal('1.5'),
                     True, False,
                     iter(()), iter([1, 2, 3]), iter({'A', 'B', 'C'}),
                     iter([[1, 2], [3, 4]]), iter((('a', 1), (2, 'b'))),
                     [], [1, 2], [[1, 2], [3, 4]],
                     (), (1, 'two', (3+0j)), (('a', 1), (2, 'b')),
                     range(0), range(3),
                     "", "A", "ABBA", "Milü",
                     b"", b"A", b"ABBA",
                     bytearray(b""), bytearray(b"A"), bytearray(b"ABBA"),
                     set(), {1, 'two', (3+0j), (4, 5, 6)},
                     frozenset(), frozenset({1, 'two', (3+0j), (4, 5, 6)}),
                     {}, {1: 'one', 'two': (2+3j), ('RC', 3): None}
                    ]
if __name__ == '__main__':
    print('Common Python types/type casting functions:')
    print('  ' + '\n  '.join(f.__name__ for f in casting_functions))
    print('\nExamples of those types:')
    print('  ' + '\n  '.join('%-26s %r' % (type(e), e) for e in examples_of_
    print('\nCasts of the examples:')
    for f, e in product(casting_functions, examples_of_types):
        try:
            ans = f(e)
        except BaseException:
            ans = 'EXCEPTION RAISED!'
        print('%-60s -> %r' % ('%s(%r)' % (f.__name__, e), ans))
```

Output:

(Elided due to size)

Python: Include_a_file[edit]

Python supports the use of execfile to allow code from arbitrary files to b

```
import mymodule
```

includes the content of mymodule.py

Names in this module can be accessed as attributes:

```
mymodule.variable
```

R[edit]

Python: Increment_a_numerical_string[edit]

Works with: Python version 2.3 through 3.4

```
next = str(int('123') + 1)
```

R[edit]

Python: Index_finite_lists_of_positive_integers[e

```
def rank(x): return int('a'.join(map(str, [1] + x)), 11)

def unrank(n):
        s = ''
        while n: s,n = "0123456789a"[n%11] + s, n//11
        return map(int, s.split('a'))[1:]

l = [1, 2, 3, 10, 100, 987654321]
print l
n = rank(l)
print n
l = unrank(n)
print l
```

Python: Infinity[edit]

This is how you get infinity:

```
>>> float('infinity')
inf
```

Note: When passing in a string to float(), values for NaN and Infinity may

The Decimal module explicitly supports +/-infinity Nan, +/-0.0, etc without

Floating-point division by 0 doesn't give you infinity, it raises an except

```
>>> 1.0 / 0.0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: float division
```

If float('infinity') doesn't work on your platform, you could use this tric

```
>>> 1e999
1.#INF
```

It works by trying to create a float bigger than the machine can handle.

R[edit]

Python: Inheritance.1[edit]

Unrevised style classes:

```
class Animal:
  pass #functions go here...

class Dog(Animal):
  pass #functions go here...

class Cat(Animal):
  pass #functions go here...

class Lab(Dog):
  pass #functions go here...

class Collie(Dog):
  pass #functions go here...
```

New style classes:

```
import time

class Animal(object):
    def __init__(self, birth=None, alive=True):
        self.birth = birth if birth else time.time()
        self.alive = alive
    def age(self):
        return time.time() - self.birth
    def kill(self):
        self.alive = False

class Dog(Animal):
    def __init__(self, bones_collected=0, **kwargs):
        self.bone_collected = bones_collected
        super(Dog, self).__init__(**kwargs)

class Cat(Animal):
    max_lives = 9
    def __init__(self, lives=max_lives, **kwargs):
        self.lives = lives
        super(Cat, self).__init__(**kwargs)
    def kill(self):
        if self.lives>0:
            self.lives -= 1
            if self.lives == 0:
                super(Cat, self).kill()
        else:
            raise ValueError
        return self

class Labrador(Dog):
    def __init__(self, guide_dog=False, **kwargs):
        self.guide_dog=False
        super(Labrador, self).__init__(**kwargs)

class Collie(Dog):
    def __init__(self, sheep_dog=False, **kwargs):
        self.sheep_dog=False
        super(Collie, self).__init__(**kwargs)

lassie = Collie()
felix = Cat()
felix.kill().kill().kill()
mr_winkle = Dog()
buddy = Labrador()
buddy.kill()
print "Felix has",felix.lives, "lives, ","Buddy is %salive!"%("" if buddy.a
```

Output:

```
Felix has 6 lives,  Buddy is not alive!
```

R[edit]

Python: Input_loop[edit]

Python file objects can be iterated like lists:

```
my_file = open(filename, 'r')
try:
    for line in my_file:
        pass # process line, includes newline
finally:
    my_file.close()
```

One can open a new stream for read and have it automatically close when don

```
from __future__ import with_statement

with open(filename, 'r') as f:
    for line in f:
        pass # process line, includes newline
```

You can also get lines manually from a file:

```
line = my_file.readline() # returns a line from the file
lines = my_file.readlines() # returns a list of the rest of the lines from
```

This does not mix well with the iteration, however.

Python: Insertion_sort[edit]

```
def insertion_sort(l):
    for i in xrange(1, len(l)):
        j = i-1
        key = l[i]
        while (l[j] > key) and (j >= 0):
           l[j+1] = l[j]
           j -= 1
        l[j+1] = key
```

Insertion sort with binary search[edit]

```
def insertion_sort_bin(seq):
    for i in range(1, len(seq)):
        key = seq[i]
        # invariant: ``seq[:i]`` is sorted
        # find the least `low' such that ``seq[low]`` is not less then `key
        #   Binary search in sorted sequence ``seq[low:up]``:
        low, up = 0, i
        while up > low:
            middle = (low + up) // 2
            if seq[middle] < key:
                low = middle + 1
            else:
                up = middle
        # insert key at position ``low``
        seq[:] = seq[:low] + [key] + seq[low:i] + seq[i + 1:]
```

This is also built-in to the standard library:

```
import bisect
def insertion_sort_bin(seq):
    for i in range(1, len(seq)):
        bisect.insort(seq, seq.pop(i), 0, i)
```

R[edit]

Python: Integer_comparison[edit]

```
#!/usr/bin/env python
```

```
a = input('Enter value of a: ')
b = input('Enter value of b: ')

if a < b:
    print 'a is less than b'
elif a > b:
    print 'a is greater than b'
elif a == b:
    print 'a is equal to b'
```

(Note: in Python3 *input()* will become *int(input())*)

An alternative implementation could use a Python dictionary to house a smal

Works with: Python version 2.x only, not 3.x

```
#!/usr/bin/env python
import sys
try:
   a = input('Enter value of a: ')
   b = input('Enter value of b: ')
except (ValueError, EnvironmentError), err:
   print sys.stderr, "Erroneous input:", err
   sys.exit(1)

dispatch = {
    -1: 'is less than',
     0: 'is equal to',
     1: 'is greater than'
     }
 print a, dispatch[cmp(a,b)], b
```

In this case the use of a dispatch table is silly. However, more generally

Python: Integer_literals[edit]

Works with: Python version 3.0

Python 3.0 brought in the binary literal and uses 0o or 0O exclusively for

```
>>> # Bin(leading 0b or 0B), Oct(leading 0o or 0O), Dec, Hex(leading 0x or
>>> 0b1011010111 == 0o1327 == 727 == 0x2d7
True
>>>
```

Works with: Python version 2.6

Python 2.6 has the binary and new octal formats of 3.0, as well as keeping t

```
>>> # Bin(leading 0b or 0B), Oct(leading 0o or 0O, or just 0), Dec, Hex(lea
>>> 0b1011010111 == 0o1327 == 01327 == 727 == 0x2d7
True
>>>
```

Python: Integer_sequence[edit]

```
i=1
while i:
    print(i)
    i += 1
```

Or, alternatively:

```
from itertools import count

for i in count():
    print(i)
```

Pythons integers are of arbitrary large precision and so programs would pro

Python: Interactive_programming[edit]

Start the interpreter by typing python at the command line (or select it fr

```
python
Python 2.6.1 (r261:67517, Dec  4 2008, 16:51:00) [MSC v.1500 32 bit (Intel)
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def f(string1, string2, separator):
        return separator.join([string1, '', string2])
```

```
>>> f('Rosetta', 'Code', ':')
'Rosetta::Code'
>>>
```

Python: Interrupts[edit]

Simple version

```
import time

def counter():
    n = 0
    t1 = time.time()
    while True:
        try:
            time.sleep(0.5)
            n += 1
            print n
        except KeyboardInterrupt, e:
            print 'Program has run for %5.3f seconds.' % (time.time() - t1)
            break

counter()
```

The following example should work on all platforms.

```
import time

def intrptWIN():
   procDone = False
   n = 0

   while not procDone:
      try:
         time.sleep(0.5)
         n += 1
         print n
      except KeyboardInterrupt, e:
         procDone = True

t1 = time.time()
intrptWIN()
tdelt = time.time() - t1
print 'Program has run for %5.3f seconds.' % tdelt
```

There is a signal module in the standard distribution
that accomodates the UNIX type signal mechanism.
However the pause() mechanism is not implemented on Windows versions.

```
import signal, time, threading
done = False
n = 0

def counter():
   global n, timer
   n += 1
   print n
   timer = threading.Timer(0.5, counter)
   timer.start()

def sigIntHandler(signum, frame):
   global done
   timer.cancel()
   done = True

def intrptUNIX():
   global timer
   signal.signal(signal.SIGINT, sigIntHandler)

   timer = threading.Timer(0.5, counter)
   timer.start()
   while not done:
      signal.pause()

t1 = time.time()
intrptUNIX()
tdelt = time.time() - t1
print 'Program has run for %5.3f seconds.' % tdelt
```

How about this one? It should work on all platforms;
and it does show how to install a signal handler:

```
import time, signal

class WeAreDoneException(Exception):
    pass

def sigIntHandler(signum, frame):
    signal.signal(signal.SIGINT, signal.SIG_DFL) # resets to default handle
    raise WeAreDoneException

t1 = time.time()

try:
    signal.signal(signal.SIGINT, sigIntHandler)
```

```
    n = 0
    while True:
        time.sleep(0.5)
        n += 1
        print n
except WeAreDoneException:
    pass

tdelt = time.time() - t1
print 'Program has run for %5.3f seconds.' % tdelt
```

Python: Introspection[edit]

```
# Checking for system version
 import sys
 major, minor, bugfix = sys.version_info[:3]
 if major < 2:
     sys.exit('Python 2 is required')


 def defined(name): # LBYL (Look Before You Leap)
     return name in globals() or name in locals() or name in vars(__builtin

 def defined2(name): # EAFP (Easier to Ask Forgiveness than Permission)
     try:
          eval(name)
          return True
     except NameError:
          return False

 if defined('bloop') and defined('abs') and callable(abs):
     print abs(bloop)

 if defined2('bloop') and defined2('abs') and callable(abs):
     print abs(bloop)
```

You can combine both tests, (But loose sight of which variable in missing/n

```
try:
    print abs(bloop)
except (NameError, TypeError):
    print "Something's missing"
```

Here is one way to print the sum of all the global integer variables:

```
def sum_of_global_int_vars():
    variables = vars(__builtins__).copy()
    variables.update(globals())
    print sum(v for v in variables.itervalues() if type(v) == int)

sum_of_global_int_vars()
```

R[edit]

Python: Inverted_index[edit]

Simple inverted index[edit]

First the simple inverted index from here together with an implementation o

```
'''
This implements: http://en.wikipedia.org/wiki/Inverted_index of 28/07/10
'''

from pprint import pprint as pp
from glob import glob
try: reduce
except: from functools import reduce
try:    raw_input
except: raw_input = input


def parsetexts(fileglob='InvertedIndex/T*.txt'):
    texts, words = {}, set()
    for txtfile in glob(fileglob):
        with open(txtfile, 'r') as f:
            txt = f.read().split()
            words |= set(txt)
            texts[txtfile.split('\\')[-1]] = txt
    return texts, words

def termsearch(terms): # Searches simple inverted index
    return reduce(set.intersection,
                  (invindex[term] for term in terms),
                  set(texts.keys()))

texts, words = parsetexts()
print('\nTexts')
pp(texts)
```

```
print('\nWords')
pp(sorted(words))

invindex = {word:set(txt
                        for txt, wrds in texts.items() if word in wrds)
            for word in words}
print('\nInverted Index')
pp({k:sorted(v) for k,v in invindex.items()})

terms = ["what", "is", "it"]
print('\nTerm Search for: ' + repr(terms))
pp(sorted(termsearch(terms)))
```

Sample Output

Python: Inverted_syntax[edit]

```
x = truevalue if condition else falsevalue
```

Qi[edit]

Python: JSON[edit]

Works with: Python version 2.6+
Works with: Python version 3.0+

```
>>> import json
>>> data = json.loads('{ "foo": 1, "bar": [10, "apples"] }')
>>> sample = { "blue": [1,2], "ocean": "water" }
>>> json_string = json.dumps(sample)
>>> json_string
'{"blue": [1, 2], "ocean": "water"}'
>>> sample
{'blue': [1, 2], 'ocean': 'water'}
>>> data
{'foo': 1, 'bar': [10, 'apples']}
```

```
Because most of JSON is valid Python syntax (except "true", "false", and "n
```

```
>>> true = True; false = False; null = None
>>> data = eval('{ "foo": 1, "bar": [10, "apples"] }')
>>> data
{'foo': 1, 'bar': [10, 'apples']}
```

Python: Jensen's_Device[edit]

```
class Ref(object):
    def __init__(self, value=None):
        self.value = value

def harmonic_sum(i, lo, hi, term):
    # term is passed by-name, and so is i
    temp = 0
    i.value = lo
    while i.value <= hi:  # Python "for" loop creates a distinct which
        temp += term() # would not be shared with the passed "i"
        i.value += 1   # Here the actual passed "i" is incremented.
    return temp

i = Ref()

# note the correspondence between the mathematical notation and the
# call to sum it's almost as good as sum(1/i for i in range(1,101))
print harmonic_sum(i, 1, 100, lambda: 1.0/i.value)
```

```
Output: 5.18737751764
```

Python: JortSort[edit]

```
>>> def jortSort(myarray):
        return list(myarray) == sorted(myarray)
>>> for data in [(1,2,4,3), (14,6,8), ['a', 'c'], ['s', 'u', 'x'], 'CVGH',
        print('jortSort(%r) is %s' % (data, jortSort(data)))
jortSort((1, 2, 4, 3)) is False
jortSort((14, 6, 8)) is False
jortSort(['a', 'c']) is True
jortSort(['s', 'u', 'x']) is True
jortSort('CVGH') is False
jortSort('PQRST') is True
>>>
```

Translation of: JavaScript

```
def jortSort(array):

  # sort the array
  originalArray = list(array)
  array.sort()

  # compare to see if it was originally sorted
  for i in range(len(originalArray)):
    if originalArray[i] != array[i]:
      return False

  return True
```

Python: Josephus_problem[edit]

```
>>> def j(n, k):
        p, i, seq = list(range(n)), 0, []
        while p:
                i = (i+k-1) % len(p)
                seq.append(p.pop(i))
        return 'Prisoner killing order: %s.\nSurvivor: %i' % (', '.join(str

>>> print(j(5, 2))
Prisoner killing order: 1, 3, 0, 4.
Survivor: 2
>>> print(j(41, 3))
Prisoner killing order: 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 0,
Survivor: 30
>>>
```

Faster way to solve in python, it does not show the killing order.

Python: Joystick_position[edit]

Library: Pygame

```
import sys
import pygame

pygame.init()
```

```
# Create a clock (for framerating)
clk = pygame.time.Clock()

# Grab joystick 0
if pygame.joystick.get_count() == 0:
    raise IOError("No joystick detected")
joy = pygame.joystick.Joystick(0)
joy.init()

# Create display
size = width, height = 600, 600
screen = pygame.display.set_mode(size)
pygame.display.set_caption("Joystick Tester")

# Frame XHair zone
frameRect = pygame.Rect((45, 45), (510, 510))

# Generate crosshair
crosshair = pygame.surface.Surface((10, 10))
crosshair.fill(pygame.Color("magenta"))
pygame.draw.circle(crosshair, pygame.Color("blue"), (5,5), 5, 0)
crosshair.set_colorkey(pygame.Color("magenta"), pygame.RLEACCEL)
crosshair = crosshair.convert()

# Generate button surfaces
writer = pygame.font.Font(pygame.font.get_default_font(), 15)
buttons = {}
for b in range(joy.get_numbuttons()):
    buttons[b] = [
        writer.render(
            hex(b)[2:].upper(),
            1,
            pygame.Color("red"),
            pygame.Color("black")
        ).convert(),
        # Get co-ords: ((width*slot)+offset, offset). Offsets chosen
        #                                             to match frames.
        ((15*b)+45, 560)
    ]

while True:
    # Pump and check the events queue
    pygame.event.pump()
    for events in pygame.event.get():
        if events.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    # Black the screen
    screen.fill(pygame.Color("black"))

    # Get joystick axes
    x = joy.get_axis(0)
    y = joy.get_axis(1)

    # Blit to the needed coords:
```

```
    # x*amplitude+(centre offset (window size/2))-(xhair offset (xh size/2)
    screen.blit(crosshair, ((x*250)+300-5, (y*250)+300-5))
    pygame.draw.rect(screen, pygame.Color("red"), frameRect, 1)

    # Get and display the joystick buttons
    for b in range(joy.get_numbuttons()):
        if joy.get_button(b):
            screen.blit(buttons[b][0], buttons[b][1])

    # Write the display
    pygame.display.flip()
    clk.tick(40) # Limit to <=40 FPS
```

Tcl[edit]

Python: Jump_anywhere[edit]

Python has both exceptions and generators but no unstructured goto ability.

The "goto" module was an April Fool's joke, published on 1st April 2004. Ye

Python: K-d_tree[edit]

Translation of: D

```
from random import seed, random
from time import clock
from operator import itemgetter
from collections import namedtuple
from math import sqrt
from copy import deepcopy


def sqd(p1, p2):
    return sum((c1 - c2) ** 2 for c1, c2 in zip(p1, p2))


class KdNode(object):
    __slots__ = ("dom_elt", "split", "left", "right")

    def __init__(self, dom_elt, split, left, right):
        self.dom_elt = dom_elt
        self.split = split
        self.left = left
        self.right = right
```

```
class Orthotope(object):
    __slots__ = ("min", "max")

    def __init__(self, mi, ma):
        self.min, self.max = mi, ma


class KdTree(object):
    __slots__ = ("n", "bounds")

    def __init__(self, pts, bounds):
        def nk2(split, exset):
            if not exset:
                return None
            exset.sort(key=itemgetter(split))
            m = len(exset) // 2
            d = exset[m]
            while m + 1 < len(exset) and exset[m + 1][split] == d[split]:
                m += 1

            s2 = (split + 1) % len(d)  # cycle coordinates
            return KdNode(d, split, nk2(s2, exset[:m]),
                                    nk2(s2, exset[m + 1:]))
        self.n = nk2(0, pts)
        self.bounds = bounds

T3 = namedtuple("T3", "nearest dist_sqd nodes_visited")


def find_nearest(k, t, p):
    def nn(kd, target, hr, max_dist_sqd):
        if kd is None:
            return T3([0.0] * k, float("inf"), 0)

        nodes_visited = 1
        s = kd.split
        pivot = kd.dom_elt
        left_hr = deepcopy(hr)
        right_hr = deepcopy(hr)
        left_hr.max[s] = pivot[s]
        right_hr.min[s] = pivot[s]

        if target[s] <= pivot[s]:
            nearer_kd, nearer_hr = kd.left, left_hr
            further_kd, further_hr = kd.right, right_hr
        else:
            nearer_kd, nearer_hr = kd.right, right_hr
            further_kd, further_hr = kd.left, left_hr

        n1 = nn(nearer_kd, target, nearer_hr, max_dist_sqd)
        nearest = n1.nearest
        dist_sqd = n1.dist_sqd
        nodes_visited += n1.nodes_visited

        if dist_sqd < max_dist_sqd:
```

```
            max_dist_sqd = dist_sqd
        d = (pivot[s] - target[s]) ** 2
        if d > max_dist_sqd:
            return T3(nearest, dist_sqd, nodes_visited)
        d = sqd(pivot, target)
        if d < dist_sqd:
            nearest = pivot
            dist_sqd = d
            max_dist_sqd = dist_sqd

        n2 = nn(further_kd, target, further_hr, max_dist_sqd)
        nodes_visited += n2.nodes_visited
        if n2.dist_sqd < dist_sqd:
            nearest = n2.nearest
            dist_sqd = n2.dist_sqd

        return T3(nearest, dist_sqd, nodes_visited)

    return nn(t.n, p, t.bounds, float("inf"))


def show_nearest(k, heading, kd, p):
    print(heading + ":")
    print("Point:            ", p)
    n = find_nearest(k, kd, p)
    print("Nearest neighbor:", n.nearest)
    print("Distance:         ", sqrt(n.dist_sqd))
    print("Nodes visited:    ", n.nodes_visited, "\n")


def random_point(k):
    return [random() for _ in range(k)]


def random_points(k, n):
    return [random_point(k) for _ in range(n)]

if __name__ == "__main__":
    seed(1)
    P = lambda *coords: list(coords)
    kd1 = KdTree([P(2, 3), P(5, 4), P(9, 6), P(4, 7), P(8, 1), P(7, 2)],
                  Orthotope(P(0, 0), P(10, 10)))
    show_nearest(2, "Wikipedia example data", kd1, P(9, 2))

    N = 400000
    t0 = clock()
    kd2 = KdTree(random_points(3, N), Orthotope(P(0, 0, 0), P(1, 1, 1)))
    t1 = clock()
    text = lambda *parts: "".join(map(str, parts))
    show_nearest(2, text("k-d tree with ", N,
                         " random 3D points (generation time: ",
                         t1-t0, "s)"),
                 kd2, random_point(3))
```

Output:

```
Wikipedia example data:
Point:            [9, 2]
Nearest neighbor: [8, 1]
Distance:         1.41421356237
Nodes visited:    3

k-d tree with 400000 random 3D points (generation time: 14.8755565302s):
Point:            [0.066694022911324868, 0.13692213852082813, 0.94939167224
Nearest neighbor: [0.067027753280507252, 0.14407354836507069, 0.94543775920
Distance:         0.00817847583914
Nodes visited:    33
```

Python: K-means++_clustering[edit]

Translation of: D

```
from math import pi, sin, cos
from collections import namedtuple
from random import random, choice
from copy import copy

try:
    import psyco
    psyco.full()
except ImportError:
    pass


FLOAT_MAX = 1e100


class Point:
    __slots__ = ["x", "y", "group"]
    def __init__(self, x=0.0, y=0.0, group=0):
        self.x, self.y, self.group = x, y, group


def generate_points(npoints, radius):
    points = [Point() for _ in xrange(npoints)]

    # note: this is not a uniform 2-d distribution
    for p in points:
        r = random() * radius
        ang = random() * 2 * pi
        p.x = r * cos(ang)
        p.y = r * sin(ang)
```

```
    return points


def nearest_cluster_center(point, cluster_centers):
    """Distance and index of the closest cluster center"""
    def sqr_distance_2D(a, b):
        return (a.x - b.x) ** 2  +  (a.y - b.y) ** 2

    min_index = point.group
    min_dist = FLOAT_MAX

    for i, cc in enumerate(cluster_centers):
        d = sqr_distance_2D(cc, point)
        if min_dist > d:
            min_dist = d
            min_index = i

    return (min_index, min_dist)


def kpp(points, cluster_centers):
    cluster_centers[0] = copy(choice(points))
    d = [0.0 for _ in xrange(len(points))]

    for i in xrange(1, len(cluster_centers)):
        sum = 0
        for j, p in enumerate(points):
            d[j] = nearest_cluster_center(p, cluster_centers[:i])[1]
            sum += d[j]

        sum *= random()

        for j, di in enumerate(d):
            sum -= di
            if sum > 0:
                continue
            cluster_centers[i] = copy(points[j])
            break

    for p in points:
        p.group = nearest_cluster_center(p, cluster_centers)[0]


def lloyd(points, nclusters):
    cluster_centers = [Point() for _ in xrange(nclusters)]

    # call k++ init
    kpp(points, cluster_centers)

    lenpts10 = len(points) >> 10

    changed = 0
    while True:
        # group element for centroids are used as counters
        for cc in cluster_centers:
            cc.x = 0
```

```
            cc.y = 0
            cc.group = 0

        for p in points:
            cluster_centers[p.group].group += 1
            cluster_centers[p.group].x += p.x
            cluster_centers[p.group].y += p.y

        for cc in cluster_centers:
            cc.x /= cc.group
            cc.y /= cc.group

        # find closest centroid of each PointPtr
        changed = 0
        for p in points:
            min_i = nearest_cluster_center(p, cluster_centers)[0]
            if min_i != p.group:
                changed += 1
                p.group = min_i

        # stop when 99.9% of points are good
        if changed <= lenpts10:
            break

    for i, cc in enumerate(cluster_centers):
        cc.group = i

    return cluster_centers


def print_eps(points, cluster_centers, W=400, H=400):
    Color = namedtuple("Color", "r g b");

    colors = []
    for i in xrange(len(cluster_centers)):
        colors.append(Color((3 * (i + 1) % 11) / 11.0,
                            (7 * i % 11) / 11.0,
                            (9 * i % 11) / 11.0))

    max_x = max_y = -FLOAT_MAX
    min_x = min_y = FLOAT_MAX

    for p in points:
        if max_x < p.x: max_x = p.x
        if min_x > p.x: min_x = p.x
        if max_y < p.y: max_y = p.y
        if min_y > p.y: min_y = p.y

    scale = min(W / (max_x - min_x),
                H / (max_y - min_y))
    cx = (max_x + min_x) / 2
    cy = (max_y + min_y) / 2

    print "%%!PS-Adobe-3.0\n%%%%BoundingBox: -5 -5 %d %d" % (W + 10, H + 10

    print ("/l {rlineto} def /m {rmoveto} def\n" +
```

```
            "/c { .25 sub exch .25 sub exch .5 0 360 arc fill } def\n" +
            "/s { moveto -2 0 m 2 2 l 2 -2 l -2 -2 l closepath " +
            "   gsave 1 setgray fill grestore gsave 3 setlinewidth" +
            " 1 setgray stroke grestore 0 setgray stroke }def")

    for i, cc in enumerate(cluster_centers):
        print ("%g %g %g setrgbcolor" %
               (colors[i].r, colors[i].g, colors[i].b))

        for p in points:
            if p.group != i:
                continue
            print ("%.3f %.3f c" % ((p.x - cx) * scale + W / 2,
                                    (p.y - cy) * scale + H / 2))

        print ("\n0 setgray %g %g s" % ((cc.x - cx) * scale + W / 2,
                                        (cc.y - cy) * scale + H / 2))

    print "\n%%%%EOF"


def main():
    npoints = 30000
    k = 7 # # clusters

    points = generate_points(npoints, 10)
    cluster_centers = lloyd(points, k)
    print_eps(points, cluster_centers)


main()
```

Racket[edit]

Python: Kaprekar_numbers[edit]

Splitting strings in a loop[edit]

(Swap the commented return statement to return the split information).

```
>>> def k(n):
        n2 = str(n**2)
        for i in range(len(n2)):
                a, b = int(n2[:i] or 0), int(n2[i:])
                if b and a + b == n:
                        return n
                        #return (n, (n2[:i], n2[i:]))


>>> [x for x in range(1,10000) if k(x)]
[1, 9, 45, 55, 99, 297, 703, 999, 2223, 2728, 4879, 4950, 5050, 5292, 7272,
>>> len([x for x in range(1,1000000) if k(x)])
54
>>>
```

A stronger code that gives a list of Kaprekar numbers within a range in a g
The range must be given as a decimal number.

```
def encode(n, base):
    result = ""
    while n:
        n, d = divmod(n, base)
        if d < 10:
            result += str(d)
        else:
            result += chr(d - 10 + ord("a"))
    return result[::-1]
def Kaprekar(n, base):
    if n == '1':
        return True
    sq = encode((int(n, base)**2), base)
    for i in range(1,len(sq)):
        if (int(sq[:i], base) + int(sq[i:], base) == int(n, base)) and (int
            return True
    return False
def Find(m, n, base):
    return [encode(i, base) for i in range(m,n+1) if Kaprekar(encode(i, bas

m = int(raw_input('Where to start?\n'))
n = int(raw_input('Where to stop?\n'))
base = int(raw_input('Enter base:'))
KNumbers = Find(m, n, base)
for i in KNumbers:
    print i
print 'The number of Kaprekar Numbers found are',
print len(KNumbers)
raw_input()
```

Using Casting Out Nines Generator[edit]

See: http://rosettacode.org/wiki/Casting_out_nines#Python for explanation a

```
Base = 10
N = 6
Paddy_cnt = 1
for n in range(N):
  for V in CastOut(Base,Start=Base**n,End=Base**(n+1)):
    for B in range(n+1,n*2+2):
      x,y = divmod(V*V,Base**B)
      if V == x+y and 0<y:
        print('{1}: {0}'.format(V, Paddy_cnt))
        Paddy_cnt += 1
        break
```

Produces:

```
1: 1
2: 9
3: 45
4: 55
5: 99
6: 297
7: 703
8: 999
9: 2223
10: 2728
11: 4879
12: 4950
13: 5050
14: 5292
15: 7272
16: 7777
17: 9999
18: 17344
19: 22222
20: 38962
21: 77778
22: 82656
23: 95121
24: 99999
25: 142857
26: 148149
27: 181819
28: 187110
29: 208495
30: 318682
31: 329967
```

```
32: 351352
33: 356643
34: 390313
35: 461539
36: 466830
37: 499500
38: 500500
39: 533170
40: 538461
41: 609687
42: 627615
43: 643357
44: 648648
45: 670033
46: 681318
47: 791505
48: 812890
49: 818181
50: 851851
51: 857143
52: 961038
53: 994708
54: 999999
```

Other bases may be used e.g.:

```
Base = 16
N = 4
Paddy_cnt = 1
for V in CastOut(Base,Start=1,End=Base**N):
  for B in range(1,N*2-1):
    x,y = divmod(V*V,Base**B)
    if V == x+y and 0<y:
      print('{1}: {0:x}'.format(V, Paddy_cnt))
      Paddy_cnt += 1
      break
```

Produces:

```
1: 1
2: 6
3: a
4: f
5: 33
6: 55
7: 5b
8: 78
```

```
9: 88
10: ab
11: cd
12: ff
13: 15f
14: 334
15: 38e
16: 492
17: 4ed
18: 7e0
19: 820
20: b13
21: b6e
22: c72
23: ccc
24: ea1
25: fa5
26: fff
27: 191a
28: 2a2b
29: 3c3c
30: 4444
31: 5556
32: 6667
33: 7f80
34: 8080
35: 9999
36: aaaa
37: bbbc
38: c3c4
39: d5d5
40: e6e6
41: ffff
```

Python: Keyboard_macros[edit]

Works on Unix platforms.

```
#!/usr/bin/env python
import curses

def print_message():
    stdscr.addstr('This is the message.\n')

stdscr = curses.initscr()
curses.noecho()
curses.cbreak()
stdscr.keypad(1)
```

```
stdscr.addstr('CTRL+P for message or q to quit.\n')
while True:
    c = stdscr.getch()
    if c == 16: print_message()
    elif c == ord('q'): break

curses.nocbreak()
stdscr.keypad(0)
curses.echo()
curses.endwin()
```

Python: Knight's_Tour[edit]

Knights tour using Warnsdorffs algorithm

```
import copy

boardsize=6
_kmoves = ((2,1), (1,2), (-1,2), (-2,1), (-2,-1), (-1,-2), (1,-2), (2,-1))


def chess2index(chess, boardsize=boardsize):
    'Convert Algebraic chess notation to internal index format'
    chess = chess.strip().lower()
    x = ord(chess[0]) - ord('a')
    y = boardsize - int(chess[1:])
    return (x, y)

def boardstring(board, boardsize=boardsize):
    r = range(boardsize)
    lines = ''
    for y in r:
        lines += '\n' + ','.join('%2i' % board[(x,y)] if board[(x,y)] else
                                 for x in r)
    return lines

def knightmoves(board, P, boardsize=boardsize):
    Px, Py = P
    kmoves = set((Px+x, Py+y) for x,y in _kmoves)
    kmoves = set( (x,y)
                  for x,y in kmoves
                  if 0 <= x < boardsize
                     and 0 <= y < boardsize
                     and not board[(x,y)] )
    return kmoves

def accessibility(board, P, boardsize=boardsize):
    access = []
```

```
    brd = copy.deepcopy(board)
    for pos in knightmoves(board, P, boardsize=boardsize):
        brd[pos] = -1
        access.append( (len(knightmoves(brd, pos, boardsize=boardsize)), po
        brd[pos] = 0
    return access

def knights_tour(start, boardsize=boardsize, _debug=False):
    board = {(x,y):0 for x in range(boardsize) for y in range(boardsize)}
    move = 1
    P = chess2index(start, boardsize)
    board[P] = move
    move += 1
    if _debug:
        print(boardstring(board, boardsize=boardsize))
    while move <= len(board):
        P = min(accessibility(board, P, boardsize))[1]
        board[P] = move
        move += 1
        if _debug:
            print(boardstring(board, boardsize=boardsize))
            input('\n%2i next: ' % move)
    return board

if __name__ == '__main__':
    while 1:
        boardsize = int(input('\nboardsize: '))
        if boardsize < 5:
            continue
        start = input('Start position: ')
        board = knights_tour(start, boardsize)
        print(boardstring(board, boardsize=boardsize))
```

Sample runs

```
boardsize: 5
Start position: c3

19,12,17, 6,21
 2, 7,20,11,16
13,18, 1,22, 5
 8, 3,24,15,10
25,14, 9, 4,23

boardsize: 8
Start position: h8

38,41,18, 3,22,27,16, 1
19, 4,39,42,17, 2,23,26
40,37,54,21,52,25,28,15
 5,20,43,56,59,30,51,24
36,55,58,53,44,63,14,29
```

```
 9, 6,45,62,57,60,31,50
46,35, 8,11,48,33,64,13
 7,10,47,34,61,12,49,32

boardsize: 10
Start position: e6

29, 4,57,24,73, 6,95,10,75, 8
58,23,28, 5,94,25,74, 7,100,11
 3,30,65,56,27,72,99,96, 9,76
22,59, 2,63,68,93,26,81,12,97
31,64,55,66, 1,82,71,98,77,80
54,21,60,69,62,67,92,79,88,13
49,32,53,46,83,70,87,42,91,78
20,35,48,61,52,45,84,89,14,41
33,50,37,18,47,86,39,16,43,90
36,19,34,51,38,17,44,85,40,15

boardsize: 200
Start position: a1

510,499,502,101,508,515,504,103,506,5021 ... 195,8550,6691,6712,197,6704,20
501,100,509,514,503,102,507,5020,5005,10 ... 690,6713,196,8553,6692,6695,19
498,511,500,4989,516,5019,5004,505,5022, ... ,30180,8559,6694,6711,8554,670
99,518,513,4992,5003,4990,5017,5044,5033 ... 30205,8552,30181,8558,6693,670
512,497,4988,517,5018,5001,5034,5011,504 ... 182,30201,30204,8555,6710,8557
519,98,4993,5002,4991,5016,5043,5052,505 ... 03,30546,30183,30200,30185,670
496,4987,520,5015,5000,5035,5012,5047,51 ... 4,30213,30202,31455,8556,6709,
97,522,4999,4994,5013,5042,5051,5060,505 ... 7,31456,31329,30184,30199,3019
4986,495,5014,521,5036,4997,5048,5101,50 ... 1327,31454,30195,31472,30187,3
523,96,4995,4998,5041,5074,5061,5050,507 ... ,31330,31471,31328,31453,30196

...

404,731,704,947,958,1013,966,1041,1078,1 ... 9969,39992,39987,39996,39867,3
 5,706,735,960,955,972,957,1060,1025,106 ... ,39978,39939,39976,39861,39990
724,403,730,705,946,967,1012,971,1040,10 ... 9975,39972,39991,39868,39863,3
707, 4,723,736,729,956,973,996,1061,1026 ... ,39850,39869,39862,39973,39852
402,725,708,943,968,945,970,1011,978,997 ... 6567,39974,39851,39864,36571,3
 3,722,737,728,741,942,977,974,995,1010, ... ,39800,39849,36570,39853,36574
720,401,726,709,944,969,742,941,980,975, ... ,14091,36568,36575,14084,14089
711, 2,721,738,727,740,715,976,745,940,9 ... 65,36576,14083,14090,36569,844
400,719,710,713,398,717,746,743,396,981, ... ,849,304,14081,840,847,302,140
 1,712,399,718,739,714,397,716,747,744,3 ... 4078,839,848,303,14082,841,846
```

The 200x200 example warmed my study in its computation but did return a tou

P.S. There is a slight deviation to a strict interpretation of Warnsdorffs

Python: Knuth's_power_tree[edit]

```
from __future__ import print_function

# remember the tree generation state and expand on demand
def path(n, p = {1:0}, lvl=[[1]]):
        if not n: return []
        while n not in p:
                q = []
                for x,y in ((x, x+y) for x in lvl[0] for y in path(x) if no
                        p[y] = x
                        q.append(y)
                lvl[0] = q

        return path(p[n]) + [n]

def tree_pow(x, n):
    r, p = {0:1, 1:x}, 0
    for i in path(n):
        r[i] = r[i-p] * r[p]
        p = i
    return r[n]

def show_pow(x, n):
    fmt = "%d: %s\n" + ["%g^%d = %f", "%d^%d = %d"][x==int(x)] + "\n"
    print(fmt % (n, repr(path(n)), x, n, tree_pow(x, n)))

for x in range(18): show_pow(2, x)
show_pow(3, 191)
show_pow(1.1, 81)
```

Output:

```
0: []
2^0 = 1

1: [1]
2^1 = 2

2: [1, 2]
2^2 = 4

<... snipped ...>

17: [1, 2, 4, 8, 16, 17]
2^17 = 131072

191: [1, 2, 3, 5, 7, 14, 19, 38, 57, 95, 190, 191]
3^191 = 1349458867428109380372815739652388491740250229403010191406670536702

81: [1, 2, 3, 5, 10, 20, 40, 41, 81]
```

1.1^81 = 2253.240236

Python: Knuth_Shuffle[edit]

Python's standard library function random.shuffle uses this algorithm and s
The function below is very similar:

```
from random import randrange

def knuth_shuffle(x):
    for i in range(len(x)-1, 0, -1):
        j = randrange(i + 1)
        x[i], x[j] = x[j], x[i]

x = list(range(10))
knuth_shuffle(x)
print("shuffled:", x)
```

Output:

```
shuffled: [5, 1, 6, 0, 8, 4, 2, 3, 9, 7]
```

R[edit]

Python: Knuth_shuffle[edit]

Python's standard library function random.shuffle uses this algorithm and s
The function below is very similar:

```
from random import randrange

def knuth_shuffle(x):
    for i in range(len(x)-1, 0, -1):
        j = randrange(i + 1)
        x[i], x[j] = x[j], x[i]
```

```
x = list(range(10))
knuth_shuffle(x)
print("shuffled:", x)
```

Output:

```
shuffled: [5, 1, 6, 0, 8, 4, 2, 3, 9, 7]
```

Python: LIFO[edit]

Works with: Python version 2.5

The faster and Pythonic way is using a deque (available from 2.4).
A regular list is a little slower.

```
from collections import deque
stack = deque()
stack.append(value) # pushing
value = stack.pop()
not stack # is empty?
```

If you need to expose your stack to the world, you may want to create a sim

```
from collections import deque

class Stack:
    def __init__(self):
        self._items = deque()
    def append(self, item):
        self._items.append(item)
    def pop(self):
        return self._items.pop()
    def __nonzero__(self):
        return bool(self._items)
```

Here is a stack implemented as linked list - with the same list interface.

```
class Stack:
```

```
    def __init__(self):
        self._first = None
    def __nonzero__(self):
        return self._first is not None
    def append(self, value):
        self._first = (value, self._first)
    def pop(self):
        if self._first is None:
            raise IndexError, "pop from empty stack"
        value, self._first = self._first
        return value
```

Notes:

Using list interface - append, __nonzero__ make it easier to use, cleanup t
For example, instead of:

```
while not stack.empty():
```

You can write:

```
while stack:
```

Quick testing show that deque is about 5 times faster then the wrapper link

Python: LU_decomposition[edit]

Translation of: D

```
from pprint import pprint

def matrixMul(A, B):
    TB = zip(*B)
    return [[sum(ea*eb for ea,eb in zip(a,b)) for b in TB] for a in A]

def pivotize(m):
    """Creates the pivoting matrix for m."""
    n = len(m)
    ID = [[float(i == j) for i in xrange(n)] for j in xrange(n)]
    for j in xrange(n):
        row = max(xrange(j, n), key=lambda i: abs(m[i][j]))
        if j != row:
            ID[j], ID[row] = ID[row], ID[j]
```

```
    return ID

def lu(A):
    """Decomposes a nxn matrix A by PA=LU and returns L, U and P."""
    n = len(A)
    L = [[0.0] * n for i in xrange(n)]
    U = [[0.0] * n for i in xrange(n)]
    P = pivotize(A)
    A2 = matrixMul(P, A)
    for j in xrange(n):
        L[j][j] = 1.0
        for i in xrange(j+1):
            s1 = sum(U[k][j] * L[i][k] for k in xrange(i))
            U[i][j] = A2[i][j] - s1
        for i in xrange(j, n):
            s2 = sum(U[k][j] * L[i][k] for k in xrange(j))
            L[i][j] = (A2[i][j] - s2) / U[j][j]
    return (L, U, P)

a = [[1, 3, 5], [2, 4, 7], [1, 1, 0]]
for part in lu(a):
    pprint(part, width=19)
    print
print
b = [[11,9,24,2],[1,5,2,6],[3,17,18,1],[2,5,7,1]]
for part in lu(b):
    pprint(part)
    print
```

Output:

```
[[1.0, 0.0, 0.0],
 [0.5, 1.0, 0.0],
 [0.5, -1.0, 1.0]]

[[2.0, 4.0, 7.0],
 [0.0, 1.0, 1.5],
 [0.0, 0.0, -2.0]]

[[0.0, 1.0, 0.0],
 [1.0, 0.0, 0.0],
 [0.0, 0.0, 1.0]]


[[1.0, 0.0, 0.0, 0.0],
 [0.2727272727272727, 1.0, 0.0, 0.0],
 [0.09090909090909091, 0.2874999999999998, 1.0, 0.0],
 [0.18181818181818182, 0.23124999999999996, 0.003597122302158069, 1.0]]

[[11.0, 9.0, 24.0, 2.0],
 [0.0, 14.545454545454547, 11.454545454545455, 0.4545454545454546],
```

```
 [0.0, 0.0, -3.4749999999999996, 5.6875],
 [0.0, 0.0, 0.0, 0.51079136690647597]]

[[1.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 1.0, 0.0],
 [0.0, 1.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 1.0]]
```

Python: LZW_compression[edit]

In this version the dicts contain mixed typed data:

```
def compress(uncompressed):
    """Compress a string to a list of output symbols."""

    # Build the dictionary.
    dict_size = 256
    dictionary = dict((chr(i), chr(i)) for i in xrange(dict_size))
    # in Python 3: dictionary = {chr(i): chr(i) for i in range(dict_size)}

    w = ""
    result = []
    for c in uncompressed:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            # Add wc to the dictionary.
            dictionary[wc] = dict_size
            dict_size += 1
            w = c

    # Output the code for w.
    if w:
        result.append(dictionary[w])
    return result


def decompress(compressed):
    """Decompress a list of output ks to a string."""
    from cStringIO import StringIO

    # Build the dictionary.
    dict_size = 256
```

```
    dictionary = dict((chr(i), chr(i)) for i in xrange(dict_size))
    # in Python 3: dictionary = {chr(i): chr(i) for i in range(dict_size)}

    # use StringIO, otherwise this becomes O(N^2)
    # due to string concatenation in a loop
    result = StringIO()
    w = compressed.pop(0)
    result.write(w)
    for k in compressed:
        if k in dictionary:
            entry = dictionary[k]
        elif k == dict_size:
            entry = w + w[0]
        else:
            raise ValueError('Bad compressed k: %s' % k)
        result.write(entry)

        # Add w+entry[0] to the dictionary.
        dictionary[dict_size] = w + entry[0]
        dict_size += 1

        w = entry
    return result.getvalue()


# How to use:
compressed = compress('TOBEORNOTTOBEORTOBEORNOT')
print (compressed)
decompressed = decompress(compressed)
print (decompressed)
```

```
Output:



['T', 'O', 'B', 'E', 'O', 'R', 'N', 'O', 'T', 256, 258, 260, 265, 259, 261,
TOBEORNOTTOBEORTOBEORNOT
```

Racket[edit]

Python: Langton's_ant[edit]

Translation of: D

```
width = 75
height = 52
nsteps = 12000
```

```
class Dir: up, right, down, left = range(4)
class Turn: left, right = False, True
class Color: white, black = '.', '#'
M = [[Color.white] * width for _ in range(height)]

x = width // 2
y = height // 2
dir = Dir.up

i = 0
while i < nsteps and 0 <= x < width and 0 <= y < height:
    turn = Turn.left if M[y][x] == Color.black else Turn.right
    M[y][x] = Color.white if M[y][x] == Color.black else Color.black

    dir = (4 + dir + (1 if turn else -1)) % 4
    dir = [Dir.up, Dir.right, Dir.down, Dir.left][dir]
    if   dir == Dir.up:    y -= 1
    elif dir == Dir.right: x -= 1
    elif dir == Dir.down:  y += 1
    elif dir == Dir.left:  x += 1
    else: assert False
    i += 1

print ("\n".join("".join(row) for row in M))
```

The output is the same as the basic D version.

Python: Last_Friday_of_each_month[edit]

```
import calendar
c=calendar.Calendar()
fridays={}
year=raw_input("year")
for item in c.yeardatescalendar(int(year)):
    for i1 in item:
        for i2 in i1:
            for i3 in i2:
                if "Fri" in i3.ctime() and year in i3.ctime():
                    month,day=str(i3).rsplit("-",1)
                    fridays[month]=day

for item in sorted((month+"-"+day for month,day in fridays.items()),
                   key=lambda x:int(x.split("-")[1])):
    print item
```

Using reduce

```
import calendar
c=calendar.Calendar()
fridays={}
year=raw_input("year")
add=list.__add__
for day in reduce(add,reduce(add,reduce(add,c.yeardatescalendar(int(year))))):

    if "Fri" in day.ctime() and year in day.ctime():
        month,day=str(day).rsplit("-",1)
        fridays[month]=day

for item in sorted((month+"-"+day for month,day in fridays.items()),
                   key=lambda x:int(x.split("-")[1])):
    print item
```

using itertools

```
import calendar
from itertools import chain
f=chain.from_iterable
c=calendar.Calendar()
fridays={}
year=raw_input("year")
add=list.__add__

for day in f(f(f(c.yeardatescalendar(int(year))))):

    if "Fri" in day.ctime() and year in day.ctime():
        month,day=str(day).rsplit("-",1)
        fridays[month]=day

for item in sorted((month+"-"+day for month,day in fridays.items()),
                   key=lambda x:int(x.split("-")[1])):
    print item
```

Python: Last_letter-first_letter[edit]

```
from collections import defaultdict

def order_words(words):
    byfirst = defaultdict(set)
    for word in words:
        byfirst[word[0]].add( word )
    #byfirst = dict(byfirst)
    return byfirst
```

```
def linkfirst(byfirst, sofar):
    '''\
    For all words matching last char of last word in sofar as FIRST char an
    return longest chain as sofar + chain
    '''

    assert sofar
    chmatch = sofar[-1][-1]
    options = byfirst[chmatch] - set(sofar)
    #print('  linkfirst options: %r %r' % (chmatch, options))
    if not options:
        return sofar
    else:
        alternatives = ( linkfirst(byfirst, list(sofar) + [word])
                         for word in options )
        mx = max( alternatives, key=len )
        #input('linkfirst: %r' % mx)
        return mx

def llfl(words):

    byfirst = order_words(words)
    return max( (linkfirst(byfirst, [word]) for word in words), key=len )

if __name__ == '__main__':
    pokemon = '''audino bagon baltoy banette bidoof braviary bronzor carrac
cresselia croagunk darmanitan deino emboar emolga exeggcute gabite
girafarig gulpin haxorus heatmor heatran ivysaur jellicent jumpluff kangask
kricketune landorus ledyba loudred lumineon lunatone machamp magnezone mamo
nosepass petilil pidgeotto pikachu pinsir poliwrath poochyena porygon2
porygonz registeel relicanth remoraid rufflet sableye scolipede scrafty sea
sealeo silcoon simisear snivy snorlax spoink starly tirtouga trapinch treec
tyrogue vigoroth vulpix wailord wartortle whismur wingull yamask'''
    pokemon = pokemon.strip().lower().split()
    pokemon = sorted(set(pokemon))
    l = llfl(pokemon)
    for i in range(0, len(l), 8): print(' '.join(l[i:i+8]))
    print(len(l))
```

Sample output

```
audino bagon baltoy banette bidoof braviary bronzor carracosta
charmeleon cresselia croagunk darmanitan deino emboar emolga exeggcute
gabite girafarig gulpin haxorus heatmor heatran ivysaur jellicent
23
```

Alternative version[edit]

Python: Leap_year[edit]

```
import calendar
calendar.isleap(year)
```

or

```
def is_leap_year(year):
    if year % 100 == 0:
        return year % 400 == 0
    return year % 4 == 0
```

Asking for forgiveness instead of permission:

```
import datetime

def is_leap_year(year):
    try:
        datetime.date(year, 2, 29)
    except ValueError:
        return False
    return True
```

Python: Least_common_multiple[edit]

gcd[edit]

Using the fractions libraries gcd function:

```
>>> import fractions
>>> def lcm(a,b): return abs(a * b) / fractions.gcd(a,b) if a and b else 0

>>> lcm(12, 18)
36
>>> lcm(-6, 14)
42
>>> assert lcm(0, 2) == lcm(2, 0) == 0
>>>
```

Prime decomposition[edit]

This imports Prime decomposition#Python

Python: Left_factorials[edit]

```
from itertools import islice

def lfact():
    yield 0
    fact, summ, n = 1, 0, 1
    while 1:
        fact, summ, n = fact*n, summ + fact, n + 1
        yield summ

print('first 11:\n  %r' % [lf for i, lf in zip(range(11), lfact())])
print('20 through 110 (inclusive) by tens:')
for lf in islice(lfact(), 20, 111, 10):
    print(lf)
print('Digits in 1,000 through 10,000 (inclusive) by thousands:\n  %r'
      % [len(str(lf)) for lf in islice(lfact(), 1000, 10001, 1000)] )
```

Output:

```
first 11:
  [0, 1, 2, 4, 10, 34, 154, 874, 5914, 46234, 409114]
20 through 110 (inclusive) by tens:
128425485935180314
9157958657951075573395300940314
20935051082417771847631371547939998232420940314
620960027832821612639424806694551108812720525606160920420940314
14107493072666957100053082208700052221165624211643994900098037874612892042
17363951180298752669971716240928287606555651984960315785085303464481511122
90608958798769534653451680465029063769402483001195636518432767461975209428
```

Python: Letter_frequency[edit]

Using collections.Counter[edit]

Works with: Python version 2.7+ and 3.1+

```
import collections, sys

def filecharcount(openfile):
    return sorted(collections.Counter(c for l in openfile for c in l).items

f = open(sys.argv[1])
print(filecharcount(f))
```

Not using collections.Counter[edit]

```
import string
if hasattr(string, 'ascii_lowercase'):
    letters = string.ascii_lowercase       # Python 2.2 and later
else:
    letters = string.lowercase             # Earlier versions
offset = ord('a')

def countletters(file_handle):
    """Traverse a file and compute the number of occurences of each letter
    return results as a simple 26 element list of integers."""
    results = [0] * len(letters)
    for line in file_handle:
        for char in line:
            char = char.lower()
            if char in letters:
                results[ord(char) - offset] += 1
                # Ordinal minus ordinal of 'a' of any lowercase ASCII lette
    return results

if __name__ == "__main__":
    sourcedata = open(sys.argv[1])
    lettercounts = countletters(sourcedata)
    for i in xrange(len(lettercounts)):
        print "%s=%d" % (chr(i + ord('a')), lettercounts[i]),
```

This example defines the function and provides a sample usage. The *if ...* _

Using a numerically indexed array (list) for this is artificial and clutter

Using defaultdict[edit]

Works with: Python version 2.5+ and 3.x

```
...
from collections import defaultdict
def countletters(file_handle):
    """Count occurences of letters and return a dictionary of them
    """
    results = defaultdict(int)
    for line in file_handle:
        for char in line:
            if char.lower() in letters:
                c = char.lower()
                results[c] += 1
    return results
```

Which eliminates the ungainly fiddling with ordinal values and offsets in f

```
lettercounts = countletters(sourcedata)
for letter,count in lettercounts.iteritems():
    print "%s=%s" % (letter, count),
```

Again eliminating all fussing with the details of converting letters into l

R[edit]

Python: Life_in_two_dimensions[edit]

Using defaultdict[edit]

This implementation uses defaultdict(int) to create dictionaries that retur
This 'trick allows celltable to be initialized to just those keys
with a value of 1.

Python allows many types other than strings and ints to be keys
in a dictionary.
The example uses a dictionary with keys that are a two entry tuple to repre
which also returns a default value of zero.
This simplifies the calculation **N** as out-of-bounds indexing
of universe returns zero.

```
import random
```

```
from collections import defaultdict

printdead, printlive = '-#'
maxgenerations = 3
cellcount = 3,3
celltable = defaultdict(int, {
 (1, 2): 1,
 (1, 3): 1,
 (0, 3): 1,
 } ) # Only need to populate with the keys leading to life

##
## Start States
##
# blinker
u = universe = defaultdict(int)
u[(1,0)], u[(1,1)], u[(1,2)] = 1,1,1

## toad
#u = universe = defaultdict(int)
#u[(5,5)], u[(5,6)], u[(5,7)] = 1,1,1
#u[(6,6)], u[(6,7)], u[(6,8)] = 1,1,1

## glider
#u = universe = defaultdict(int)
#maxgenerations = 16
#u[(5,5)], u[(5,6)], u[(5,7)] = 1,1,1
#u[(6,5)] = 1
#u[(7,6)] = 1

## random start
#universe = defaultdict(int,
#                        # array of random start values
#                        ( ((row, col), random.choice((0,1)))
#                          for col in range(cellcount[0])
#                          for row in range(cellcount[1])
#                        ) )  # returns 0 for out of bounds

for i in range(maxgenerations):
    print "\nGeneration %3i:" % ( i, )
    for row in range(cellcount[1]):
        print "  ", ''.join(str(universe[(row,col)])
                            for col in range(cellcount[0])).replace(
                                '0', printdead).replace('1', printlive)
    nextgeneration = defaultdict(int)
    for row in range(cellcount[1]):
        for col in range(cellcount[0]):
            nextgeneration[(row,col)] = celltable[
                ( universe[(row,col)],
                  -universe[(row,col)] + sum(universe[(r,c)]
                                             for r in range(row-1,row+2)
                                             for c in range(col-1, col+2) )
                ) ]
    universe = nextgeneration
```

Output:

```
 (sample)

Generation   0:
   ---
   ###
   ---

Generation   1:
   -#-
   -#-
   -#-

Generation   2:
   ---
   ###
```

Boardless approach[edit]

A version using the boardless approach.
A world is represented as a set of (x, y) coordinates of all the alive cell

```
from collections import Counter

def life(world, N):
    "Play Conway's game of life for N generations from initial world."
    for g in range(N+1):
        display(world, g)
        counts = Counter(n for c in world for n in offset(neighboring_cells
        world = {c for c in counts
                if counts[c] == 3 or (counts[c] == 2 and c in world)}

neighboring_cells = [(-1, -1), (-1, 0), (-1, 1),
                     ( 0, -1),          ( 0, 1),
                     ( 1, -1), ( 1, 0), ( 1, 1)]

def offset(cells, delta):
    "Slide/offset all the cells by delta, a (dx, dy) vector."
    (dx, dy) = delta
    return {(x+dx, y+dy) for (x, y) in cells}

def display(world, g):
    "Display the world as a grid of characters."
    print '          GENERATION {}:'.format(g)
    Xs, Ys = zip(*world)
    Xrange = range(min(Xs), max(Xs)+1)
```

```
    for y in range(min(Ys), max(Ys)+1):
        print ''.join('#' if (x, y) in world else '.'
                      for x in Xrange)

blinker = {(1, 0), (1, 1), (1, 2)}
block   = {(0, 0), (1, 1), (0, 1), (1, 0)}
toad    = {(1, 2), (0, 1), (0, 0), (0, 2), (1, 3), (1, 1)}
glider  = {(0, 1), (1, 0), (0, 0), (0, 2), (2, 1)}
world   = (block | offset(blinker, (5, 2)) | offset(glider, (15, 5)) | offs
           | {(18, 2), (19, 2), (20, 2), (21, 2)} | offset(block, (35, 7)))


life(world, 5)
```

Output:

```
          GENERATION 0:
##.................................
##.................................
......#..........####..............
......#............................
......#............................
..............##.......#...........
..............#.#......##..........
..............#........##.......##
.........................#.......##
          GENERATION 1:
##.................................
##...............##................
```

Python: Linear_congruential_generator[edit]

```
def bsd_rand(seed):
   def rand():
      rand.seed = (1103515245*rand.seed + 12345) & 0x7fffffff
      return rand.seed
   rand.seed = seed
   return rand

def msvcrt_rand(seed):
   def rand():
      rand.seed = (214013*rand.seed + 2531011) & 0x7fffffff
      return rand.seed >> 16
   rand.seed = seed
   return rand
```

Works with: Python version 3.x

```
def bsd_rand(seed):
   def rand():
      nonlocal seed
      seed = (1103515245*seed + 12345) & 0x7fffffff
      return seed
   return rand

def msvcrt_rand(seed):
   def rand():
      nonlocal seed
      seed = (214013*seed + 2531011) & 0x7fffffff
      return seed >> 16
   return rand
```

Racket[edit]

Python: Linux_CPU_utilization[edit]

```
from __future__ import print_function
from time import sleep


last_idle = last_total = 0
while True:
    with open('/proc/stat') as f:
        fields = [float(column) for column in f.readline().strip().split()[
    idle, total = fields[3], sum(fields)
    idle_delta, total_delta = idle - last_idle, total - last_total
    last_idle, last_total = idle, total
    utilisation = 100.0 * (1.0 - idle_delta / total_delta)
    print('%5.1f%%' % utilisation, end='\r')
    sleep(5)
```

Output:

Lines end in \r which causes old values to be overwritten by new when \r is

```
 26.5%
```

```
 12.4%
 10.6%
 49.5%
 15.5%
 13.8%
  8.3%
 11.0%
 18.5%
 13.9%
 11.8%
 35.6%
```

Python: List_Comprehension[edit]

List comprehension:

```
[(x,y,z) for x in xrange(1,n+1) for y in xrange(x,n+1) for z in xrange(y,n+
```

A Python generator comprehension (note the outer round brackets), returns a

```
((x,y,z) for x in xrange(1,n+1) for y in xrange(x,n+1) for z in xrange(y,n+
```

Python: List_Flattening[edit]

Recursive[edit]

```
>>> def flatten(lst):
        return sum( ([x] if not isinstance(x, list) else flatten(x)
                     for x in lst), [] )

>>> lst = [[1], 2, [[3,4], 5], [[[]]], [[[6]]], 7, 8, []]
>>> flatten(lst)
[1, 2, 3, 4, 5, 6, 7, 8]
```

Non-recursive[edit]

Function flat is iterative and flattens the list in-place. It follows the P

```
>>> def flat(lst):
    i=0
    while i<len(lst):
        while True:
            try:
                lst[i:i+1] = lst[i]
            except (TypeError, IndexError):
                break
        i += 1

>>> lst = [[1], 2, [[3,4], 5], [[[]]], [[[6]]], 7, 8, []]
>>> flat(lst)
>>> lst
[1, 2, 3, 4, 5, 6, 7, 8]
```

Python: List_comprehensions[edit]

List comprehension:

```
[(x,y,z) for x in xrange(1,n+1) for y in xrange(x,n+1) for z in xrange(y,n+
```

A Python generator comprehension (note the outer round brackets), returns a

```
((x,y,z) for x in xrange(1,n+1) for y in xrange(x,n+1) for z in xrange(y,n+
```

R[edit]

Python: List_rooted_trees[edit]

```
def bags(n,cache={}):
        if not n: return [(0, "")]

        upto = sum([bags(x) for x in range(n-1, 0, -1)], [])
        return [(c+1, '('+s+')') for c,s in bagchain((0, ""), n-1, upto)]
```

```
def bagchain(x, n, bb, start=0):
        if not n: return [x]

        out = []
        for i in range(start, len(bb)):
                c,s = bb[i]
                if c <= n: out += bagchain((x[0] + c, x[1] + s), n-c, bb, i
        return out

# Maybe this lessens eye strain. Maybe not.
def replace_brackets(s):
        depth,out = 0,[]
        for c in s:
                if c == '(':
                        out.append("([{"[depth%3])
                        depth += 1
                else:
                        depth -= 1
                        out.append(")]}"[depth%3])
        return "".join(out)

for x in bags(5): print(replace_brackets(x[1]))
```

```
Output:
```

```
([{([])}])
([{()()}])
```

Python: Logical_operations[edit]

```
def logic(a, b):
        print 'a and b:', a and b
        print 'a or b:' , a or b
        print 'not a:'  , not a
```

```
Note: Any normal object can be treated as a Boolean in Python. Numeric obje
```

R[edit]

Python: Long_multiplication[edit]

(Note that Python comes with arbitrary length integers).

```
#!/usr/bin/env python
print 2**64*2**64
```

Python: Longest_Common_Substring[edit]

Using Indexes[edit]

```
import re
s1 = "thisisatest"
s2 = "testing123testing"
longest = ""
i = 0
for x in s1:
    if re.search(x, s2):
      s = x
      while re.search(s, s2):
        if len(s)>len(longest):
            longest = s
        if i+len(s) == len(s1):
            break
        s = s1[i:i+len(s)+1]
    i += 1
print longest
```

Output:

```
"test"
```

Racket[edit]

Python: Longest_string_challenge[edit]

```
import fileinput

# originally, return len(a) - len(b) if positive, 0 otherwise.
# Observing that longer is used for its Boolean result,
# and that '' is False, while any other string is True,
# longer need only to return a after removing len(b) characters,
# which is done without resorting to len().
def longer(a, b):
    while a and b:
        a, b = a[1:], b[1:]
    return a

longest, lines = '', ''
for x in fileinput.input():
    if longer(x, longest):
        lines, longest = x, x
    elif not longer(longest, x):
        lines += x

print(lines, end='')
```

Sample runs

```
paddy@paddy-VirtualBox:~$ cat <<! | python3.2 longlines.py
a
bb
ccc
ddd
ee
f
ggg
!
ccc
ddd
ggg
paddy@paddy-VirtualBox:~$ touch nothing.txt
paddy@paddy-VirtualBox:~$ cat nothing.txt  | python3.2 longlines.py
paddy@paddy-VirtualBox:~$
```

Python: Look-and-say_sequence[edit]

Translation of: C sharp

```
def lookandsay(number):
    result = ""

    repeat = number[0]
    number = number[1:]+" "
    times = 1

    for actual in number:
        if actual != repeat:
            result += str(times)+repeat
            times = 1
            repeat = actual
        else:
            times += 1

    return result

num = "1"

for i in range(10):
    print num
    num = lookandsay(num)
```

Functional

Works with: Python version 2.4+

```
>>> from itertools import groupby
>>> def lookandsay(number):
        return ''.join( str(len(list(g))) + k
                        for k,g in groupby(number) )

>>> numberstring='1'
>>> for i in range(10):
        print numberstring
        numberstring = lookandsay(numberstring)
```

Output:

```
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
```

```
13211311123113112211
```

As a generator

```
>>> from itertools import groupby, islice
>>>
>>> def lookandsay(number='1'):
        while True:
                yield number
                number = ''.join( str(len(list(g))) + k
                                  for k,g in groupby(number) )


>>> print('\n'.join(islice(lookandsay(), 10)))
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
```

Using regular expressions

Python: Loop_Structures[edit]

for[edit]

Frequently one wants to both iterate over a list and increment a counter:

```
 mylist = ["foo", "bar", "baz"]
 for i, x in enumerate(mylist):
     print "Element no.", i, " is", x
```

Iterating over more than one list + incrementing a counter:

```
 for counter, [x, y, z] in enumerate(zip(lst1, lst2, lst3)):
     print counter, x, y, z
```

list comprehension expressions[edit]

Typically used when you want to create a list and there is little logic inv

```
positives = [n for n in numbers if n > 0]
```

A list comprehension is an expression rather than a statement. This allows

```
   def square_each(n):
       results = []
       for each in n:
          results.append(each * each)
       return results
   squares_3x5 = square_each([x for x in range(100) if (x%3)==0 and (x%5)==
   # Return a list of all the squares of numbers from 1 up to 100 those num
   # multiples of both 3 and 5.
```

while[edit]

Typical use:

```
 while True:
     # Do stuff...
     if condition:
         break
```

You can add optional *else*, which is executed only if the expression tested

```
 while True:
     # Do stuff...
     if found:
         results = ...
         break
 else:
    print 'Not found'
```

Since Python has no "bottom-tested" loop construct (such as "do ... until")

Python: Lucas-Lehmer_test[edit]

```
from sys import stdout
from math import sqrt, log

def is_prime ( p ):
  if p == 2: return True # Lucas-Lehmer test only works on odd primes
  elif p <= 1 or p % 2 == 0: return False
  else:
    for i in range(3, int(sqrt(p))+1, 2 ):
      if p % i == 0: return False
    return True

def is_mersenne_prime ( p ):
  if p == 2:
    return True
  else:
    m_p = ( 1 << p ) - 1
    s = 4
    for i in range(3, p+1):
      s = (s ** 2 - 2) % m_p
    return s == 0

precision = 20000   # maximum requested number of decimal places of 2 ** MP
long_bits_width = precision * log(10, 2)
upb_prime = int( long_bits_width - 1 ) / 2    # no unsigned #
upb_count = 45      # find 45 mprimes if int was given enough bits #

print (" Finding Mersenne primes in M[2..%d]:"%upb_prime)

count=0
for p in range(2, int(upb_prime+1)):
  if is_prime(p) and is_mersenne_prime(p):
```

```
    print("M%d"%p),
    stdout.flush()
    count += 1
  if count >= upb_count: break
print
```

Output:

```
 Finding Mersenne primes in M[2..33218]:
 M2 M3 M5 M7 M13 M17 M19 M31 M61 M89 M107 M127 M521 M607 M1279 M2203 M2281 M
```

Faster loop without division[edit]

```
def isqrt(n):
    if n < 0:
        raise ValueError
    elif n < 2:
        return n
    else:
        a = 1 << ((1 + n.bit_length()) >> 1)
        while True:
            b = (a + n // a) >> 1
            if b >= a:
                return a
            a = b

def isprime(n):
    if n < 5:
        return n == 2 or n == 3
    elif n%2 == 0:
        return False
    else:
        r = isqrt(n)
        k = 3
        while k <= r:
            if n%k == 0:
                return False
            k += 2
        return True

def lucas_lehmer_fast(n):
    if n == 2:
        return True
    elif not isprime(n):
        return False
```

```
    else:
        m = 2**n - 1
        s = 4
        for i in range(2, n):
            sqr = s*s
            s = (sqr & m) + (sqr >> n)
            if s >= m:
                s -= m
            s -= 2
        return s == 0

# test taken from the previous rosetta implementation

from math import log
from sys import stdout

precision = 20000   # maximum requested number of decimal places of 2 ** MP
long_bits_width = precision * log(10, 2)
upb_prime = int( long_bits_width - 1 ) / 2    # no unsigned #
# upb_count = 45      # find 45 mprimes if int was given enough bits #
upb_count = 15      # find 45 mprimes if int was given enough bits #

print (" Finding Mersenne primes in M[2..%d]:"%upb_prime)

count=0
# for p in range(2, upb_prime+1):
for p in range(2, int(upb_prime+1)):
  if lucas_lehmer_fast(p):
    print("M%d"%p),
    stdout.flush()
    count += 1
  if count >= upb_count: break
print
```

The main loop may be run much faster using gmpy2 :

```
import gmpy2 as mp

def lucas_lehmer(n):
    if n == 2:
        return True
    if not mp.is_prime(n):
        return False
    two = mp.mpz(2)
    m = two**n - 1
    s = two*two
    for i in range(2, n):
        sqr = s*s
        s = (sqr & m) + (sqr >> n)
        if s >= m:
            s -= m
```

```
        s -= two
    return mp.is_zero(s)
```

With this, one can test all primes below 10^5 in around 24 hours on a Core

The primes found are

2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3

Of course, they agree with OEIS A000043.

Racket[edit]

Python: Lucky_and_even_lucky_numbers[edit]

The generator

```
from __future__ import print_function

def lgen(even=False, nmax=1000000):
    start = 2 if even else 1
    n, lst = 1, list(range(start, nmax + 1, 2))
    lenlst = len(lst)
    yield lst[0]
    while n < lenlst and lst[n] < lenlst:
        yield lst[n]
        n, lst = n + 1, [j for i,j in enumerate(lst, 1) if i % lst[n]]
        lenlst = len(lst)
    # drain
    for i in lst[n:]:
        yield i
```

The argument handler

```
from itertools import islice
import sys, re

class ArgumentError(Exception):
    pass
def arghandler(argstring):
    match_obj = re.match( r"""(?mx)
    (?:
```

```
    (?P<SINGLE>
       (?: ^ (?P<SINGLEL> \d+ ) (?:  | \s , \s lucky ) \s* $ )
      |(?: ^ (?P<SINGLEE> \d+ ) (?:  | \s , \s evenLucky ) \s* $ )
    )
   |(?P<KTH>
       (?: ^ (?P<KTHL> \d+ \s \d+ ) (?:  | \s lucky ) \s* $ )
      |(?: ^ (?P<KTHE> \d+ \s \d+ ) (?:  | \s evenLucky ) \s* $ )
    )
   |(?P<RANGE>
       (?: ^ (?P<RANGEL> \d+ \s -\d+ ) (?:  | \s lucky ) \s* $ )
      |(?: ^ (?P<RANGEE> \d+ \s -\d+ ) (?:  | \s evenLucky ) \s* $ )
    )
  )""", argstring)

  if match_obj:
      # Retrieve group(s) by name
      SINGLEL = match_obj.group('SINGLEL')
      SINGLEE = match_obj.group('SINGLEE')
      KTHL = match_obj.group('KTHL')
      KTHE = match_obj.group('KTHE')
      RANGEL = match_obj.group('RANGEL')
      RANGEE = match_obj.group('RANGEE')
      if SINGLEL:
          j = int(SINGLEL)
          assert 0 < j < 10001, "Argument out of range"
          print("Single %i'th lucky number:" % j, end=' ')
          print( list(islice(lgen(), j-1, j))[0] )
      elif SINGLEE:
          j = int(SINGLEE)
          assert 0 < j < 10001, "Argument out of range"
          print("Single %i'th even lucky number:" % j, end=' ')
          print( list(islice(lgen(even=True), j-1, j))[0] )
      elif KTHL:
          j, k = [int(num) for num in KTHL.split()]
          assert 0 < j < 10001, "first argument out of range"
          assert 0 < k < 10001 and k > j, "second argument out of range"
          print("List of %i ... %i lucky numbers:" % (j, k), end=' ')
          for n, luck in enumerate(lgen(), 1):
              if n > k: break
              if n >=j: print(luck, end = ', ')
          print('')
      elif KTHE:
          j, k = [int(num) for num in KTHE.split()]
          assert 0 < j < 10001, "first argument out of range"
          assert 0 < k < 10001 and k > j, "second argument out of range"
          print("List of %i ... %i even lucky numbers:" % (j, k), end=' '
          for n, luck in enumerate(lgen(even=True), 1):
              if n > k: break
              if n >=j: print(luck, end = ', ')
          print('')
      elif RANGEL:
          j, k = [int(num) for num in RANGEL.split()]
          assert 0 < j < 10001, "first argument out of range"
          assert 0 < -k < 10001 and -k > j, "second argument out of range
          k = -k
          print("List of lucky numbers in the range %i ... %i :" % (j, k)
```

```
            for n in lgen():
                if n > k: break
                if n >=j: print(n, end = ', ')
            print('')
        elif RANGEE:
            j, k = [int(num) for num in RANGEE.split()]
            assert 0 < j < 10001, "first argument out of range"
            assert 0 < -k < 10001 and -k > j, "second argument out of range
            k = -k
            print("List of even lucky numbers in the range %i ... %i :" % (
            for n in lgen(even=True):
                if n > k: break
                if n >=j: print(n, end = ', ')
            print('')
    else:
        raise ArgumentError('''

  Error Parsing Arguments!

  Expected Arguments of the form (where j and k are integers):

      j                #  Jth       lucky number
      j  ,      lucky  #  Jth       lucky number
      j  ,  evenLucky  #  Jth  even lucky number
                       #
      j  k             #  Jth  through  Kth (inclusive)       lucky numbers
      j  k      lucky  #  Jth  through  Kth (inclusive)       lucky numbers
      j  k  evenLucky  #  Jth  through  Kth (inclusive)  even lucky numbers
                       #
      j -k             #  all       lucky numbers in the range  j --? |k|
      j -k      lucky  #  all       lucky numbers in the range  j --? |k|
      j -k  evenLucky  #  all  even lucky numbers in the range  j --? |k|
        ''')

if __name__ == '__main__':
    arghandler(' '.join(sys.argv[1:]))
```

Output:

```
# Output when arguments are: 1 20 lucky
List of 1 ... 20 lucky numbers: 1, 3, 7, 9, 13, 15, 21, 25, 31, 33, 37, 43,
# Output when arguments are: 1 20 evenLucky
List of 1 ... 20 even lucky numbers: 2, 4, 6, 10, 12, 18, 20, 22, 26, 34, 3
# Output when arguments are: 6000 -6100 lucky
List of lucky numbers in the range 6000 ... 6100 : 6009, 6019, 6031, 6049, 6
# Output when arguments are: 6000 -6100 evenLucky
```

Python: Luhn_test[edit]

The divmod in the function below conveniently splits a number into its two

```
>>> def luhn(n):
        r = [int(ch) for ch in str(n)][::-1]
        return (sum(r[0::2]) + sum(sum(divmod(d*2,10)) for d in r[1::2])) %

>>> for n in (49927398716, 49927398717, 1234567812345678, 1234567812345670)
        print(n, luhn(n))
```

Output:

```
49927398716 True
49927398717 False
1234567812345678 False
1234567812345670 True
```

Python: Luhn_test_of_credit_card_numbers[edit]

The divmod in the function below conveniently splits a number into its two

```
>>> def luhn(n):
        r = [int(ch) for ch in str(n)][::-1]
        return (sum(r[0::2]) + sum(sum(divmod(d*2,10)) for d in r[1::2])) %

>>> for n in (49927398716, 49927398717, 1234567812345678, 1234567812345670)
        print(n, luhn(n))
```

Output:

```
49927398716 True
49927398717 False
1234567812345678 False
1234567812345670 True
```

Python: Lychrel_numbers[edit]

```
from __future__ import print_function

def add_reverse(num, max_iter=1000):
    i, nums = 0, {num}
    while True:
        i, num = i+1, num + reverse_int(num)
        nums.add(num)
        if reverse_int(num) == num or i >= max_iter:
            break
    return nums

#@functools.lru_cache(maxsize=2**20)
def reverse_int(num):
    return int(str(num)[::-1])

def split_roots_from_relateds(roots_and_relateds):
    roots = roots_and_relateds[::]
    i = 1
    while i < len(roots):
        this = roots[i]
        if any(this.intersection(prev) for prev in roots[:i]):
            del roots[i]
        else:
            i += 1
    root = [min(each_set) for each_set in roots]
    related = [min(each_set) for each_set in roots_and_relateds]
    related = [n for n in related if n not in root]
    return root, related

def find_lychrel(maxn, max_reversions):
    'Lychrel number generator'
    series = [add_reverse(n, max_reversions*2) for n in range(1, maxn + 1)]
    roots_and_relateds = [s for s in series if len(s) > max_reversions]
    return split_roots_from_relateds(roots_and_relateds)


if __name__ == '__main__':
    maxn, reversion_limit = 10000, 500
    print("Calculations using n = 1..%i and limiting each search to 2*%i re
          % (maxn, reversion_limit))
    lychrel, l_related = find_lychrel(maxn, reversion_limit)
    print('  Number of Lychrel numbers:', len(lychrel))
    print('    Lychrel numbers:', ', '.join(str(n) for n in lychrel))
    print('  Number of Lychrel related:', len(l_related))
    #print('    Lychrel related:', ', '.join(str(n) for n in l_related))
    pals = [x for x in lychrel + l_related  if x == reverse_int(x)]
    print('  Number of Lychrel palindromes:', len(pals))
    print('    Lychrel palindromes:', ', '.join(str(n) for n in pals))
```

Output:

```
Calculations using n = 1..10000 and limiting each search to 2*500 reverse-d
  Number of Lychrel numbers: 5
    Lychrel numbers: 196, 879, 1997, 7059, 9999
  Number of Lychrel related: 244
  Number of Lychrel palindromes: 3
    Lychrel palindromes: 9999, 4994, 8778
```

Python: MD4[edit]

Use 'hashlib' from python's standard library.

Library: hashlib

```
import hashlib
print hashlib.new("md4",raw_input().encode('utf-16le')).hexdigest().upper()
```

Python: Machine_code[edit]

Works with: CPython version 3.x

The ctypes module is meant for calling existing native code from Python, bu

```
import ctypes
import os
from ctypes import c_ubyte, c_int

code = bytes([0x8b, 0x44, 0x24, 0x04, 0x03, 0x44, 0x24, 0x08, 0xc3])

code_size = len(code)
# copy code into an executable buffer
if (os.name == 'posix'):
    import mmap
    executable_map = mmap.mmap(-1, code_size, mmap.MAP_PRIVATE | mmap.MAP_A
    # we must keep a reference to executable_map until the call, to avoid f
    executable_map.write(code)
    # the mmap object won't tell us the actual address of the mapping, but 
    # some ctypes object over its buffer, then asking the address of that
    func_address = ctypes.addressof(c_ubyte.from_buffer(executable_map))
elif (os.name == 'nt'):
    # the mmap module doesn't support protection flags on Windows, so execu
    code_buffer = ctypes.create_string_buffer(code)
```

```
    PAGE_EXECUTE_READWRITE = 0x40  # Windows constants that would usually c
    MEM_COMMIT = 0x1000
    executable_buffer_address = ctypes.windll.kernel32.VirtualAlloc(0, code_
    if (executable_buffer_address == 0):
        print('Warning: Failed to enable code execution, call will likely c
        func_address = ctypes.addressof(code_buffer)
    else:
        ctypes.memmove(executable_buffer_address, code_buffer, code_size)
        func_address = executable_buffer_address
else:
    # for other platforms, we just hope DEP isn't enabled
    code_buffer = ctypes.create_string_buffer(code)
    func_address = ctypes.addressof(code_buffer)

prototype = ctypes.CFUNCTYPE(c_int, c_ubyte, c_ubyte) # build a function pr
func = prototype(func_address)                        # build an actual fun
res = func(7,12)
print(res)
```

Python: Mad_Libs[edit]

```
import re

# Optional Python 2.x compatibility
#try: input = raw_input
#except: pass

template = '''<name> went for a walk in the park. <he or she>
found a <noun>. <name> decided to take it home.'''

def madlibs(template):
    print('The story template is:\n' + template)
    fields = sorted(set( re.findall('<[^>]+>', template) ))
    values = input('\nInput a comma-separated list of words to replace the
                   '\n  %s: ' % ','.join(fields)).split(',')
    story = template
    for f,v in zip(fields, values):
        story = story.replace(f, v)
    print('\nThe story becomes:\n\n' + story)

madlibs(template)
```

Output:

```
The story template is:
<name> went for a walk in the park. <he or she>
```

```
found a <noun>. <name> decided to take it home.

Input a comma-separated list of words to replace the following items
  <he or she>,<name>,<noun>: She,Monica L.,cockerel

The story becomes:

Monica L. went for a walk in the park. She
found a cockerel. Monica L. decided to take it home.
```

Python: Magic_squares_of_odd_order[edit]

```
>>> def magic(n):
    for row in range(1, n + 1):
        print(' '.join('%*i' % (len(str(n**2)), cell) for cell in
                       (n * ((row + col - 1 + n // 2) % n) +
                       ((row + 2 * col - 2) % n) + 1
                       for col in range(1, n + 1))))
    print('\nAll sum to magic number %i' % ((n * n + 1) * n // 2))


>>> for n in (5, 3, 7):
        print('\nOrder %i\n=======' % n)
        magic(n)
```

```
Order 5
=======
17 24  1  8 15
23  5  7 14 16
 4  6 13 20 22
10 12 19 21  3
11 18 25  2  9

All sum to magic number 65

Order 3
=======
8 1 6
3 5 7
4 9 2

All sum to magic number 15

Order 7
=======
30 39 48  1 10 19 28
38 47  7  9 18 27 29
46  6  8 17 26 35 37
 5 14 16 25 34 36 45
```

```
13 15 24 33 42 44  4
21 23 32 41 43  3 12
22 31 40 49  2 11 20

All sum to magic number 175
>>>
```

Racket[edit]

Python: Main_step_of_GOST_28147-89[edit]

Translation of: C

```
k8 = [  14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7 ]
k7 = [  15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10 ]
k6 = [  10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8 ]
k5 = [   7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15 ]
k4 = [   2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9 ]
k3 = [  12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11 ]
k2 = [   4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1 ]
k1 = [  13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7 ]

k87 = [0] * 256
k65 = [0] * 256
k43 = [0] * 256
k21 = [0] * 256

def kboxinit():
        for i in range(256):
                k87[i] = k8[i >> 4] << 4 | k7[i & 15]
                k65[i] = k6[i >> 4] << 4 | k5[i & 15]
                k43[i] = k4[i >> 4] << 4 | k3[i & 15]
                k21[i] = k2[i >> 4] << 4 | k1[i & 15]

def f(x):
        x = ( k87[x>>24 & 255] << 24 | k65[x>>16 & 255] << 16 |
              k43[x>> 8 & 255] <<  8 | k21[x & 255] )
        return x<<11 | x>>(32-11)
```

Python: Make_a_backup_file[edit]

```
import os
```

```
targetfile = "pycon-china"
os.rename(os.path.realpath(targetfile), os.path.realpath(targetfile)+".bak"
f = open(os.path.realpath(targetfile), "w")
f.write("this task was solved during a talk about rosettacode at the PyCon
f.close()
```

Racket[edit]

Python: Make_directory_path[edit]

```
from errno import EEXIST
from os import mkdir, curdir
from os.path import split, exists

def mkdirp(path, mode=0777):
    head, tail = split(path)
    if not tail:
        head, tail = split(head)
    if head and tail and not exists(head):
        try:
            mkdirp(head, mode)
        except OSError as e:
            # be happy if someone already created the path
            if e.errno != EEXIST:
                raise
        if tail == curdir:  # xxx/newdir/. exists if xxx/newdir exists
            return
    try:
        mkdir(path, mode)
    except OSError as e:
        # be happy if someone already created the path
        if e.errno != EEXIST:
            raise
```

Python: Man_or_boy_test[edit]

Works with: Python version 2.5

```
#!/usr/bin/env python
import sys
sys.setrecursionlimit(1025)
```

```
def a(in_k, x1, x2, x3, x4, x5):
    k = [in_k]
    def b():
        k[0] -= 1
        return a(k[0], b, x1, x2, x3, x4)
    return x4() + x5() if k[0] <= 0 else b()

x = lambda i: lambda: i
print(a(10, x(1), x(-1), x(-1), x(1), x(0)))
```

A better-looking alternative to using lists as storage are function attribu

```
#!/usr/bin/env python
import sys
sys.setrecursionlimit(1025)

def a(k, x1, x2, x3, x4, x5):
    def b():
        b.k -= 1
        return a(b.k, b, x1, x2, x3, x4)
    b.k = k
    return x4() + x5() if b.k <= 0 else b()

x = lambda i: lambda: i
print(a(10, x(1), x(-1), x(-1), x(1), x(0)))
```

Output:

Python: Mandelbrot_set[edit]

Translation of the ruby solution

```
# Python 3.0+ and 2.5+
try:
    from functools import reduce
except:
    pass


def mandelbrot(a):
    return reduce(lambda z, _: z * z + a, range(50), 0)
```

```
def step(start, step, iterations):
    return (start + (i * step) for i in range(iterations))

rows = (("*" if abs(mandelbrot(complex(x, y))) < 2 else " "
        for x in step(-2.0, .0315, 80))
        for y in step(1, -.05, 41))

print("\n".join("".join(row) for row in rows))
```

A more "Pythonic" version of the code:

```
import math

def mandelbrot(z , c , n=40):
    if abs(z) > 1000:
        return float("nan")
    elif n > 0:
        return mandelbrot(z ** 2 + c, c, n - 1)
    else:
        return z ** 2 + c

print("\n".join(["".join(["#" if not math.isnan(mandelbrot(0, x + 1j * y).r
                 for x in [a * 0.02 for a in range(-80, 30)]])
                 for y in [a * 0.05 for a in range(-20, 20)]])
     )
```

Finally, we can also use Matplotlib to visualize the Mandelbrot set with Py

Library: matplotlib

Library: numpy

```
from pylab import *
from numpy import NaN

def m(a):
        z = 0
        for n in range(1, 100):
                z = z**2 + a
                if abs(z) > 2:
                        return n
        return NaN

X = arange(-2, .5, .002)
Y = arange(-1,  1, .002)
```

```
Z = zeros((len(Y), len(X)))

for iy, y in enumerate(Y):
        print (iy, "of", len(Y))
        for ix, x in enumerate(X):
                Z[iy,ix] = m(x + 1j * y)

imshow(Z, cmap = plt.cm.prism, interpolation = 'none', extent = (X.min(), X
xlabel("Re(c)")
ylabel("Im(c)")
savefig("mandelbrot_python.svg")
show()
```

R[edit]

Python: Map_range[edit]

```
>>> def maprange( a, b, s):
        (a1, a2), (b1, b2) = a, b
        return  b1 + ((s - a1) * (b2 - b1) / (a2 - a1))

>>> for s in range(11):
        print("%2g maps to %g" % (s, maprange( (0, 10), (-1, 0), s)))
```

```
 0 maps to -1
 1 maps to -0.9
 2 maps to -0.8
 3 maps to -0.7
 4 maps to -0.6
 5 maps to -0.5
 6 maps to -0.4
 7 maps to -0.3
 8 maps to -0.2
 9 maps to -0.1
10 maps to 0
```

Because of Pythons strict, dynamic, typing rules for numbers the same funct

```
>>> from fractions import Fraction
>>> for s in range(11):
        print("%2g maps to %s" % (s, maprange( (0, 10), (-1, 0), Fraction(s
```

```
 0 maps to -1
```

```
 1 maps to -9/10
 2 maps to -4/5
 3 maps to -7/10
 4 maps to -3/5
 5 maps to -1/2
 6 maps to -2/5
 7 maps to -3/10
 8 maps to -1/5
 9 maps to -1/10
10 maps to 0
>>>
```

Racket[edit]

Python: Markov_Algorithm[edit]

The example uses a regexp to parse the syntax of the grammar. This regexp i

The example gains flexibility by not being tied to specific files. The func

```
import re

def extractreplacements(grammar):
    return [ (matchobj.group('pat'), matchobj.group('repl'), bool(matchobj.
                for matchobj in re.finditer(syntaxre, grammar)
                if matchobj.group('rule')]

def replace(text, replacements):
    while True:
        for pat, repl, term in replacements:
            if pat in text:
                text = text.replace(pat, repl, 1)
                if term:
                    return text
                break
        else:
            return text

syntaxre = r"""(?mx)
^(?:
  (?: (?P<comment> \# .* ) ) |
  (?: (?P<blank>   \s*  ) (?: \n | $ )  ) |
  (?: (?P<rule>    (?P<pat> .+? ) \s+ -> \s+ (?P<term> \.)? (?P<repl> .+) )
)$
"""

grammar1 = """\
```

```
# This rules file is extracted from Wikipedia:
# http://en.wikipedia.org/wiki/Markov_Algorithm
A -> apple
B -> bag
S -> shop
T -> the
the shop -> my brother
a never used -> .terminating rule
"""

grammar2 = '''\
# Slightly modified from the rules on Wikipedia
A -> apple
B -> bag
S -> .shop
T -> the
the shop -> my brother
a never used -> .terminating rule
'''

grammar3 = '''\
# BNF Syntax testing rules
A -> apple
WWWW -> with
Bgage -> ->.*
B -> bag
->.* -> money
W -> WW
S -> .shop
T -> the
the shop -> my brother
a never used -> .terminating rule
'''

grammar4 = '''\
### Unary Multiplication Engine, for testing Markov Algorithm implementatio
### By Donal Fellows.
# Unary addition engine
_+1 -> _1+
1+1 -> 11+
# Pass for converting from the splitting of multiplication into ordinary
# addition
1! -> !1
,! -> !+
_! -> _
# Unary multiplication by duplicating left side, right side times
1*1 -> x,@y
1x -> xX
X, -> 1,1
X1 -> 1X
_x -> _X
,x -> ,X
y1 -> 1y
y_ -> _
# Next phase of applying
1@1 -> x,@y
```

```
1@_ -> @_
,@_ -> !_
++ -> +
# Termination cleanup for addition
_1 -> 1
1+_ -> 1
_+_ ->
'''


grammar5 = '''\
# Turing machine: three-state busy beaver
#
# state A, symbol 0 => write 1, move right, new state B
A0 -> 1B
# state A, symbol 1 => write 1, move left, new state C
0A1 -> C01
1A1 -> C11
# state B, symbol 0 => write 1, move left, new state A
0B0 -> A01
1B0 -> A11
# state B, symbol 1 => write 1, move right, new state B
B1 -> 1B
# state C, symbol 0 => write 1, move left, new state B
0C0 -> B01
1C0 -> B11
# state C, symbol 1 => write 1, move left, halt
0C1 -> H01
1C1 -> H11
'''

text1 = "I bought a B of As from T S."

text2 = "I bought a B of As W my Bgage from T S."

text3 = '_1111*11111_'

text4 = '000000A000000'



if __name__ == '__main__':
    assert replace(text1, extractreplacements(grammar1)) \
           == 'I bought a bag of apples from my brother.'
    assert replace(text1, extractreplacements(grammar2)) \
           == 'I bought a bag of apples from T shop.'
    # Stretch goals
    assert replace(text2, extractreplacements(grammar3)) \
           == 'I bought a bag of apples with my money from T shop.'
    assert replace(text3, extractreplacements(grammar4)) \
           == '11111111111111111111'
    assert replace(text4, extractreplacements(grammar5)) \
           == '00011H1111000'
```

Python: Math_constants_and_functions[edit]

```
import math

math.e          # e
math.pi         # pi
math.sqrt(x)    # square root  (Also commonly seen as x ** 0.5 to obviate i
math.log(x)     # natural logarithm
math.log10(x)   # base 10 logarithm
math.exp(x)     # e raised to the power of x
abs(x)          # absolute value
math.floor(x)   # floor
math.ceil(x)    # ceiling
x ** y          # exponentiation
pow(x, y[, n])  # exponentiation [, modulo n (useful in certain encryption/

# The math module constants and functions can, of course, be imported direc
#   from math import e, pi, sqrt, log, log10, exp, floor, ceil
```

R[edit]

Python: Matrix-exponentiation_operator[edit]

Using matrixMul from Matrix multiplication#Python

Python: Matrix_Multiplication[edit]

```
a=((1,  1,  1,   1), # matrix A #
     (2,  4,  8,  16),
     (3,  9, 27,  81),
     (4, 16, 64, 256))

b=((  4  , -3  ,  4/3.,  -1/4. ), # matrix B #
     (-13/3., 19/4., -7/3.,  11/24.),
     (  3/2., -2.  ,  7/6.,  -1/4. ),
     ( -1/6.,  1/4., -1/6.,   1/24.))



def MatrixMul( mtx_a, mtx_b):
    tpos_b = zip( *mtx_b)
    rtn = [[ sum( ea*eb for ea,eb in zip(a,b)) for b in tpos_b] for a in mt
    return rtn


v = MatrixMul( a, b )
```

```
print 'v = ('
for r in v:
    print '[',
    for val in r:
        print '%8.2f '%val,
    print ']'
print ')'


u = MatrixMul(b,a)

print 'u = '
for r in u:
    print '[',
    for val in r:
        print '%8.2f '%val,
    print ']'
print ')'
```

Another one,

Translation of: Scheme

```
from operator import mul

def matrixMul(m1, m2):
  return map(
    lambda row:
      map(
        lambda *column:
          sum(map(mul, row, column)),
        *m2),
    m1)
```

Python: Matrix_Transpose[edit]

```
m=((1,  1,  1,   1),
   (2,  4,  8,  16),
   (3,  9, 27,  81),
   (4, 16, 64, 256),
   (5, 25,125, 625))
print(zip(*m))
# in Python 3.x, you would do:
# print(list(zip(*m)))
```

Output:

```
 [(1, 2, 3, 4, 5),
  (1, 4, 9, 16, 25),
  (1, 8, 27, 64, 125),
  (1, 16, 81, 256, 625)]
```

Python: Matrix_arithmetic[edit]

Using the module file spermutations.py from Permutations by swapping. The a

```
from itertools import permutations
from operator import mul
from math import fsum
from spermutations import spermutations

def prod(lst):
    return reduce(mul, lst, 1)

def perm(a):
    n = len(a)
    r = range(n)
    s = permutations(r)
    return fsum(prod(a[i][sigma[i]] for i in r) for sigma in s)

def det(a):
    n = len(a)
    r = range(n)
    s = spermutations(n)
    return fsum(sign * prod(a[i][sigma[i]] for i in r)
                for sigma, sign in s)

if __name__ == '__main__':
    from pprint import pprint as pp

    for a in (
            [
             [1, 2],
             [3, 4]],

            [
             [1, 2, 3, 4],
             [4, 5, 6, 7],
             [7, 8, 9, 10],
             [10, 11, 12, 13]],

            [
             [ 0,  1,  2,  3,  4],
             [ 5,  6,  7,  8,  9],
             [10, 11, 12, 13, 14],
```

```
            [15, 16, 17, 18, 19],
            [20, 21, 22, 23, 24]],
        ):
        print('')
        pp(a)
        print('Perm: %s Det: %s' % (perm(a), det(a)))
```

Sample output

```
[[1, 2], [3, 4]]
Perm: 10 Det: -2

[[1, 2, 3, 4], [4, 5, 6, 7], [7, 8, 9, 10], [10, 11, 12, 13]]
```

Python: Matrix_exponentiation_operator[edit]

Using matrixMul from Matrix multiplication#Python

```
>>> from operator import mul
>>> def matrixMul(m1, m2):
  return map(
    lambda row:
      map(
        lambda *column:
          sum(map(mul, row, column)),
        *m2),
    m1)

>>> def identity(size):
        size = range(size)
        return [[(i==j)*1 for i in size] for j in size]

>>> def matrixExp(m, pow):
        assert pow>=0 and int(pow)==pow, "Only non-negative, integer powers
        accumulator = identity(len(m))
        for i in range(pow):
                accumulator = matrixMul(accumulator, m)
        return accumulator

>>> def printtable(data):
        for row in data:
                print ' '.join('%-5s' % ('%s' % cell) for cell in row)


>>> m = [[3,2], [2,1]]
>>> for i in range(5):
        print '\n%i:' % i
```

```
        printtable( matrixExp(m, i) )



0:
1     0
0     1

1:
3     2
2     1

2:
13    8
8     5

3:
55    34
34    21

4:
233   144
144   89
>>> printtable( matrixExp(m, 10) )
1346269 832040
832040 514229
>>>
```

R[edit]

Python: Matrix_multiplication[edit]

```
a=((1,  1,  1,   1), # matrix A #
     (2,  4,  8,  16),
     (3,  9, 27,  81),
     (4, 16, 64, 256))

b=((  4  , -3  ,  4/3.,  -1/4. ), # matrix B #
     (-13/3., 19/4., -7/3.,  11/24.),
     (  3/2., -2.  ,  7/6.,  -1/4. ),
     ( -1/6.,  1/4., -1/6.,   1/24.))



def MatrixMul( mtx_a, mtx_b):
    tpos_b = zip( *mtx_b)
    rtn = [[ sum( ea*eb for ea,eb in zip(a,b)) for b in tpos_b] for a in mt
    return rtn
```

```
v = MatrixMul( a, b )

print 'v = ('
for r in v:
    print '[',
    for val in r:
        print '%8.2f '%val,
    print ']'
print ')'


u = MatrixMul(b,a)

print 'u = '
for r in u:
    print '[',
    for val in r:
        print '%8.2f '%val,
    print ']'
print ')'
```

Another one,
Translation of: Scheme

```
from operator import mul

def matrixMul(m1, m2):
  return map(
    lambda row:
      map(
        lambda *column:
          sum(map(mul, row, column)),
        *m2),
    m1)
```

Python: Matrix_transposition[edit]

```
m=((1,  1,  1,   1),
   (2,  4,  8,  16),
   (3,  9, 27,  81),
   (4, 16, 64, 256),
   (5, 25,125, 625))
print(zip(*m))
# in Python 3.x, you would do:
# print(list(zip(*m)))
```

Output:

```
 [(1, 2, 3, 4, 5),
  (1, 4, 9, 16, 25),
  (1, 8, 27, 64, 125),
  (1, 16, 81, 256, 625)]
```

Python: Max_Licenses_In_Use[edit]

```
out, max_out, max_times = 0, -1, []
for job in open('mlijobs.txt'):
    out += 1 if "OUT" in job else -1
    if out > max_out:
        max_out, max_times = out, []
    if out == max_out:
        max_times.append(job.split()[3])

print("Maximum simultaneous license use is %i at the following times:" % ma
print('  ' + '\n  '.join(max_times))
```

Output:

```
Maximum simultaneous license use is 99 at the following times:
  2008/10/03_08:39:34
  2008/10/03_08:40:40
```

Python: Maximum_triangle_path_sum[edit]

A simple mostly imperative solution:

```
def solve(tri):
    while len(tri) > 1:
        t0 = tri.pop()
        t1 = tri.pop()
        tri.append([max(t0[i], t0[i+1]) + t for i,t in enumerate(t1)])
    return tri[0][0]


data = """
```

```
                         94 48
                        95 30 96
                      77 71 26 67
                     97 13 76 38 45
                   07 36 79 16 37 68
                  48 07 09 18 70 26 06
                18 72 79 46 59 79 29 90
               20 76 87 11 32 07 07 49 18
             27 83 58 35 71 11 25 57 29 85
            14 64 36 96 27 11 58 56 92 18 55
          02 90 03 60 48 49 41 46 33 36 47 23
         92 50 48 02 36 59 42 79 72 20 82 77 42
       56 78 38 80 39 75 02 71 66 66 01 03 55 72
      44 25 67 84 71 67 11 61 40 57 58 89 40 56 36
    85 32 25 85 57 48 84 35 47 62 17 01 01 99 89 52
   06 71 28 75 94 48 37 10 23 51 06 48 53 18 74 98 15
27 02 92 23 08 71 76 84 15 52 92 63 81 10 44 10 69 93"""

print solve([map(int, row.split()) for row in data.splitlines()])
```

Output:

```
1320
```

A more functional version, similar to the Haskell entry (same output):

Python: Maze_generation[edit]

```
from random import shuffle, randrange

def make_maze(w = 16, h = 8):
    vis = [[0] * w + [1] for _ in range(h)] + [[1] * (w + 1)]
    ver = [["|  "] * w + ['|'] for _ in range(h)] + [[]]
    hor = [["+--"] * w + ['+'] for _ in range(h + 1)]

    def walk(x, y):
        vis[y][x] = 1

        d = [(x - 1, y), (x, y + 1), (x + 1, y), (x, y - 1)]
        shuffle(d)
        for (xx, yy) in d:
            if vis[yy][xx]: continue
            if xx == x: hor[max(y, yy)][x] = "+  "
            if yy == y: ver[y][max(x, xx)] = "   "
            walk(xx, yy)

    walk(randrange(w), randrange(h))
```

```
    s = ""
    for (a, b) in zip(hor, ver):
        s += ''.join(a + ['\n'] + b + ['\n'])
    return s

if __name__ == '__main__':
    print(make_maze())
```

Output:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|        |     |     |                    |     |
+  +  +  +  +  +  +  +  +--+--+--+--+--+  +--+  +
|  |  |     |  |  |     |     |        |        |
+--+  +--+--+  +  +--+--+--+  +  +--+  +--+--+  +
|     |     |  |  |  |        |     |  |        |
```

Python: Maze_solving[edit]

```
# python 3

def Dijkstra(Graph, source):
    '''
        +   +---+---+
        | 0   1   2 |
        +---+   +   +
        | 3   4 | 5
        +---+---+---+

        >>> graph = (        # or ones on the diagonal
        ...     (0,1,0,0,0,0,),
        ...     (1,0,1,0,1,0,),
        ...     (0,1,0,0,0,1,),
        ...     (0,0,0,0,1,0,),
        ...     (0,1,0,1,0,0,),
        ...     (0,0,1,0,0,0,),
        ... )
        ...
        >>> Dijkstra(graph, 0)
        ([0, 1, 2, 3, 2, 3], [1e+140, 0, 1, 4, 1, 2])
        >>> display_solution([1e+140, 0, 1, 4, 1, 2])
        5<2<1<0
    '''
    # Graph[u][v] is the weight from u to v (however 0 means infinity)
    infinity = float('infinity')
    n = len(graph)
    dist = [infinity]*n   # Unknown distance function from source to v
```

```
    previous = [infinity]*n # Previous node in optimal path from source
    dist[source] = 0        # Distance from source to source
    Q = list(range(n)) # All nodes in the graph are unoptimized - thus are
    while Q:           # The main loop
        u = min(Q, key=lambda n:dist[n])                 # vertex in Q with
        Q.remove(u)
        if dist[u] == infinity:
            break # all remaining vertices are inaccessible from source
        for v in range(n):               # each neighbor v of u
            if Graph[u][v] and (v in Q): # where v has not yet been visited
                alt = dist[u] + Graph[u][v]
                if alt < dist[v]:       # Relax (u,v,a)
                    dist[v] = alt
                    previous[v] = u
    return dist,previous

def display_solution(predecessor):
    cell = len(predecessor)-1
    while cell:
        print(cell,end='<')
        cell = predecessor[cell]
    print(0)
```

Racket[edit]

Python: Mean[edit]

Works with: Python version 3.0
.
Works with: Python version 2.6

Python: Measure_relative_performance_of_sorting_a

Works with: Python version 2.5

Examples of sorting routines[edit]

```
def builtinsort(x):
    x.sort()

def partition(seq, pivot):
```

```
   low, middle, up = [], [], []
   for x in seq:
       if x < pivot:
           low.append(x)
       elif x == pivot:
           middle.append(x)
       else:
           up.append(x)
   return low, middle, up
import random
def qsortranpart(seq):
   size = len(seq)
   if size < 2: return seq
   low, middle, up = partition(seq, random.choice(seq))
   return qsortranpart(low) + middle + qsortranpart(up)
```

Sequence generators[edit]

```
def ones(n):
    return [1]*n

def reversedrange(n):
    return reversed(range(n))

def shuffledrange(n):
    x = range(n)
    random.shuffle(x)
    return x
```

Write timings[edit]

```
def write_timings(npoints=10, maxN=10**4, sort_functions=(builtinsort,inser
                  sequence_creators = (ones, range, shuffledrange)):
   Ns = range(2, maxN, maxN//npoints)
   for sort in sort_functions:
       for make_seq in sequence_creators:
           Ts = [usec(sort, (make_seq(n),)) for n in Ns]
           writedat('%s-%s-%d-%d.xy' % (sort.__name__,  make_seq.__name__,
```

Where *writedat()* is defined in the Write float arrays to a text file, *usec(*

Plot timings[edit]

Library: matplotlib

Library: numpy

```
import operator
import numpy, pylab
def plotdd(dictplotdict):
   """See ``plot_timings()`` below."""
   symbols = ('o', '^', 'v', '<', '>', 's', '+', 'x', 'D', 'd',
              '1', '2', '3', '4', 'h', 'H', 'p', '|', '_')
   colors = list('bgrcmyk') # split string on distinct characters
   for npoints, plotdict in dictplotdict.iteritems():
       for ttle, lst in plotdict.iteritems():
           pylab.hold(False)
           for i, (label, polynom, x, y) in enumerate(sorted(lst,key=operat
               pylab.plot(x, y, colors[i % len(colors)] + symbols[i % len(s
               pylab.hold(True)
               y = numpy.polyval(polynom, x)
               pylab.plot(x, y, colors[i % len(colors)], label= '_nolegend_
           pylab.legend(loc='upper left')
           pylab.xlabel(polynom.variable)
           pylab.ylabel('log2( time in microseconds )')
           pylab.title(ttle, verticalalignment='bottom')
           figname = '_%(npoints)03d%(ttle)s' % vars()
           pylab.savefig(figname+'.png')
           pylab.savefig(figname+'.pdf')
           print figname
```

See Plot x, y arrays and Polynomial Fitting subtasks for a basic usage of *p*

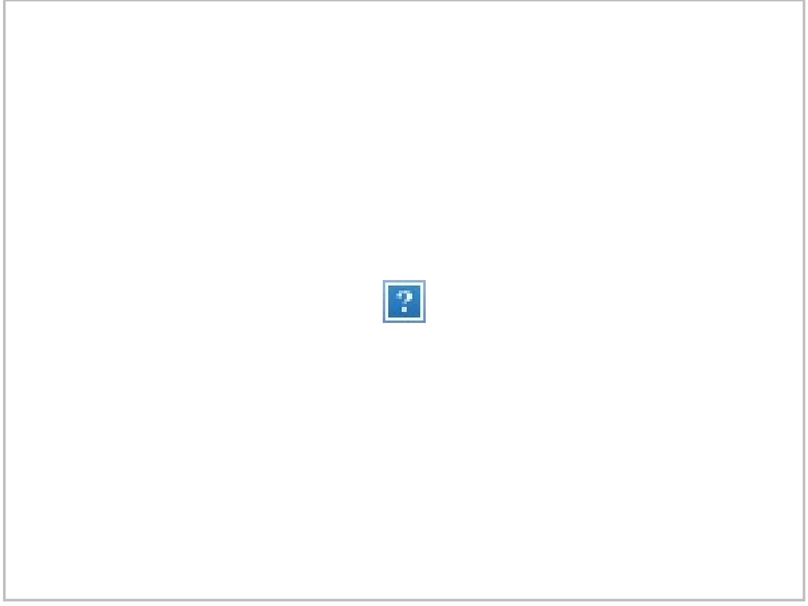
```
import collections, itertools, glob, re
import numpy
def plot_timings():
   makedict = lambda: collections.defaultdict(lambda: collections.defaultdi
   df = makedict()
   ds = makedict()
   # populate plot dictionaries
   for filename in glob.glob('*.xy'):
       m = re.match(r'([^-]+)-([^-]+)-(\d+)-(\d+)\.xy', filename)
       print filename
       assert m, filename
       funcname, seqname, npoints, maxN = m.groups()
       npoints, maxN = int(npoints), int(maxN)
       a = numpy.fromiter(itertools.imap(float, open(filename).read().split
       Ns = a[::2]  # sequences lengths
       Ts = a[1::2] # corresponding times
       assert len(Ns) == len(Ts) == npoints
       assert max(Ns) <= maxN
       #
       logsafe = numpy.logical_and(Ns>0, Ts>0)
       Ts = numpy.log2(Ts[logsafe])
```

```
        Ns = numpy.log2(Ns[logsafe])
        coeffs = numpy.polyfit(Ns, Ts, deg=1)
        poly = numpy.poly1d(coeffs, variable='log2(N)')
        #
        df[npoints][funcname].append((seqname, poly, Ns, Ts))
        ds[npoints][seqname].append((funcname, poly, Ns, Ts))
    # actual plotting
    plotdd(df)
    plotdd(ds) # see ``plotdd()`` above
```

Figures: log2(time in microseconds) vs. log2(sequence length)

log(Time) vs. log(N): Relative performance on [1]*N as an input

log(Time) vs. log(N): Relative performance on range(N) as an input



log(Time) vs. log(N): Relative performance on random permutation of range(N

```
sort_functions = [
    builtinsort,         # see implementation above
    insertion_sort,      # see [[Insertion sort]]
    insertion_sort_lowb, # ''insertion_sort'', where sequential search is r
                         #     by lower_bound() function
    qsort,               # see [[Quicksort]]
    qsortranlc,          # ''qsort'' with randomly choosen ''pivot''
                         #     and the filtering via list comprehension
    qsortranpart,        # ''qsortranlc'' with filtering via ''partition''
    qsortranpartis,      # ''qsortranpart'', where for a small input sequen
    ]                    #     ''insertion_sort'' is called
if __name__=="__main__":
   import sys
   sys.setrecursionlimit(10000)
   write_timings(npoints=100, maxN=1024, # 1 <= N <= 2**10 an input sequenc
                 sort_functions=sort_functions,
                 sequence_creators = (ones, range, shuffledrange))
   plot_timings()
```

Executing above script we get belowed figures.

Python: Median[edit]

```
def median(aray):
    srtd = sorted(aray)
    alen = len(srtd)
    return 0.5*( srtd[(alen-1)//2] + srtd[alen//2])

a = (4.1, 5.6, 7.2, 1.7, 9.3, 4.4, 3.2)
print a, median(a)
a = (4.1, 7.2, 1.7, 9.3, 4.4, 3.2)
```

```
print a, median(a)
```

Python: Median_filter[edit]

Works with: Python version 2.6

Python: Memory_Allocation[edit]

Python has the array module:
This module defines an object type which can compactly represent an array o

Python: Memory_allocation[edit]

Python has the array module:
This module defines an object type which can compactly represent an array o

ctypes type	C type	Python type

Type code	C Type	Python Type	Minimum size in bytes
'c'	char	character	1
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	long	2
'l'	signed long	int	4
'L'	unsigned long	long	4
'f'	float	float	4
'd'	double	float	8

The actual representation of values is determined by the machine architectu
(strictly speaking, by the C implementation). The actual size can be access
through the itemsize attribute. The values stored for 'L' and
'I' items will be represented as Python long integers when retrieved,

because Python’s plain integer type cannot represent the full range of C’s unsigned (long) integers.

Example

```
>>> from array import array
>>> argslist = [('l', []), ('c', 'hello world'), ('u', u'hello \u2641'),
        ('l', [1, 2, 3, 4, 5]), ('d', [1.0, 2.0, 3.14])]
>>> for typecode, initializer in argslist:
        a = array(typecode, initializer)
        print a
        del a


array('l')
array('c', 'hello world')
array('u', u'hello \u2641')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.1400000000000001])
>>>
```

Python: Memory_layout_of_a_data_structure[edit]

The ctypes module allows for the creation of Structures that can map betwee

```
from ctypes import Structure, c_int

rs232_9pin  = "_0 CD RD TD DTR SG DSR RTS CTS RI".split()
rs232_25pin = ( "_0  PG  TD  RD  RTS CTS DSR SG  CD  pos neg"
                "_11 SCD SCS STD TC  SRD RC"
                "_18 SRS DTR SQD RI DRS XTC" ).split()

class RS232_9pin(Structure):
    _fields_ = [(__, c_int, 1) for __ in rs232_9pin]


class RS232_25pin(Structure):
    _fields_ = [(__, c_int, 1) for __ in rs232_25pin]
```

Racket[edit]

Python: Menu[edit]

```
def _menu(items):
    for indexitem in enumerate(items):
        print ("  %2i) %s" % indexitem)

def _ok(reply, itemcount):
    try:
        n = int(reply)
        return 0 <= n < itemcount
    except:
        return False

def selector(items, prompt):
    'Prompt to select an item from the items'
    if not items: return ''
    reply = -1
    itemcount = len(items)
    while not _ok(reply, itemcount):
        _menu(items)
        # Use input instead of raw_input for Python 3.x
        reply = raw_input(prompt).strip()
    return items[int(reply)]

if __name__ == '__main__':
    items = ['fee fie', 'huff and puff', 'mirror mirror', 'tick tock']
    item = selector(items, 'Which is from the three pigs: ')
    print ("You chose: " + item)
```

Sample runs:

Python: Merge_sort[edit]

Works with: Python version 2.6+

```
from heapq import merge

def merge_sort(m):
    if len(m) <= 1:
        return m

    middle = len(m) // 2
    left = m[:middle]
    right = m[middle:]

    left = merge_sort(left)
    right = merge_sort(right)
```

```
    return list(merge(left, right))
```

Pre-2.6, merge() could be implemented like this:

```
def merge(left, right):
    result = []
    left_idx, right_idx = 0, 0
    while left_idx < len(left) and right_idx < len(right):
        # change the direction of this comparison to change the direction o
        if left[left_idx] <= right[right_idx]:
            result.append(left[left_idx])
            left_idx += 1
        else:
            result.append(right[right_idx])
            right_idx += 1

    if left:
        result.extend(left[left_idx:])
    if right:
        result.extend(right[right_idx:])
    return result
```

R[edit]

Python: Metered_concurrency[edit]

Python threading module includes a semaphore implementation. This code show

```
import time
import threading

# Only 4 workers can run in the same time
sem = threading.Semaphore(4)

workers = []
running = 1


def worker():
    me = threading.currentThread()
    while 1:
        sem.acquire()
        try:
```

```
            if not running:
                break
            print '%s acquired semaphore' % me.getName()
            time.sleep(2.0)
        finally:
            sem.release()
        time.sleep(0.01) # Let others acquire

# Start 10 workers
for i in range(10):
    t = threading.Thread(name=str(i), target=worker)
    workers.append(t)
    t.start()

# Main loop
try:
    while 1:
        time.sleep(0.1)
except KeyboardInterrupt:
    running = 0
    for t in workers:
        t.join()
```

Python: Metronome[edit]

```
#lang Python
import time

def main(bpm = 72, bpb = 4):
    sleep = 60.0 / bpm
    counter = 0
    while True:
        counter += 1
        if counter % bpb:
            print 'tick'
        else:
            print 'TICK'
        time.sleep(sleep)



main()
```

Python: Middle_three_digits[edit]

```
>>> def middle_three_digits(i):
	s = str(abs(i))
	length = len(s)
	assert length >= 3 and length % 2 == 1, "Need odd and >= 3 digits"
	mid = length // 2
	return s[mid-1:mid+2]

>>> passing = [123, 12345, 1234567, 987654321, 10001, -10001, -123, -100, 1
>>> failing = [1, 2, -1, -10, 2002, -2002, 0]
>>> for x in passing + failing:
	try:
		answer = middle_three_digits(x)
	except AssertionError as error:
		answer = error
	print("middle_three_digits(%s) returned: %r" % (x, answer))


middle_three_digits(123) returned: '123'
middle_three_digits(12345) returned: '234'
middle_three_digits(1234567) returned: '345'
middle_three_digits(987654321) returned: '654'
middle_three_digits(10001) returned: '000'
middle_three_digits(-10001) returned: '000'
middle_three_digits(-123) returned: '123'
middle_three_digits(-100) returned: '100'
middle_three_digits(100) returned: '100'
middle_three_digits(-12345) returned: '234'
middle_three_digits(1) returned: AssertionError('Need odd and >= 3 digits',
middle_three_digits(2) returned: AssertionError('Need odd and >= 3 digits',
middle_three_digits(-1) returned: AssertionError('Need odd and >= 3 digits'
middle_three_digits(-10) returned: AssertionError('Need odd and >= 3 digits
middle_three_digits(2002) returned: AssertionError('Need odd and >= 3 digit
middle_three_digits(-2002) returned: AssertionError('Need odd and >= 3 digi
middle_three_digits(0) returned: AssertionError('Need odd and >= 3 digits',
>>>
```

Python: Midpoint_circle_algorithm[edit]

Works with: Python version 3.1

Python: Mode[edit]

The following solutions require that the elements be *hashable*.

Works with: Python version 2.5+ and 3.x

```
>>> from collections import defaultdict
```

```
>>> def modes(values):
        count = defaultdict(int)
        for v in values:
                count[v] +=1
        best = max(count.values())
        return [k for k,v in count.items() if v == best]

>>> modes([1,3,6,6,6,6,7,7,12,12,17])
[6]
>>> modes((1,1,2,4,4))
[1, 4]
```

Works with: Python version 2.7+ and 3.1+

```
>>> from collections import Counter
>>> def modes(values):
        count = Counter(values)
        best = max(count.values())
        return [k for k,v in count.items() if v == best]

>>> modes([1,3,6,6,6,6,7,7,12,12,17])
[6]
>>> modes((1,1,2,4,4))
[1, 4]
```

If you just want one mode (instead of all of them), here's a one-liner for t

```
def onemode(values):
    return max(set(values), key=values.count)
```

Python: Modular_arithmetic[edit]

Works with: Python version 3.x

We need to implement a Modulo type first, then give one of its instances to

Thanks to duck typing, the function doesn't need to care about the actual t

```
import operator
import functools

@functools.total_ordering
class Mod:
    __slots__ = ['val','mod']
```

```
    def __init__(self, val, mod):
        if not isinstance(val, int):
            raise ValueError('Value must be integer')
        if not isinstance(mod, int) or mod<=0:
            raise ValueError('Modulo must be positive integer')
        self.val = val % mod
        self.mod = mod

    def __repr__(self):
        return 'Mod({}, {})'.format(self.val, self.mod)

    def __int__(self):
        return self.val

    def __eq__(self, other):
        if isinstance(other, Mod):
            if self.mod == other.mod:
                return self.val==other.val
            else:
                return NotImplemented
        elif isinstance(other, int):
            return self.val == other
        else:
            return NotImplemented

    def __lt__(self, other):
        if isinstance(other, Mod):
            if self.mod == other.mod:
                return self.val<other.val
            else:
                return NotImplemented
        elif isinstance(other, int):
            return self.val < other
        else:
            return NotImplemented

    def _check_operand(self, other):
        if not isinstance(other, (int, Mod)):
            raise TypeError('Only integer and Mod operands are supported')
        if isinstance(other, Mod) and self.mod != other.mod:
            raise ValueError('Inconsistent modulus: {} vs. {}'.format(self.m

    def __pow__(self, other):
        self._check_operand(other)
        # We use the built-in modular exponentiation function, this way we
        return Mod(pow(self.val, int(other), self.mod), self.mod)

    def __neg__(self):
        return Mod(self.mod - self.val, self.mod)

    def __pos__(self):
        return self # The unary plus operator does nothing.

    def __abs__(self):
        return self # The value is always kept non-negative, so the abs fun
```

```
# Helper functions to build common operands based on a template.
# They need to be implemented as functions for the closures to work properl
def _make_op(opname):
    op_fun = getattr(operator, opname)  # Fetch the operator by name from t
    def op(self, other):
        self._check_operand(other)
        return Mod(op_fun(self.val, int(other)) % self.mod, self.mod)
    return op

def _make_reflected_op(opname):
    op_fun = getattr(operator, opname)
    def op(self, other):
        self._check_operand(other)
        return Mod(op_fun(int(other), self.val) % self.mod, self.mod)
    return op

# Build the actual operator overload methods based on the template.
for opname, reflected_opname in [('__add__', '__radd__'), ('__sub__', '__rs
    setattr(Mod, opname, _make_op(opname))
    setattr(Mod, reflected_opname, _make_reflected_op(opname))

def f(x):
    return x**100+x+1

print(f(Mod(10,13)))
# Output: Mod(1, 13)
```

Racket[edit]

Python: Modular_exponentiation[edit]

```
a = 2988348162058574136915891421498819466320163312926952423791023078876139
b = 2351399303373464486466122544523690094744975233415544072992656881240319
m = 10 ** 40
print(pow(a, b, m))
```

Output:

```
1527229998585248450016808958343740453059
```

OCaml[edit]

Python: Modular_inverse[edit]

Implementation of this pseudocode with this.

```
>>> def extended_gcd(aa, bb):
    lastremainder, remainder = abs(aa), abs(bb)
    x, lastx, y, lasty = 0, 1, 1, 0
    while remainder:
        lastremainder, (quotient, remainder) = remainder, divmod(lastremain
        x, lastx = lastx - quotient*x, x
        y, lasty = lasty - quotient*y, y
    return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb 

>>> def modinv(a, m):
        g, x, y = extended_gcd(a, m)
        if g != 1:
                raise ValueError
        return x % m

>>> modinv(42, 2017)
1969
>>>
```

Racket[edit]

Python: Modulinos[edit]

Python has scripted main.

```
#!/usr/bin/env python

# life.py

def meaning_of_life():
  return 42

if __name__ == "__main__":
  print("Main: The meaning of life is %s" % meaning_of_life())
```

Python: Monte_Carlo_methods[edit]

At the interactive prompt[edit]

```
Python 2.6rc2 (r26rc2:66507, Sep 18 2008, 14:27:33) [MSC v.1500 32 bit (Int
IDLE 2.6rc2

One use of the "sum" function is to count how many times something is true
```

```
>>> import random, math
>>> throws = 1000
>>> 4.0 * sum(math.hypot(*[random.random()*2-1
                         for q in [0,1]]) < 1
              for p in xrange(throws)) / float(throws)
3.1520000000000001
>>> throws = 1000000
>>> 4.0 * sum(math.hypot(*[random.random()*2-1
                         for q in [0,1]]) < 1
              for p in xrange(throws)) / float(throws)
3.1396359999999999
>>> throws = 100000000
>>> 4.0 * sum(math.hypot(*[random.random()*2-1
                         for q in [0,1]]) < 1
              for p in xrange(throws)) / float(throws)
3.1415666400000002
```

As a program using a function[edit]

```
from random import random
from math import hypot
try:
    import psyco
    psyco.full()
except:
    pass

def pi(nthrows):
    inside = 0
    for i in xrange(nthrows):
        if hypot(random(), random()) < 1:
            inside += 1
    return 4.0 * inside / nthrows
```

```
for n in [10**4, 10**6, 10**7, 10**8]:
    print "%9d: %07f" % (n, pi(n))
```

Faster implementation using Numpy[edit]

```
import numpy as np

n = input('Number of samples: ')
print np.sum(np.random.rand(n)**2+np.random.rand(n)**2<1)/float(n)*4
```

Python: Morse_code[edit]

```
import time, winsound #, sys

char2morse = {
          "!": "---.",      "\"": ".-..-.",     "$": "...-..-",    "'": ".-
          "(": "-.--.",      ")": "-.--.-",     "+": ".-.-.",      ",": "--
          "-": "-....-",     ".": ".-.-.-",     "/": "-..-.",
          "0": "-----",      "1": ".----",      "2": "..---",      "3": "..
          "4": "....-",      "5": ".....",      "6": "-....",      "7": "--
          "8": "---..",      "9": "----.",
          ":": "---...",     ";": "-.-.-.",     "=": "-...-",      "?": "..
          "@": ".--.-.",
          "A": ".-",         "B": "-...",       "C": "-.-.",       "D": "-.
          "E": ".",          "F": "..-.",       "G": "--.",        "H": "..
          "I": "..",         "J": ".---",       "K": "-.-",        "L": ".-
          "M": "--",         "N": "-.",         "O": "---",        "P": ".-
          "Q": "--.-",       "R": ".-.",        "S": "...",        "T": "-"
          "U": "..-",        "V": "...-",       "W": ".--",        "X": "-.
          "Y": "-.--",       "Z": "--..",
          "[": "-.--.",      "]": "-.--.-",     "_": "..--.-",
 }

e = 50      # Element time in ms. one dit is on for e then off for e
f = 1280    # Tone freq. in hertz
chargap = 1 # Time between characters of a word, in units of e
wordgap = 7 # Time between words, in units of e

def gap(n=1):
    time.sleep(n * e / 1000)
off = gap

def on(n=1):
```

```
    winsound.Beep(f, n * e)

def dit():
    on(); off()

def dah():
    on(3); off()

def bloop(n=3):
    winsound.Beep(f//2, n * e)

def windowsmorse(text):
    for word in text.strip().upper().split():
        for char in word:
            for element in char2morse.get(char, '?'):
                if element == '-':
                    dah()
                elif element == '.':
                    dit()
                else:
                    bloop()
            gap(chargap)
        gap(wordgap)

# Outputs its own source file as Morse. An audible quine!
#with open(sys.argv[0], 'r') as thisfile:
#    windowsmorse(thisfile.read())

while True:
    windowsmorse(input('A string to change into morse: '))
```

Python: Most_frequent_k_chars_distance[edit]

Works with: Python version 2.7+

unoptimized and limited

```
import collections
def MostFreqKHashing(inputString, K):
    occuDict = collections.defaultdict(int)
    for c in inputString:
        occuDict[c] += 1
    occuList = sorted(occuDict.items(), key = lambda x: x[1], reverse = Tru
    outputStr = ''.join(c + str(cnt) for c, cnt in occuList[:K])
    return outputStr

#If number of occurrence of the character is not more than 9
def MostFreqKSimilarity(inputStr1, inputStr2):
```

```
    similarity = 0
    for i in range(0, len(inputStr1), 2):
        c = inputStr1[i]
        cnt1 = int(inputStr1[i + 1])
        for j in range(0, len(inputStr2), 2):
            if inputStr2[j] == c:
                cnt2 = int(inputStr2[j + 1])
                similarity += cnt1 + cnt2
                break
    return similarity

def MostFreqKSDF(inputStr1, inputStr2, K, maxDistance):
    return maxDistance - MostFreqKSimilarity(MostFreqKHashing(inputStr1,K),
```

optimized

A version that replaces the intermediate string with OrderedDict to reduce

```
import collections
def MostFreqKHashing(inputString, K):
    occuDict = collections.defaultdict(int)
    for c in inputString:
        occuDict[c] += 1
    occuList = sorted(occuDict.items(), key = lambda x: x[1], reverse = Tru
    outputDict = collections.OrderedDict(occuList[:K])
    #Return OrdredDict instead of string for faster lookup.
    return outputDict

def MostFreqKSimilarity(inputStr1, inputStr2):
    similarity = 0
    for c, cnt1 in inputStr1.items():
        #Reduce the time complexity of lookup operation to about O(1).
        if c in inputStr2:
            cnt2 = inputStr2[c]
            similarity += cnt1 + cnt2
    return similarity

def MostFreqKSDF(inputStr1, inputStr2, K, maxDistance):
    return maxDistance - MostFreqKSimilarity(MostFreqKHashing(inputStr1,K),
```

Test:

```
str1 = "LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLFSAIPYIGT
str2 = "EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKD
K = 2
maxDistance = 100
dict1 = MostFreqKHashing(str1, 2)
print("%s:"%dict1)
print(''.join(c + str(cnt) for c, cnt in dict1.items()))
```

```
dict2 = MostFreqKHashing(str2, 2)
print("%s:"%dict2)
print(''.join(c + str(cnt) for c, cnt in dict2.items()))
print(MostFreqKSDF(str1, str2, K, maxDistance))
```

Output:

```
OrderedDict([('L', 9), ('T', 8)]):
L9T8
OrderedDict([('F', 9), ('L', 8)]):
F9L8
83
```

Python: Mouse_position[edit]

Library: Python Tkinter module (Tk 8.5)

Mouse position using Tkinter graphics library nearly universally included i
There are other alternatives but they are platform specific.

Shows position of mouse while it is over the program windows and
changes color of window when mouse is near (<10) hot spot 100,100.

Code is based on post in Daniweb: http://www.daniweb.com/forums/post616327.

```
import Tkinter as tk

def showxy(event):
    xm, ym = event.x, event.y
    str1 = "mouse at x=%d  y=%d" % (xm, ym)
    # show cordinates in title
    root.title(str1)
    # switch color to red if mouse enters a set location range
    x,y, delta = 100, 100, 10
    frame.config(bg='red'
                 if abs(xm - x) < delta and abs(ym - y) < delta
                 else 'yellow')

root = tk.Tk()
frame = tk.Frame(root, bg= 'yellow', width=300, height=200)
frame.bind("<Motion>", showxy)
frame.pack()

root.mainloop()
```

```
#simple way of ,get cursor xy data
#niwantha33@gmail.com
from Tkinter import *
win=Tk()
win.geometry("200x300")
def xy(event):
    xm, ym = event.x, event.y
    xy_data = "x=%d  y=%d" % (xm, ym)
    lab=Label(win,text=xy_data)
    lab.grid(row=0,column=0)

win.bind("<Motion>",xy)
mainloop()
```

Scala[edit]

Python: Moving_Average[edit]

Works with: Python version 3.x

Both implementations use the deque datatype.

Procedural[edit]

```
from collections import deque

def simplemovingaverage(period):
    assert period == int(period) and period > 0, "Period must be an integer

    summ = n = 0.0
    values = deque([0.0] * period)     # old value queue

    def sma(x):
        nonlocal summ, n

        values.append(x)
```

```
        summ += x - values.popleft()
        n = min(n+1, period)
        return summ / n

    return sma
```

Class based[edit]

```
from collections import deque

class Simplemovingaverage():
    def __init__(self, period):
        assert period == int(period) and period > 0, "Period must be an inte
        self.period = period
        self.stream = deque()

    def __call__(self, n):
        stream = self.stream
        stream.append(n)    # appends on the right
        streamlength = len(stream)
        if streamlength > self.period:
            stream.popleft()
            streamlength -= 1
        if streamlength == 0:
            average = 0
        else:
            average = sum( stream ) / streamlength

        return average
```

Tests

```
if __name__ == '__main__':
    for period in [3, 5]:
        print ("\nSIMPLE MOVING AVERAGE (procedural): PERIOD =", period)
        sma = simplemovingaverage(period)
        for i in range(1,6):
            print ("  Next number = %-2g, SMA = %g " % (i, sma(i)))
        for i in range(5, 0, -1):
            print ("  Next number = %-2g, SMA = %g " % (i, sma(i)))
    for period in [3, 5]:
        print ("\nSIMPLE MOVING AVERAGE (class based): PERIOD =", period)
        sma = Simplemovingaverage(period)
        for i in range(1,6):
            print ("  Next number = %-2g, SMA = %g " % (i, sma(i)))
        for i in range(5, 0, -1):
            print ("  Next number = %-2g, SMA = %g " % (i, sma(i)))
```

Output:

```
SIMPLE MOVING AVERAGE (procedural): PERIOD = 3
  Next number = 1 , SMA = 1
  Next number = 2 , SMA = 1.5
  Next number = 3 , SMA = 2
  Next number = 4 , SMA = 3
  Next number = 5 , SMA = 4
  Next number = 5 , SMA = 4.66667
  Next number = 4 , SMA = 4.66667
  Next number = 3 , SMA = 4
  Next number = 2 , SMA = 3
  Next number = 1 , SMA = 2

SIMPLE MOVING AVERAGE (procedural): PERIOD = 5
  Next number = 1 , SMA = 1
  Next number = 2 , SMA = 1.5
  Next number = 3 , SMA = 2
  Next number = 4 , SMA = 2.5
  Next number = 5 , SMA = 3
  Next number = 5 , SMA = 3.8
  Next number = 4 , SMA = 4.2
  Next number = 3 , SMA = 4.2
  Next number = 2 , SMA = 3.8
  Next number = 1 , SMA = 3

SIMPLE MOVING AVERAGE (class based): PERIOD = 3
  Next number = 1 , SMA = 1
  Next number = 2 , SMA = 1.5
  Next number = 3 , SMA = 2
  Next number = 4 , SMA = 3
  Next number = 5 , SMA = 4
  Next number = 5 , SMA = 4.66667
  Next number = 4 , SMA = 4.66667
  Next number = 3 , SMA = 4
  Next number = 2 , SMA = 3
  Next number = 1 , SMA = 2

SIMPLE MOVING AVERAGE (class based): PERIOD = 5
  Next number = 1 , SMA = 1
  Next number = 2 , SMA = 1.5
  Next number = 3 , SMA = 2
  Next number = 4 , SMA = 2.5
  Next number = 5 , SMA = 3
  Next number = 5 , SMA = 3.8
  Next number = 4 , SMA = 4.2
  Next number = 3 , SMA = 4.2
  Next number = 2 , SMA = 3.8
  Next number = 1 , SMA = 3
```

Python: Multiline_shebang[edit]

We can use multiple strings to make the shell commands do nothing from Pyth

```
#!/bin/bash
"exec" "python" "$0"

print "Hello World"
```

Output:

```
$ ./myScript
Hello World
```

Control structures (if/for/etc.) can't be quoted,
but one can use the following to embed any script:

```
#!/bin/sh
"true" '''\'
if [ -L $0 ]; then
...
exec "$interpreter" "$@"
exit 127
'''

__doc__ = """module docstring"""

print "Hello World"
```

Here we use a) the code '''\' translates to \ in shell, but opens a multi-l

Python: Multiple_distinct_objects[edit]

The mistake is often written as:

```
[Foo()] * n # here Foo() can be any expression that returns a new object
```

which is incorrect since Foo() is only evaluated once. A common correct ver

```
[Foo() for i in range(n)]
```

which evaluates Foo() *n* times and collects each result in a list. This last

R[edit]

Python: Multiple_inheritance[edit]

```
class Camera:
  pass #functions go here...
```

```
class MobilePhone:
  pass #functions go here...
```

```
class CameraPhone(Camera, MobilePhone):
  pass #functions go here...
```

Racket[edit]

Python: Multiple_regression[edit]

Library: numpy

Method with matrix operations

Python: Multiplication_tables[edit]

```
>>> size = 12
>>> width = len(str(size**2))
>>> for row in range(-1,size+1):
        if row==0:
                print("─"*width + "┼"+"─"*((width+1)*size-1))
        else:
                print("".join("%*s%1s" % ((width,) + (("x","│")      if row==-1 and col==0
                                                    else (row,"│") if row>0   and col==0
                                                    else (col,"")  if row==-1
                                                    else ("","")   if row>col
                                                    else (row*col,"")))
                               for col in range(size+1)))


  x│  1   2   3   4   5   6   7   8   9  10  11  12
───┼───────────────────────────────────────────────
  1│  1   2   3   4   5   6   7   8   9  10  11  12
  2│      4   6   8  10  12  14  16  18  20  22  24
  3│          9  12  15  18  21  24  27  30  33  36
  4│             16  20  24  28  32  36  40  44  48
  5│                 25  30  35  40  45  50  55  60
  6│                     36  42  48  54  60  66  72
  7│                         49  56  63  70  77  84
  8│                             64  72  80  88  96
  9│                                 81  90  99 108
 10│                                    100 110 120
 11│                                        121 132
 12│                                            144
>>>
```

The above works with Python 3.X, which uses Unicode strings by default.

Declaring a file type of UTF-8 and adding a u to all string literals to tra
(As would using ASCII minus, plus, and pipe characters: "-", "+", "|"; instead of the non

R[edit]

Python: Multiplicative_order[edit]

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def lcm(a, b):
```

```
    return (a*b) / gcd(a, b)

def isPrime(p):
    return (p > 1) and all(f == p for f,e in factored(p))

primeList = [2,3,5,7]
def primes():
    for p in primeList:
        yield p
    while 1:
        p += 2
        while not isPrime(p):
            p += 2
        primeList.append(p)
        yield p

def factored( a):
    for p in primes():
        j = 0
        while a%p == 0:
            a /= p
            j += 1
        if j > 0:
            yield (p,j)
        if a < p*p: break
    if a > 1:
        yield (a,1)


def multOrdr1(a,(p,e) ):
    m = p**e
    t = (p-1)*(p**(e-1)) #  = Phi(p**e) where p prime
    qs = [1,]
    for f in factored(t):
        qs = [ q * f[0]**j for j in range(1+f[1]) for q in qs ]
    qs.sort()

    for q in qs:
        if pow( a, q, m )==1: break
    return q


def multOrder(a,m):
    assert gcd(a,m) == 1
    mofs = (multOrdr1(a,r) for r in factored(m))
    return reduce(lcm, mofs, 1)


if __name__ == "__main__":
    print multOrder(37, 1000)        # 100
    b = 10**20-1
    print multOrder(2, b) # 3748806900
    print multOrder(17,b) # 1499522760
    b = 100001
    print multOrder(54,b)
    print pow( 54, multOrder(54,b),b)
```

```
    if any( (1==pow(54,r, b)) for r in range(1,multOrder(54,b))):
        print 'Exists a power r < 9090 where pow(54,r,b)==1'
    else:
        print 'Everything checks.'
```

Racket[edit]

Python: Multiplies_of_3_and_5[edit]

Three ways of performing the calculation are shown including direct calcula

Python: Multisplit[edit]

Using Regular expressions[edit]

```
>>> import re
>>> def ms2(txt="a!===b=!=c", sep=["==", "!=", "="]):
        if not txt or not sep:
                return []
        ans = m = []
        for m in re.finditer('(.*?)(?:' + '|'.join('('+re.escape(s)+')' for
                ans += [m.group(1), (m.lastindex-2, m.start(m.lastindex))]
        if m and txt[m.end(m.lastindex):]:
                ans += [txt[m.end(m.lastindex):]]
        return ans

>>> ms2()
['a', (1, 1), '', (0, 3), 'b', (2, 6), '', (1, 7), 'c']
>>> ms2(txt="a!===b=!=c", sep=["=", "!=", "=="])
['a', (1, 1), '', (0, 3), '', (0, 4), 'b', (0, 6), '', (1, 7), 'c']
```

Not using RE's[edit]

Inspired by C-version

```
def multisplit(text, sep):
    lastmatch = i = 0
    matches = []
```

```
    while i < len(text):
        for j, s in enumerate(sep):
            if text[i:].startswith(s):
                if i > lastmatch:
                    matches.append(text[lastmatch:i])
                matches.append((j, i))  # Replace the string containing the
                lastmatch = i + len(s)
                i += len(s)
                break
        else:
            i += 1
    if i > lastmatch:
        matches.append(text[lastmatch:i])
    return matches

>>> multisplit('a!===b=!=c', ['==', '!=', '='])
['a', (1, 1), (0, 3), 'b', (2, 6), (1, 7), 'c']
>>> multisplit('a!===b=!=c', ['!=', '==', '='])
['a', (0, 1), (1, 3), 'b', (2, 6), (0, 7), 'c']
```

Alternative version

```
def min_pos(List):
        return List.index(min(List))

def find_all(S, Sub, Start = 0, End = -1, IsOverlapped = 0):
        Res = []
        if End == -1:
                End = len(S)
        if IsOverlapped:
                DeltaPos = 1
        else:
                DeltaPos = len(Sub)
        Pos = Start
        while True:
                Pos = S.find(Sub, Pos, End)
                if Pos == -1:
                        break
                Res.append(Pos)
                Pos += DeltaPos
        return Res

def multisplit(S, SepList):
        SepPosListList = []
        SLen = len(S)
        SepNumList = []
        ListCount = 0
        for i, Sep in enumerate(SepList):
                SepPosList = find_all(S, Sep, 0, SLen, IsOverlapped = 1)
                if SepPosList != []:
                        SepNumList.append(i)
```

```
                        SepPosListList.append(SepPosList)
                        ListCount += 1
        if ListCount == 0:
                return [S]
        MinPosList = []
        for i in range(ListCount):
                MinPosList.append(SepPosListList[i][0])
        SepEnd = 0
        MinPosPos = min_pos(MinPosList)
        Res = []
        while True:
                Res.append( S[SepEnd : MinPosList[MinPosPos]] )
                Res.append([SepNumList[MinPosPos], MinPosList[MinPosPos]])
                SepEnd = MinPosList[MinPosPos] + len(SepList[SepNumList[Min
                while True:
                        MinPosPos = min_pos(MinPosList)
                        if MinPosList[MinPosPos] < SepEnd:
                                del SepPosListList[MinPosPos][0]
                                if len(SepPosListList[MinPosPos]) == 0:
                                        del SepPosListList[MinPosPos]
                                        del MinPosList[MinPosPos]
                                        del SepNumList[MinPosPos]
                                        ListCount -= 1
                                        if ListCount == 0:
                                                break
                                else:
                                        MinPosList[MinPosPos] = SepPosListL
                        else:
                                break
                if ListCount == 0:
                        break
        Res.append(S[SepEnd:])
        return Res


S = "a!===b=!=c"
multisplit(S, ["==", "!=", "="]) # output: ['a', [1, 1], '', [0, 3], 'b', [
multisplit(S, ["=", "!=", "=="]) # output: ['a', [1, 1], '', [0, 3], '', [0
```

Racket[edit]

Python: Munching_squares[edit]

Library: PIL

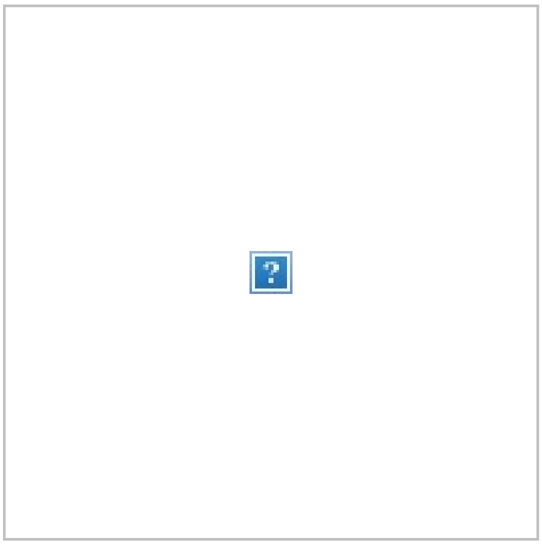
```
import Image, ImageDraw

image = Image.new("RGB", (256, 256))
```

```
drawingTool = ImageDraw.Draw(image)

for x in range(256):
    for y in range(256):
        drawingTool.point((x, y), (0, x^y, 0))

del drawingTool
image.save("xorpic.png", "PNG")
```



[Racket](#)[[edit](#)]

[Python: Musical_scale](#)[[edit](#)]

(Windows)

```
>>> import winsound
>>> for note in [261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 52
        winsound.Beep(int(note+.5), 500)
>>>
```

[Racket](#)[[edit](#)]

[Python: Mutex](#)[[edit](#)]

Demonstrating semaphores.
Note that semaphores can be considered as a multiple version of mutex;
while a mutex allows a singular exclusive access to code or resources,

Python: Mutual_Recursion[edit]

Works with: Python version 3.0
.
Works with: Python version 2.6

```
def F(n): return 1 if n == 0 else n - M(F(n-1))
def M(n): return 0 if n == 0 else n - F(M(n-1))

print ([ F(n) for n in range(20) ])
print ([ M(n) for n in range(20) ])
```

Output:

```
[1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 6, 7, 8, 8, 9, 9, 10, 11, 11, 12]
[0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 6, 7, 7, 8, 9, 9, 10, 11, 11, 12]
```

Python: Mutual_recursion[edit]

Works with: Python version 3.0
.
Works with: Python version 2.6

```
def F(n): return 1 if n == 0 else n - M(F(n-1))
def M(n): return 0 if n == 0 else n - F(M(n-1))

print ([ F(n) for n in range(20) ])
print ([ M(n) for n in range(20) ])
```

Output:

```
[1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 6, 7, 8, 8, 9, 9, 10, 11, 11, 12]
[0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 6, 7, 7, 8, 9, 9, 10, 11, 11, 12]
```

In python there is no need to pre-declare *M* for it to be used in the defini

Python: N'th[edit]

```
_suffix = ['th', 'st', 'nd', 'rd', 'th', 'th', 'th', 'th', 'th', 'th']

def nth(n):
    return "%i'%s" % (n, _suffix[n%10] if n % 100 <= 10 or n % 100 > 20 els

if __name__ == '__main__':
    for j in range(0,1001, 250):
        print(' '.join(nth(i) for i in list(range(j, j+25))))
```

Output:

```
0'th 1'st 2'nd 3'rd 4'th 5'th 6'th 7'th 8'th 9'th 10'th 11'th 12'th 13'th 1
250'th 251'st 252'nd 253'rd 254'th 255'th 256'th 257'th 258'th 259'th 260't
500'th 501'st 502'nd 503'rd 504'th 505'th 506'th 507'th 508'th 509'th 510't
750'th 751'st 752'nd 753'rd 754'th 755'th 756'th 757'th 758'th 759'th 760't
1000'th 1001'st 1002'nd 1003'rd 1004'th 1005'th 1006'th 1007'th 1008'th 100
```

Alternate version

```
#!/usr/bin/env python3

def ord(n):
    try:
        s = ['st', 'nd', 'rd'][(n-1)%10]
        if (n-10)%100//10:
            return str(n)+s
    except IndexError:
        pass
    return str(n)+'th'

if __name__ == '__main__':
    print(*(ord(n) for n in range(26)))
    print(*(ord(n) for n in range(250,266)))
    print(*(ord(n) for n in range(1000,1026)))
```

Output:

```
0th 1st 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th 12th 13th 14th 15th 16th
18th 19th 20th 21st 22nd 23rd 24th 25th
250th 251st 252nd 253rd 254th 255th 256th 257th 258th 259th 260th 261st 262
263rd 264th 265th
1000th 1001st 1002nd 1003rd 1004th 1005th 1006th 1007th 1008th 1009th 1010t
1011th 1012th 1013th 1014th 1015th 1016th 1017th 1018th 1019th 1020th 1021s
1022nd 1023rd 1024th 1025th
```

Python: NYSIIS[edit]

A literal translation of the algorithm from the Wikipedia article.

```
import re

_vowels = 'AEIOU'

def replace_at(text, position, fromlist, tolist):
    for f, t in zip(fromlist, tolist):
        if text[position:].startswith(f):
            return ''.join([text[:position],
                            t,
                            text[position+len(f):]])
    return text

def replace_end(text, fromlist, tolist):
    for f, t in zip(fromlist, tolist):
        if text.endswith(f):
            return text[:-len(f)] + t
    return text

def nysiis(name):
    name = re.sub(r'\W', '', name).upper()
    name = replace_at(name, 0, ['MAC', 'KN', 'K', 'PH', 'PF', 'SCH'],
                               ['MCC', 'N',  'C', 'FF', 'FF', 'SSS'])
    name = replace_end(name, ['EE', 'IE', 'DT', 'RT', 'RD', 'NT', 'ND'],
                             ['Y',  'Y',  'D',  'D',  'D',  'D',  'D'])
    key, key1 = name[0], ''
    i = 1
    while i < len(name):
        #print(i, name, key1, key)
        n_1, n = name[i-1], name[i]
        n1_ = name[i+1] if i+1 < len(name) else ''
        name = replace_at(name, i, ['EV'] + list(_vowels), ['AF'] + ['A']*5
        name = replace_at(name, i, 'QZM', 'GSN')
        name = replace_at(name, i, ['KN', 'K'], ['N', 'C'])
        name = replace_at(name, i, ['SCH', 'PH'], ['SSS', 'FF'])
```

```
        if n == 'H' and (n_1 not in _vowels or n1_ not in _vowels):
            name = ''.join([name[:i], n_1, name[i+1:]])
        if n == 'W' and n_1 in _vowels:
            name = ''.join([name[:i], 'A', name[i+1:]])
        if key and key[-1] != name[i]:
            key += name[i]
        i += 1
    key = replace_end(key, ['S', 'AY', 'A'], ['', 'Y', ''])
    return key1 + key

if __name__ == '__main__':
    names = ['Bishop', 'Carlson', 'Carr', 'Chapman', 'Franklin',
             'Greene', 'Harper', 'Jacobs', 'Larson', 'Lawrence',
             'Lawson', 'Louis, XVI', 'Lynch', 'Mackenzie', 'Matthews',
             'McCormack', 'McDaniel', 'McDonald', 'Mclaughlin', 'Morrison',
             "O'Banion", "O'Brien", 'Richards', 'Silva', 'Watkins',
             'Wheeler', 'Willis', 'brown, sr', 'browne, III', 'browne, IV',
             'knight', 'mitchell', "o'daniel"]
    for name in names:
        print('%15s: %s' % (name, nysiis(name)))
```

Output:

```
         Bishop: BASAP
        Carlson: CARLSAN
           Carr: CAR
        Chapman: CAPNAN
       Franklin: FRANCLAN
         Greene: GRAN
         Harper: HARPAR
         Jacobs: JACAB
         Larson: LARSAN
       Lawrence: LARANC
         Lawson: LASAN
     Louis, XVI: LASXV
          Lynch: LYNC
      Mackenzie: MCANSY
       Matthews: MATA
      McCormack: MCARNAC
       McDaniel: MCDANAL
       McDonald: MCDANALD
     Mclaughlin: MCLAGLAN
       Morrison: MARASAN
       O'Banion: OBANAN
        O'Brien: OBRAN
       Richards: RACARD
          Silva: SALV
        Watkins: WATCAN
        Wheeler: WALAR
         Willis: WALA
      brown, sr: BRANSR
```

```
    browne, III: BRAN
     browne, IV: BRANAV
         knight: NAGT
       mitchell: MATCAL
       o'daniel: ODANAL
```

Python: N_distinct_objects[edit]

The mistake is often written as:

```
[Foo()] * n # here Foo() can be any expression that returns a new object
```

which is incorrect since Foo() is only evaluated once. A common correct ver

```
[Foo() for i in range(n)]
```

which evaluates Foo() *n* times and collects each result in a list. This last

R[edit]

Python: Named_Arguments[edit]

Basic explanation[edit]

A more detailed explanation of parameters, arguments, and how they are used

In Python, a regular parameter of a function can be used as *either a positi*

```
def subtract(x, y):
    return x - y

subtract(5, 3)         # used as positional parameters; evaluates to 2
subtract(y = 3, x = 5) # used as named parameters;      evaluates to 2
```

Parameters can be made optional by providing a default argument, as describ

Detailed Explanation[edit]

Python: Named_parameters[edit]

Basic explanation[edit]

A more detailed explanation of parameters, arguments, and how they are used

In Python, a regular parameter of a function can be used as *either a positi*

```
def subtract(x, y):
    return x - y

subtract(5, 3)         # used as positional parameters; evaluates to 2
subtract(y = 3, x = 5) # used as named parameters;      evaluates to 2
```

Parameters can be made optional by providing a default argument, as describ

Detailed Explanation[edit]

Function Definition Parameters[edit]

Function definitions in Python allow for the following *parameter* types:

- Optional *default parameter* types which are explicitly specified by na
- An optional *positional parameter* which is an identifier preceded by "
- And an optional *keyword parameter* which is an identifier preceded by

If any of the parameter types are given then they must appear in the order

The syntax of function parameter declarations is more formally defined as:

```
funcdef        ::=  "def" funcname "(" [parameter_list] ")" ":" suite
dotted_name    ::=  identifier ("." identifier)*
parameter_list ::=  (defparameter ",")*
                    (  posparameter [, keyparameter]
                    | keyparameter
                    | defparameter [","] )
defparameter   ::=  parameter ["=" expression]
posparameter   ::=  "*" identifier
keyparameter   ::=  "**" identifier
sublist        ::=  parameter ("," parameter)* [","]
parameter      ::=  identifier | "(" sublist ")"
```

Function Call Arguments[edit]

The call of a function in python can use the following *argument* types:

- *Positional arguments* that are mapped by their position in the call ar
- *Sequence arguments* that are the character "*" followed by an expressi
- All positional arguments must appear before any keyword argument.
- *Keyword arguments* of the form parameter_name "=" value will map the v
- *Mapping arguments* that are the characters "**" followed by an express
- If the function *definition* includes a *positional parameter*, then if t
- If the function *definition* includes a *keyword parameter*, then if the
- Any *default parameter* of the function *definition* that is not assigned
- Any *default parameter* of the function *definition* that is still un-ass
- In addition, multiple mappings to any parameter will raise a TypeErro

The more formal definition of a function call's syntax is

```
call               ::=  primary "(" [argument_list [","]
                        | expression genexpr_for] ")"
argument_list      ::=  positional_arguments ["," keyword_arguments]
```

```
                            ["," sequence_argument] ["," keyword_arguments]
                            ["," mapping_argument]
                          | keyword_arguments ["," sequence_argument]
                            ["," mapping_argument]
                          | sequence_argument ["," sequence_argument] ["," 
                          | mapping_argument
positional_arguments ::=  expression ("," expression)*
keyword_arguments    ::=  keyword_item ("," keyword_item)*
sequence_argument    ::=  "*" expression
mapping_argument     ::=  "**" expression
keyword_item         ::=  identifier "=" expression
```

Examples[edit]

```
>>> from __future__ import print_function
>>>
>>> def show_args(defparam1, defparam2 = 'default value', *posparam, **keyp
  "Straight-forward function to show its arguments"
  print ("  Default Parameters:")
  print ("    defparam1 value is:", defparam1)
  print ("    defparam2 value is:", defparam2)

  print ("  Positional Arguments:")
  if posparam:
    n = 0
    for p in posparam:
      print ("    positional argument:", n, "is:", p)
      n += 1
  else:
    print ("    <None>")

  print ("  Keyword Arguments (by sorted key name):")
  if keyparam:
    for k,v in sorted(keyparam.items()):
      print ("    keyword argument:", k, "is:", v)
  else:
    print ("    <None>")


>>> show_args('POSITIONAL', 'ARGUMENTS')
  Default Parameters:
    defparam1 value is: POSITIONAL
    defparam2 value is: ARGUMENTS
  Positional Arguments:
    <None>
  Keyword Arguments (by sorted key name):
    <None>
>>> show_args(defparam2='ARGUMENT', defparam1='KEYWORD')
  Default Parameters:
    defparam1 value is: KEYWORD
    defparam2 value is: ARGUMENT
  Positional Arguments:
```

```
    <None>
  Keyword Arguments (by sorted key name):
    <None>
>>> show_args( *('SEQUENCE', 'ARGUMENTS') )
  Default Parameters:
    defparam1 value is: SEQUENCE
    defparam2 value is: ARGUMENTS
  Positional Arguments:
    <None>
  Keyword Arguments (by sorted key name):
    <None>
>>> show_args( **{'defparam2':'ARGUMENTS', 'defparam1':'MAPPING'} )
  Default Parameters:
    defparam1 value is: MAPPING
    defparam2 value is: ARGUMENTS
  Positional Arguments:
    <None>
  Keyword Arguments (by sorted key name):
    <None>
>>> show_args('ONLY DEFINE defparam1 ARGUMENT')
  Default Parameters:
    defparam1 value is: ONLY DEFINE defparam1 ARGUMENT
    defparam2 value is: default value
  Positional Arguments:
    <None>
  Keyword Arguments (by sorted key name):
    <None>
>>> show_args('POSITIONAL', 'ARGUMENTS',
              'EXTRA', 'POSITIONAL', 'ARGUMENTS')
  Default Parameters:
    defparam1 value is: POSITIONAL
    defparam2 value is: ARGUMENTS
  Positional Arguments:
    positional argument: 0 is: EXTRA
    positional argument: 1 is: POSITIONAL
    positional argument: 2 is: ARGUMENTS
  Keyword Arguments (by sorted key name):
    <None>
>>> show_args('POSITIONAL', 'ARGUMENTS',
              kwa1='EXTRA', kwa2='KEYWORD', kwa3='ARGUMENTS')
  Default Parameters:
    defparam1 value is: POSITIONAL
    defparam2 value is: ARGUMENTS
  Positional Arguments:
    <None>
  Keyword Arguments (by sorted key name):
    keyword argument: kwa1 is: EXTRA
    keyword argument: kwa2 is: KEYWORD
    keyword argument: kwa3 is: ARGUMENTS
>>> show_args('POSITIONAL',
              'ARGUMENTS', 'EXTRA', 'POSITIONAL', 'ARGUMENTS',
              kwa1='EXTRA', kwa2='KEYWORD', kwa3='ARGUMENTS')
  Default Parameters:
    defparam1 value is: POSITIONAL
    defparam2 value is: ARGUMENTS
  Positional Arguments:
```

```
    positional argument: 0 is: EXTRA
    positional argument: 1 is: POSITIONAL
    positional argument: 2 is: ARGUMENTS
  Keyword Arguments (by sorted key name):
    keyword argument: kwa1 is: EXTRA
    keyword argument: kwa2 is: KEYWORD
    keyword argument: kwa3 is: ARGUMENTS
>>> # But note:
>>> show_args('POSITIONAL', 'ARGUMENTS',
              kwa1='EXTRA', kwa2='KEYWORD', kwa3='ARGUMENTS',
              'EXTRA', 'POSITIONAL', 'ARGUMENTS')
SyntaxError: non-keyword arg after keyword arg
>>>
```

Python: Names_to_numbers[edit]

This example assumes that the module from Number_names#Python is stored as

The example understands the textual format generated from number-to-names m

Note: This example and Number_names#Python need to be kept in sync

```
from spell_integer import spell_integer, SMALL, TENS, HUGE

def int_from_words(num):
    words = num.replace(',','').replace(' and ', ' ').replace('-', ' ').spl
    if words[0] == 'minus':
        negmult = -1
        words.pop(0)
    else:
        negmult = 1
    small, total = 0, 0
    for word in words:
        if word in SMALL:
            small += SMALL.index(word)
        elif word in TENS:
            small += TENS.index(word) * 10
        elif word == 'hundred':
            small *= 100
        elif word == 'thousand':
            total += small * 1000
            small = 0
        elif word in HUGE:
            total += small * 1000 ** HUGE.index(word)
            small = 0
        else:
            raise ValueError("Don't understand %r part of %r" % (word, num)
    return negmult * (total + small)
```

```
if __name__ == '__main__':
    # examples
    for n in range(-10000, 10000, 17):
        assert n == int_from_words(spell_integer(n))

    for n in range(20):
        assert 13**n == int_from_words(spell_integer(13**n))

    print('\n##\n## These tests show <==> for a successful round trip, othe
    for n in (0, -3, 5, -7, 11, -13, 17, -19, 23, -29):
        txt = spell_integer(n)
        num = int_from_words(txt)
        print('%+4i <%s> %s' % (n, '==' if n == num else '??', txt))
    print('')

    n = 201021002001
    while n:
        txt = spell_integer(n)
        num = int_from_words(txt)
        print('%12i <%s> %s' % (n, '==' if n == num else '??', txt))
        n //= -10
    txt = spell_integer(n)
    num = int_from_words(txt)
    print('%12i <%s> %s' % (n, '==' if n == num else '??', txt))
    print('')
```

Output:

```
##
## These tests show <==> for a successful round trip, otherwise <??>
##

  +0 <==> zero
  -3 <==> minus three
  +5 <==> five
  -7 <==> minus seven
 +11 <==> eleven
 -13 <==> minus thirteen
 +17 <==> seventeen
 -19 <==> minus nineteen
 +23 <==> twenty-three
 -29 <==> minus twenty-nine

201021002001 <==> two hundred and one billion, twenty-one million, two thou
-20102100201 <==> minus twenty billion, one hundred and two million, one hu
  2010210020 <==> two billion, ten million, two hundred and ten thousand, a
  -201021002 <==> minus two hundred and one million, twenty-one thousand, a
    20102100 <==> twenty million, one hundred and two thousand, and one hun
    -2010210 <==> minus two million, ten thousand, two hundred and ten
      201021 <==> two hundred and one thousand, and twenty-one
      -20103 <==> minus twenty thousand, one hundred and three
```

```
        2010 <==> two thousand, and ten
        -201 <==> minus two hundred and one
          20 <==> twenty
          -2 <==> minus two
           0 <==> zero
```

Racket[edit]

Python: Naming_conventions[edit]

- Class names are typically in CamelCase, often this is reflected in th
- Private member functions are embeded between "__" to make a member fu
- Variables are generally lower-case.

Racket[edit]

Python: Narcissist[edit]

For Python 2.x:

```
import sys
with open(sys.argv[0]) as quine:
    code = raw_input("Enter source code: ")
    if code == quine.read():
        print("Accept")
    else:
        print("Reject")
```

Python: Native_shebang[edit]

Extract: "If you need to create a .pyc file for a module that is not import

```
>>> import py_compile
>>> py_compile.compile('echo.py')
```

File: echo.py

```
#!/path/to/python
# Although `#!/usr/bin/env python` may be better if the path to python can

import sys
print " ".join(sys.argv[1:])
```

Usage:

```
./echo.py Hello, world!
```

Output:

```
Hello, world!
```

Python: Natural_sorting[edit]

All eight features:

```
# -*- coding: utf-8 -*-
# Not Python 3.x (Can't compare str and int)


from itertools import groupby
from unicodedata import decomposition, name
from pprint import pprint as pp

commonleaders = ['the'] # lowercase leading words to ignore
replacements = {u'ß': 'ss',  # Map single char to replacement string
                u'ſ': 's',
                u'ʒ': 's',
                }
```

```
hexdigits = set('0123456789abcdef')
decdigits = set('0123456789')   # Don't use str.isnumeric

def splitchar(c):
    ' De-ligature. De-accent a char'
    de = decomposition(c)
    if de:
        # Just the words that are also hex numbers
        de = [d for d in de.split()
                  if all(c.lower()
                         in hexdigits for c in d)]
        n = name(c, c).upper()
        # (Gosh it's onerous)
        if len(de)> 1 and 'PRECEDE' in n:
            # E.g. ŉ  LATIN SMALL LETTER N PRECEDED BY APOSTROPHE
            de[1], de[0] = de[0], de[1]
        tmp = [ unichr(int(k, 16)) for k in de]
        base, others = tmp[0], tmp[1:]
        if 'LIGATURE' in n:
            # Assume two character ligature
            base += others.pop(0)
    else:
        base = c
    return base


def sortkeygen(s):
    '''Generate 'natural' sort key for s

    Doctests:
        >>> sortkeygen('  some extra    spaces  ')
        [u'some extra spaces']
        >>> sortkeygen('CasE InseNsItIve')
        [u'case insensitive']
        >>> sortkeygen('The Wind in the Willows')
        [u'wind in the willows']
        >>> sortkeygen(u'\462 ligature')
        [u'ij ligature']
        >>> sortkeygen(u'\335\375 upper/lower case Y with acute accent')
        [u'yy upper/lower case y with acute accent']
        >>> sortkeygen('foo9.txt')
        [u'foo', 9, u'.txt']
        >>> sortkeygen('x9y99')
        [u'x', 9, u'y', 99]
    '''
    # Ignore leading and trailing spaces
    s = unicode(s).strip()
    # All space types are equivalent
    s = ' '.join(s.split())
    # case insentsitive
    s = s.lower()
    # Title
    words = s.split()
    if len(words) > 1 and words[0] in commonleaders:
        s = ' '.join( words[1:])
```

```
    # accent and ligatures
    s = ''.join(splitchar(c) for c in s)
    # Replacements (single char replaced by one or more)
    s = ''.join( replacements.get(ch, ch) for ch in s )
    # Numeric sections as numerics
    s = [ int("".join(g)) if isinteger else "".join(g)
          for isinteger,g in groupby(s, lambda x: x in decdigits)]

    return s

def naturalsort(items):
    ''' Naturally sort a series of strings

    Doctests:
        >>> naturalsort(['The Wind in the Willows','The 40th step more',
                         'The 39 steps', 'Wanda'])
        ['The 39 steps', 'The 40th step more', 'Wanda', 'The Wind in the Wi

    '''
    return sorted(items, key=sortkeygen)

if __name__ == '__main__':
    import string

    ns = naturalsort

    print '\n# Ignoring leading spaces'
    txt = ['%signore leading spaces: 2%+i' % (' '*i, i-2) for i in range(4)
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Ignoring multiple adjacent spaces (m.a.s)'
    txt = ['ignore m.a.s%s spaces: 2%+i' % (' '*i, i-2) for i in range(4)]
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Equivalent whitespace characters'
    txt = ['Equiv.%sspaces: 3%+i' % (ch, i-3)
           for i,ch in enumerate(reversed(string.whitespace))]
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Case Indepenent sort'
    s = 'CASE INDEPENENT'
    txt = [s[:i].lower() + s[i:] + ': 3%+i' % (i-3) for i in range(1,5)]
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Numeric fields as numerics'
    txt = ['foo100bar99baz0.txt', 'foo100bar10baz0.txt',
           'foo1000bar99baz10.txt', 'foo1000bar99baz9.txt']
    print 'Text strings:'; pp(txt)
```

```
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Title sorts'
    txt = ['The Wind in the Willows','The 40th step more',
                         'The 39 steps', 'Wanda']
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Equivalent accented characters (and case)'
    txt = ['Equiv. %s accents: 2%+i' % (ch, i-2)
           for i,ch in enumerate(u'\xfd\xddyY')]
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Separated ligatures'
    txt = [u'\462 ligatured ij', 'no ligature',]
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; pp(sorted(txt))
    print 'Naturally sorted:'; pp(ns(txt))

    print '\n# Character replacements'
    s = u'ʒſßs' # u'\u0292\u017f\xdfs'
    txt = ['Start with an %s: 2%+i' % (ch, i-2)
           for i,ch in enumerate(s)]
    print 'Text strings:'; pp(txt)
    print 'Normally sorted :'; print '\n'.join(sorted(txt))
    print 'Naturally sorted:'; print '\n'.join(ns(txt))
```

Sample Python output[edit]

```
# Ignoring leading spaces
Text strings:
['ignore leading spaces: 2-2',
 ' ignore leading spaces: 2-1',
 '  ignore leading spaces: 2+0',
 '   ignore leading spaces: 2+1']
Normally sorted :
['   ignore leading spaces: 2+1',
 '  ignore leading spaces: 2+0',
 ' ignore leading spaces: 2-1',
 'ignore leading spaces: 2-2']
Naturally sorted:
['  ignore leading spaces: 2+0',
 '   ignore leading spaces: 2+1',
```

Python: Nautical_bell[edit]

As well as typing output to stdout, this program plays a sound for each bel

```
import time, calendar, sched, winsound

duration = 750      # Bell duration in ms
freq = 1280         # Bell frequency in hertz
bellchar = "\u2407"
watches = 'Middle,Morning,Forenoon,Afternoon,First/Last dog,First'.split(',

def gap(n=1):
    time.sleep(n * duration / 1000)
off = gap

def on(n=1):
    winsound.Beep(freq, n * duration)

def bong():
    on(); off(0.5)

def bongs(m):
    for i in range(m):
        print(bellchar, end=' ')
        bong()
        if i % 2:
            print('  ', end='')
            off(0.5)
    print('')

scheds =  sched.scheduler(time.time, time.sleep)

def ships_bell(now=None):
    def adjust_to_half_hour(atime):
        atime[4] = (atime[4] // 30) * 30
        atime[5] = 0
        return atime

    debug = now is not None
    rightnow = time.gmtime()
    if not debug:
        now = adjust_to_half_hour( list(rightnow) )
    then = now[::]
    then[4] += 30
    hr, mn = now[3:5]
    watch, b = divmod(int(2 * hr + mn // 30 - 1), 8)
    b += 1
    bells = '%i bell%s' % (b, 's' if b > 1 else ' ')
    if debug:
        print("%02i:%02i, %-20s %s" % (now[3], now[4], watches[watch] + ' w
    else:
        print("%02i:%02i, %-20s %s" % (rightnow[3], rightnow[4], watches[wa
    bongs(b)
    if not debug:
        scheds.enterabs(calendar.timegm(then), 0, ships_bell)
        #print(time.struct_time(then))
```

```
        scheds.run()

def dbg_tester():
    for h in range(24):
        for m in (0, 30):
            if (h,m) == (24,30): break
            ships_bell( [2013, 3, 2, h, m, 15, 5, 61, 0] )


if __name__ == '__main__':
    ships_bell()
```

Output:

```
00:00, First watch          8 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇ ␇
00:30, Middle watch         1 bell  ␇
01:00, Middle watch         2 bells ␇ ␇
01:30, Middle watch         3 bells ␇ ␇   ␇
02:00, Middle watch         4 bells ␇ ␇   ␇ ␇
02:30, Middle watch         5 bells ␇ ␇   ␇ ␇   ␇
03:00, Middle watch         6 bells ␇ ␇   ␇ ␇   ␇ ␇
03:30, Middle watch         7 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇
04:00, Middle watch         8 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇ ␇
04:30, Morning watch        1 bell  ␇
05:00, Morning watch        2 bells ␇ ␇
05:30, Morning watch        3 bells ␇ ␇   ␇
06:00, Morning watch        4 bells ␇ ␇   ␇ ␇
06:30, Morning watch        5 bells ␇ ␇   ␇ ␇   ␇
07:00, Morning watch        6 bells ␇ ␇   ␇ ␇   ␇ ␇
07:30, Morning watch        7 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇
08:00, Morning watch        8 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇ ␇
08:30, Forenoon watch       1 bell  ␇
09:00, Forenoon watch       2 bells ␇ ␇
09:30, Forenoon watch       3 bells ␇ ␇   ␇
10:00, Forenoon watch       4 bells ␇ ␇   ␇ ␇
10:30, Forenoon watch       5 bells ␇ ␇   ␇ ␇   ␇
11:00, Forenoon watch       6 bells ␇ ␇   ␇ ␇   ␇ ␇
11:30, Forenoon watch       7 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇
12:00, Forenoon watch       8 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇ ␇
12:30, Afternoon watch      1 bell  ␇
13:00, Afternoon watch      2 bells ␇ ␇
13:30, Afternoon watch      3 bells ␇ ␇   ␇
14:00, Afternoon watch      4 bells ␇ ␇   ␇ ␇
14:30, Afternoon watch      5 bells ␇ ␇   ␇ ␇   ␇
15:00, Afternoon watch      6 bells ␇ ␇   ␇ ␇   ␇ ␇
15:30, Afternoon watch      7 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇
16:00, Afternoon watch      8 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇ ␇
16:30, First/Last dog watch 1 bell  ␇
17:00, First/Last dog watch 2 bells ␇ ␇
17:30, First/Last dog watch 3 bells ␇ ␇   ␇
18:00, First/Last dog watch 4 bells ␇ ␇   ␇ ␇
```

```
18:30, First/Last dog watch 5 bells ␇ ␇   ␇ ␇   ␇
19:00, First/Last dog watch 6 bells ␇ ␇   ␇ ␇   ␇ ␇
19:30, First/Last dog watch 7 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇
20:00, First/Last dog watch 8 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇ ␇
20:30, First watch          1 bell  ␇
21:00, First watch          2 bells ␇ ␇
21:30, First watch          3 bells ␇ ␇   ␇
22:00, First watch          4 bells ␇ ␇   ␇ ␇
22:30, First watch          5 bells ␇ ␇   ␇ ␇   ␇
23:00, First watch          6 bells ␇ ␇   ␇ ␇   ␇ ␇
23:30, First watch          7 bells ␇ ␇   ␇ ␇   ␇ ␇   ␇
```

Racket[edit]

Python: Nil[edit]

```
x = None
if x is None:
  print "x is None"
else:
  print "x is not None"
```

Output:

```
x is None
```

Python: Non-continuous_subsequences[edit]

Translation of: Scheme

```
def ncsub(seq, s=0):
    if seq:
        x = seq[:1]
        xs = seq[1:]
        p2 = s % 2
        p1 = not p2
        return [x + ys for ys in ncsub(xs, s + p1)] + ncsub(xs, s + p2)
    else:
        return [[]] if s >= 3 else []
```

Output:

```
>>> ncsub(range(1, 4))
[[1, 3]]
>>> ncsub(range(1, 5))
[[1, 2, 4], [1, 3, 4], [1, 3], [1, 4], [2, 4]]
>>> ncsub(range(1, 6))
[[1, 2, 3, 5], [1, 2, 4, 5], [1, 2, 4], [1, 2, 5], [1, 3, 4, 5], [1, 3, 4],
 [1, 3, 5], [1, 3], [1, 4, 5], [1, 4], [1, 5], [2, 3, 5], [2, 4, 5], [2, 4]
 [2, 5], [3, 5]]
```

A faster Python + Psyco JIT version:

```
from sys import argv
import psyco

def C(n, k):
    result = 1
    for d in xrange(1, k+1):
        result *= n
        n -= 1
        result /= d
    return result

# http://oeis.org/A002662
nsubs = lambda n: sum(C(n, k) for k in xrange(3, n+1))

def ncsub(seq):
    n = len(seq)
    result = [None] * nsubs(n)
    pos = 0

    for i in xrange(1, 2 ** n):
        S  = []
        nc = False
        for j in xrange(n + 1):
            k = i >> j
            if k == 0:
                if nc:
                    result[pos] = S
                    pos += 1
                break
            elif k % 2:
                S.append(seq[j])
            elif S:
                nc = True
    return result

from sys import argv
import psyco
```

```
psyco.full()
n = 10 if len(argv) < 2 else int(argv[1])
print len( ncsub(range(1, n)) )
```

Python: Nonoblock[edit]

```
def nonoblocks(blocks, cells):
    if not blocks or blocks[0] == 0:
        yield [(0, 0)]
    else:
        assert sum(blocks) + len(blocks)-1 <= cells, \
            'Those blocks will not fit in those cells'
        blength, brest = blocks[0], blocks[1:]      # Deal with the first b
        minspace4rest = sum(1+b for b in brest)     # The other blocks need
        # Slide the start position from left to max RH index allowing for o
        for bpos in range(0, cells - minspace4rest - blength + 1):
            if not brest:
                # No other blocks to the right so just yield this one.
                yield [(bpos, blength)]
            else:
                # More blocks to the right so create a *sub-problem* of pla
                # the brest blocks in the cells one space to the right of t
                # this block.
                offset = bpos + blength +1
                nonoargs = (brest, cells - offset)  # Pre-compute arguments
                # recursive call to nonoblocks yields multiple sub-position
                for subpos in nonoblocks(*nonoargs):
                    # Remove the offset from sub block positions
                    rest = [(offset + bp, bl) for bp, bl in subpos]
                    # Yield this block plus sub blocks positions
                    vec = [(bpos, blength)] + rest
                    yield vec

def pblock(vec, cells):
    'Prettyprints each run of blocks with a different letter A.. for each b
    vector = ['_'] * cells
    for ch, (bp, bl) in enumerate(vec, ord('A')):
        for i in range(bp, bp + bl):
            vector[i] = chr(ch) if vector[i] == '_' else'?'
    return '|' + '|'.join(vector) + '|'


if __name__ == '__main__':
    for blocks, cells in (
            ([2, 1], 5),
            ([], 5),
            ([8], 10),
            ([2, 3, 2, 3], 15),
           # ([4, 3], 10),
           # ([2, 1], 5),
           # ([3, 1], 10),
```

```
            ([2, 3], 5),
            ):
        print('\nConfiguration:\n    %s # %i cells and %r blocks' % (pblock
        print('  Possibilities:')
        for i, vector in enumerate(nonoblocks(blocks, cells)):
            print('   ', pblock(vector, cells))
        print('  A total of %i Possible configurations.' % (i+1))
```

Output:

```
Configuration:
    |_|_|_|_|_| # 5 cells and [2, 1] blocks
  Possibilities:
    |A|A|_|B|_|
    |A|A|_|_|B|
    |_|A|A|_|B|
  A total of 3 Possible configurations.

Configuration:
    |_|_|_|_|_| # 5 cells and [] blocks
  Possibilities:
    |_|_|_|_|_|
```

Python: Nonogram_solver[edit]

First fill cells by deduction, then search through all combinations. It cou

```
from itertools import izip

def gen_row(w, s):
    """Create all patterns of a row or col that match given runs."""
    def gen_seg(o, sp):
        if not o:
            return [[2] * sp]
        return [[2] * x + o[0] + tail
                for x in xrange(1, sp - len(o) + 2)
                for tail in gen_seg(o[1:], sp - x)]

    return [x[1:] for x in gen_seg([[1] * i for i in s], w + 1 - sum(s))]


def deduce(hr, vr):
    """Fix inevitable value of cells, and propagate."""
    def allowable(row):
        return reduce(lambda a, b: [x | y for x, y in izip(a, b)], row)

    def fits(a, b):
```

```
    return all(x & y for x, y in izip(a, b))

def fix_col(n):
    """See if any value in a given column is fixed;
    if so, mark its corresponding row for future fixup."""
    c = [x[n] for x in can_do]
    cols[n] = [x for x in cols[n] if fits(x, c)]
    for i, x in enumerate(allowable(cols[n])):
        if x != can_do[i][n]:
            mod_rows.add(i)
            can_do[i][n] &= x

def fix_row(n):
    """Ditto, for rows."""
    c = can_do[n]
    rows[n] = [x for x in rows[n] if fits(x, c)]
    for i, x in enumerate(allowable(rows[n])):
        if x != can_do[n][i]:
            mod_cols.add(i)
            can_do[n][i] &= x

def show_gram(m):
    # If there's 'x', something is wrong.
    # If there's '?', needs more work.
    for x in m:
        print " ".join("x#.?"[i] for i in x)
    print

w, h = len(vr), len(hr)
rows = [gen_row(w, x) for x in hr]
cols = [gen_row(h, x) for x in vr]
can_do = map(allowable, rows)

# Initially mark all columns for update.
mod_rows, mod_cols = set(), set(xrange(w))

while mod_cols:
    for i in mod_cols:
        fix_col(i)
    mod_cols = set()
    for i in mod_rows:
        fix_row(i)
    mod_rows = set()

if all(can_do[i][j] in (1, 2) for j in xrange(w) for i in xrange(h)):
    print "Solution would be unique" # but could be incorrect!
else:
    print "Solution may not be unique, doing exhaustive search:"

# We actually do exhaustive search anyway. Unique solution takes
# no time in this phase anyway, but just in case there's no
# solution (could happen?).
out = [0] * h

def try_all(n = 0):
    if n >= h:
```

```
            for j in xrange(w):
                if [x[j] for x in out] not in cols[j]:
                    return 0
            show_gram(out)
            return 1
        sol = 0
        for x in rows[n]:
            out[n] = x
            sol += try_all(n + 1)
        return sol

    n = try_all()
    if not n:
        print "No solution."
    elif n == 1:
        print "Unique solution."
    else:
        print n, "solutions."
    print


def solve(p, show_runs=True):
    s = [[[ord(c) - ord('A') + 1 for c in w] for w in l.split()]
         for l in p.splitlines()]
    if show_runs:
        print "Horizontal runs:", s[0]
        print "Vertical runs:", s[1]
    deduce(s[0], s[1])


def main():
    # Read problems from file.
    fn = "nonogram_problems.txt"
    for p in (x for x in open(fn).read().split("\n\n") if x):
        solve(p)

    print "Extra example not solvable by deduction alone:"
    solve("B B A A\nB B A A")

    print "Extra example where there is no solution:"
    solve("B A A\nA A A")

main()
```

Output:

```
Horizontal runs: [[3], [2, 1], [3, 2], [2, 2], [6], [1, 5], [6], [1], [2]]
Vertical runs: [[1, 2], [3, 1], [1, 5], [7, 1], [5], [3], [4], [3]]
Solution would be unique
. # # # . . . .
# # . # . . . .
```

```
. # # # . . # #
. . # # . . # #
. . # # # # # #
# . # # # # # .
# # # # # # . .
. . . . # . . .
. . . # # . . .

Unique solution

(... etc. ...)
```

Racket[edit]

Python: Noughts_and_crosses[edit]

The computer enforces the rules but plays a random game.

```
'''
    Tic-tac-toe game player.
    Input the index of where you wish to place your mark at your turn.
'''

import random

board = list('123456789')
wins = ((0,1,2), (3,4,5), (6,7,8),
        (0,3,6), (1,4,7), (2,5,8),
        (0,4,8), (2,4,6))

def printboard():
    print('\n'.join(' '.join(board[x:x+3]) for x in(0,3,6)))

def score():
    for w in wins:
        b = board[w[0]]
        if b in 'XO' and all (board[i] == b for i in w):
            return b, [i+1 for i in w]
    return None, None

def finished():
    return all (b in 'XO' for b in board)

def space():
    return [ b for b in board if b not in 'XO']
```

```
def my_turn(xo):
    options = space()
    choice = random.choice(options)
    board[int(choice)-1] = xo
    return choice

def your_turn(xo):
    options = space()
    while True:
        choice = input(" Put your %s in any of these positions: %s "
                       % (xo, ''.join(options))).strip()
        if choice in options:
            break
        print( "Whoops I don't understand the input" )
    board[int(choice)-1] = xo
    return choice

def me(xo='X'):
    printboard()
    print('I go at', my_turn(xo))
    return score()
    assert not s[0], "\n%s wins across %s" % s

def you(xo='O'):
    printboard()
    # Call my_turn(xo) below for it to play itself
    print('You went at', your_turn(xo))
    return score()
    assert not s[0], "\n%s wins across %s" % s


print(__doc__)
while not finished():
    s = me('X')
    if s[0]:
        printboard()
        print("\n%s wins across %s" % s)
        break
    if not finished():
        s = you('O')
        if s[0]:
            printboard()
            print("\n%s wins across %s" % s)
            break
else:
    print('\nA draw')
```

Sample Game

```
    Tic-tac-toe game player.
    Input the index of where you wish to place your mark at your turn.
```

```
1 2 3
4 5 6
7 8 9
I go at 9
1 2 3
4 5 6
7 8 X
 Put your O in any of these positions: 12345678 1
You went at 1
O 2 3
4 5 6
7 8 X
I go at 3
O 2 X
```

Python: Nth_root[edit]

```
from decimal import Decimal, getcontext

def nthroot (n, A, precision):
    getcontext().prec = precision

    n = Decimal(n)
    x_0 = A / n #step 1: make a while guess.
    x_1 = 1     #need it to exist before step 2
    while True:
        #step 2:
        x_0, x_1 = x_1, (1 / n)*((n - 1)*x_0 + (A / (x_0 ** (n - 1))))
        if x_0 == x_1:
            return x_1



print nthroot(5, 34, 10)
print nthroot(10,42, 20)
print nthroot(2, 5, 400)
```

R[edit]

Python: Nth_root_algorithm[edit]

```
from decimal import Decimal, getcontext

def nthroot (n, A, precision):
    getcontext().prec = precision
```

```
    n = Decimal(n)
    x_0 = A / n #step 1: make a while guess.
    x_1 = 1     #need it to exist before step 2
    while True:
        #step 2:
        x_0, x_1 = x_1, (1 / n)*((n - 1)*x_0 + (A / (x_0 ** (n - 1))))
        if x_0 == x_1:
            return x_1



print nthroot(5, 34, 10)
print nthroot(10,42, 20)
print nthroot(2, 5, 400)
```

Python: Null[edit]

```
x = None
if x is None:
  print "x is None"
else:
  print "x is not None"
```

Output:

Python: Null_object[edit]

```
x = None
if x is None:
  print "x is None"
else:
  print "x is not None"
```

Output:

```
x is None
```

Python: Number_base_conversion[edit]

Converting from string to number is easy:

```
i = int('1a',16)  # returns the integer 26
```

Converting from number to string is harder:

```
digits = "0123456789abcdefghijklmnopqrstuvwxyz"
def baseN(num,b):
   return (((num == 0) and  "0" )
           or ( baseN(num // b, b).lstrip("0")
                + digits[num % b]))

# alternatively:
def baseN(num,b):
  if num == 0: return "0"
  result = ""
  while num != 0:
    num, d = divmod(num, b)
    result += digits[d]
  return result[::-1] # reverse

k = 26
s = baseN(k,16) # returns the string 1a
```

Python: Number_reversal_game[edit]

```
'''
number reversal game
    Given a jumbled list of the numbers 1 to 9
    Show the list.
    Ask the player how many digits from the left to reverse.
    Reverse those digits then ask again.
    until all the digits end up in ascending order.

'''

import random

print(__doc__)
data, trials = list('123456789'), 0
while data == sorted(data):
    random.shuffle(data)
while data != sorted(data):
    trials += 1
    flip = int(input('#%2i: LIST: %r Flip how many?: ' % (trials, ' '.join(d
    data[:flip] = reversed(data[:flip])

print('\nYou took %2i attempts to put the digits in order!' % trials)
```

Sample output:

```
number reversal game
    Given a jumbled list of the numbers 1 to 9
```

Python: Numeric_error_propagation[edit]

```
from collections import namedtuple
import math

class I(namedtuple('Imprecise', 'value, delta')):
    'Imprecise type: I(value=0.0, delta=0.0)'

    __slots__ = ()

    def __new__(_cls, value=0.0, delta=0.0):
        'Defaults to 0.0 ± delta'
        return super().__new__(_cls, float(value), abs(float(delta)))

    def reciprocal(self):
        return I(1. / self.value, self.delta / (self.value**2))

    def __str__(self):
        'Shorter form of Imprecise as string'
        return 'I(%g, %g)' % self

    def __neg__(self):
        return I(-self.value, self.delta)

    def __add__(self, other):
        if type(other) == I:
            return I( self.value + other.value, (self.delta**2 + other.delt
        try:
            c = float(other)
        except:
            return NotImplemented
        return I(self.value + c, self.delta)

    def __sub__(self, other):
        return self + (-other)

    def __radd__(self, other):
        return I.__add__(self, other)

    def __mul__(self, other):
        if type(other) == I:
            #if id(self) == id(other):
            #    return self ** 2
            a1,b1 = self
            a2,b2 = other
            f = a1 * a2
```

```
            return I( f, f * ( (b1 / a1)**2 + (b2 / a2)**2 )**0.5 )
        try:
            c = float(other)
        except:
            return NotImplemented
        return I(self.value * c, self.delta * c)

    def __pow__(self, other):
        if type(other) == I:
            return NotImplemented
        try:
            c = float(other)
        except:
            return NotImplemented
        f = self.value ** c
        return I(f, f * c * (self.delta / self.value))

    def __rmul__(self, other):
        return I.__mul__(self, other)

    def __truediv__(self, other):
        if type(other) == I:
            return self.__mul__(other.reciprocal())
        try:
            c = float(other)
        except:
            return NotImplemented
        return I(self.value / c, self.delta / c)

    def __rtruediv__(self, other):
        return other * self.reciprocal()

    __div__, __rdiv__ = __truediv__, __rtruediv__

Imprecise = I

def distance(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return ((x1 - x2)**2 + (y1 - y2)**2)**0.5

x1 = I(100, 1.1)
x2 = I(200, 2.2)
y1 = I( 50, 1.2)
y2 = I(100, 2.3)

p1, p2 = (x1, y1), (x2, y2)
print("Distance between points\n  p1: %s\n  and p2: %s\n  = %r" % (
      p1, p2, distance(p1, p2)))
```

Sample output

```
Distance between points
  p1: (I(value=100.0, delta=1.1), I(value=50.0, delta=1.2))
  and p2: (I(value=200.0, delta=2.2), I(value=100.0, delta=2.3))
  = I(value=111.80339887498948, delta=2.4871670631463423)
```

Racket[edit]

Python: Numerical_Integration[edit]

Answers are first given using floating point arithmatic, then using fractio

```
from fractions import Fraction

def left_rect(f,x,h):
  return f(x)

def mid_rect(f,x,h):
  return f(x + h/2)

def right_rect(f,x,h):
  return f(x+h)

def trapezium(f,x,h):
  return (f(x) + f(x+h))/2.0

def simpson(f,x,h):
  return (f(x) + 4*f(x + h/2) + f(x+h))/6.0

def cube(x):
  return x*x*x

def reciprocal(x):
  return 1/x

def identity(x):
  return x

def integrate( f, a, b, steps, meth):
   h = (b-a)/steps
   ival = h * sum(meth(f, a+i*h, h) for i in range(steps))
   return ival

# Tests
for a, b, steps, func in ((0., 1., 100, cube), (1., 100., 1000, reciprocal)
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps) = %r' %
              (func.__name__, rule.__name__, a, b, steps,
```

```
             integrate( func, a, b, steps, rule)))
    a, b = Fraction.from_float(a), Fraction.from_float(b)
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps and fracti
              (func.__name__, rule.__name__, a, b, steps,
               float(integrate( func, a, b, steps, rule))))

# Extra tests (compute intensive)
for a, b, steps, func in ((0., 5000., 5000000, identity),
                          (0., 6000., 6000000, identity)):
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps) = %r' %
              (func.__name__, rule.__name__, a, b, steps,
               integrate( func, a, b, steps, rule)))
    a, b = Fraction.from_float(a), Fraction.from_float(b)
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps and fracti
              (func.__name__, rule.__name__, a, b, steps,
               float(integrate( func, a, b, steps, rule))))
```

Tests

```
for a, b, steps, func in ((0., 1., 100, cube), (1., 100., 1000, reciprocal)
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps) = %r' %
              (func.__name__, rule.__name__, a, b, steps,
               integrate( func, a, b, steps, rule)))
    a, b = Fraction.from_float(a), Fraction.from_float(b)
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps and fracti
              (func.__name__, rule.__name__, a, b, steps,
               float(integrate( func, a, b, steps, rule))))

# Extra tests (compute intensive)
for a, b, steps, func in ((1., 5000., 5000000, identity),
                          (1., 6000., 6000000, identity)):
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps) = %r' %
              (func.__name__, rule.__name__, a, b, steps,
               integrate( func, a, b, steps, rule)))
    a, b = Fraction.from_float(a), Fraction.from_float(b)
    for rule in (left_rect, mid_rect, right_rect, trapezium, simpson):
        print('%s integrated using %s\n  from %r to %r (%i steps and fracti
              (func.__name__, rule.__name__, a, b, steps,
               float(integrate( func, a, b, steps, rule))))
```

Sample test Output

Python: Object_serialization[edit]

```
# Object Serialization in Python
# serialization in python is accomplished via the Pickle module.
# Alternatively, one can use the cPickle module if speed is the key,
# everything else in this example remains the same.

import pickle

class Entity:
        def __init__(self):
                self.name = "Entity"
        def printName(self):
                print self.name

class Person(Entity): #OldMan inherits from Entity
        def __init__(self): #override constructor
                self.name = "Cletus"

instance1 = Person()
instance1.printName()

instance2 = Entity()
instance2.printName()

target = file("objects.dat", "w") # open file

#  Serialize
pickle.dump((instance1, instance2), target) # serialize `instance1` and `in
target.close() # flush file stream
print "Serialized..."

# Unserialize
target = file("objects.dat") # load again
i1, i2 = pickle.load(target)
print "Unserialized..."

i1.printName()
i2.printName()
```

Python: Old_Russian_measure_of_length[edit]

Run as:

```
     commandname <value> <unit>
```

```
from sys import argv
```

```
unit2mult = {"arshin": 0.7112, "centimeter": 0.01,     "diuym":   0.0254,
             "fut":    0.3048, "kilometer":  1000.0,   "liniya":  0.00254,
             "meter":  1.0,    "milia":      7467.6,   "piad":    0.1778,
             "sazhen": 2.1336, "tochka":     0.000254, "vershok": 0.04445,
             "versta": 1066.8}

if __name__ == '__main__':
    assert len(argv) == 3, 'ERROR. Need two arguments - number then units'
    try:
        value = float(argv[1])
    except:
        print('ERROR. First argument must be a (float) number')
        raise
    unit = argv[2]
    assert unit in unit2mult, ( 'ERROR. Only know the following units: '
                                + ' '.join(unit2mult.keys()) )

    print("%g %s to:" % (value, unit))
    for unt, mlt in sorted(unit2mult.items()):
        print('  %10s: %g' % (unt, value * unit2mult[unit] / mlt))
```

Output:

```
1 meter to:
      arshin: 1.40607
  centimeter: 100
       diuym: 39.3701
         fut: 3.28084
   kilometer: 0.001
      liniya: 393.701
       meter: 1
       milia: 0.000133912
        piad: 5.6243
      sazhen: 0.468691
      tochka: 3937.01
     vershok: 22.4972
      versta: 0.000937383
```

Output:

```
1 milia to:
      arshin: 10500
  centimeter: 746760
       diuym: 294000
         fut: 24500
   kilometer: 7.4676
      liniya: 2.94e+06
```

```
       meter: 7467.6
       milia: 1
        piad: 42000
      sazhen: 3500
      tochka: 2.94e+07
     vershok: 168000
      versta: 7
```

Output:

When given a wrong number

```
ERROR. First argument must be a (float) number
Traceback (most recent call last):
  File "C:\Users\Paddy\Google Drive\Code\old_russian_lengths.py", line 18,
    value = float(argv[1])
ValueError: could not convert string to float: '1xx'
```

Output:

When given a wrong unit

```
Traceback (most recent call last):
  File "C:\Users\Paddy\Google Drive\Code\old_russian_lengths.py", line 24,
    + ' '.join(unit2mult.keys()) )
AssertionError: ERROR. Only know the following units: kilometer tochka vers
```

Python: Old_lady_swallowed_a_fly[edit]

```
import zlib, base64

b64 = b'''
eNrtVE1rwzAMvedXaKdeRn7ENrb21rHCzmrs1m49K9gOJv9+cko/HBcGg0LHcpOfnq2np0QL
2FuKgBbICDAoeoiKwEc0hqIUgLAxfV0tQJCdhQM7qh68kheswKeBt5ROYetTemYMCC3rii//
WMS3WkhXVyuFAaLT261JuBWwu4iDbvYp1tYzHVS68VEIObwFgaDB0KizuFs38aSdqKv3TgcJ
uPYdn2B1opwIpeKE53qPftxRd88Y6uoVbdPzWxznrQ3ZUi3DudQ/bcELbevqM32iCIrj3IIh
W6plOJf6L6xaajZjzqW/qAsKIvITBGs9Nm3glboZzkVP5l6Y+0bHLnedD0CttIyrpEU5Kv7N
Mz3XkPBc/TSN3yxGiqMiipHRekycK0ZwMhM8jerGC9zuZaoTho3kMKSfJjLaF8v8wLzmXMqM
zJvGew/jnZPzclA08yAkikegDTTUMfzwDXBcwoE='''
```

```
print(zlib.decompress(base64.b64decode(b64)).decode("utf-8", "strict"))
```

Racket[edit]

Python: One_of_n_lines_in_a_file[edit]

To be more in line with the spirit of the problem, one_of_n will take the "

```
from random import randrange
try:
    range = xrange
except: pass

def one_of_n(lines): # lines is any iterable
    choice = None
    for i, line in enumerate(lines):
        if randrange(i+1) == 0:
            choice = line
    return choice

def one_of_n_test(n=10, trials=1000000):
    bins = [0] * n
    if n:
        for i in range(trials):
            bins[one_of_n(range(n))] += 1
    return bins

print(one_of_n_test())
```

Sample output

```
[99833, 100303, 99902, 100132, 99608, 100117, 99531, 100017, 99795, 100762]
```

Python: OpenWebNet_Password[edit]

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def ownCalcPass (password, nonce) :
    m_1 = 0xFFFFFFFFL
    m_8 = 0xFFFFFFF8L
    m_16 = 0xFFFFFFF0L
    m_128 = 0xFFFFFF80L
    m_16777216 = 0XFF000000L
    flag = True
    num1 = 0L
    num2 = 0L
    password = long(password)

    for c in nonce :
        num1 = num1 & m_1
        num2 = num2 & m_1
        if c == '1':
            length = not flag
            if not length :
                num2 = password
            num1 = num2 & m_128
            num1 = num1 >> 7
            num2 = num2 << 25
            num1 = num1 + num2
            flag = False
        elif c == '2':
            length = not flag
            if not length :
                num2 = password
            num1 = num2 & m_16
            num1 = num1 >> 4
            num2 = num2 << 28
            num1 = num1 + num2
            flag = False
        elif c == '3':
            length = not flag
            if not length :
                num2 = password
            num1 = num2 & m_8
            num1 = num1 >> 3
            num2 = num2 << 29
            num1 = num1 + num2
            flag = False
        elif c == '4':
            length = not flag

            if not length:
                num2 = password
            num1 = num2 << 1
            num2 = num2 >> 31
            num1 = num1 + num2
            flag = False
        elif c == '5':
```

```
            length = not flag
            if not length:
                num2 = password
            num1 = num2 << 5
            num2 = num2 >> 27
            num1 = num1 + num2
            flag = False
        elif c == '6':
            length = not flag
            if not length:
                num2 = password
            num1 = num2 << 12
            num2 = num2 >> 20
            num1 = num1 + num2
            flag = False
        elif c == '7':
            length = not flag
            if not length:
                num2 = password
            num1 = num2 & 0xFF00L
            num1 = num1 + (( num2 & 0xFFL ) << 24 )
            num1 = num1 + (( num2 & 0xFF0000L ) >> 16 )
            num2 = ( num2 & m_16777216 ) >> 8
            num1 = num1 + num2
            flag = False
        elif c == '8':
            length = not flag
            if not length:
                num2 = password
            num1 = num2 & 0xFFFFL
            num1 = num1 << 16
            num1 = num1 + ( num2 >> 24 )
            num2 = num2 & 0xFF0000L
            num2 = num2 >> 8
            num1 = num1 + num2
            flag = False
        elif c == '9':
            length = not flag
            if not length:
                num2 = password
            num1 = ~num2
            flag = False
        else :
            num1 = num2
        num2 = num1
    return num1 & m_1

def ownTestCalcPass (passwd, nonce, expected) :
        res = ownCalcPass(passwd, nonce)
        m = passwd+' '+nonce+' '+str(res)+' '+str(expected)
        if res == long(expected) :
                print 'PASS '+m
        else :
                print 'FAIL '+m

if __name__ == '__main__':
```

```
import sys

ownTestCalcPass('12345','603356072','25280520')
ownTestCalcPass('12345','410501656','119537670')
```

Python: Operator_precedence[edit]

See this table and the whole page for details on Python version 3.x
An excerpt of which is this table:

Precedence	Operator	Description
lowest	**lambda**	Lambda expression
	if – else	Conditional expression
	or	Boolean OR
	and	Boolean AND
	not x	Boolean NOT
	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests,
	|	Bitwise OR
	^	Bitwise XOR
	&	Bitwise AND
	<<, >>	Shifts
	+, -	Addition and subtraction
	***, /, //, %**	Multiplication, division, remainder [1]
	+x, -x, ~x	Positive, negative, bitwise NOT
	**	Exponentiation [2]
	x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
highest	**(expressions...), [expressions...], {key:datum...}, {expressions...}**	Binding or tuple display, list display, dictionary display, set display

Footnotes

1. The % operator is also used for string formatting; the same precedenc
2. The power operator ** binds less tightly than an arithmetic or bitwis

Python: Optional_parameters[edit]

Works with: Python version 2.x
 only (the "cmp" argument to sorted() is no longer accepted in Python 3)

Using a pretty-printer for the table

```
>>> def printtable(data):
    for row in data:
        print ' '.join('%-5s' % ('"%s"' % cell) for cell in row)
```

```
>>> import operator
>>> def sorttable(table, ordering=None, column=0, reverse=False):
    return sorted(table, cmp=ordering, key=operator.itemgetter(column), reve

>>> data = [["a", "b", "c"], ["", "q", "z"], ["zap", "zip", "Zot"]]
>>> printtable(data)
"a"   "b"   "c"
""    "q"   "z"
"zap" "zip" "Zot"
>>> printtable( sorttable(data) )
""    "q"   "z"
"a"   "b"   "c"
"zap" "zip" "Zot"
>>> printtable( sorttable(data, column=2) )
"zap" "zip" "Zot"
"a"   "b"   "c"
""    "q"   "z"
>>> printtable( sorttable(data, column=1) )
"a"   "b"   "c"
""    "q"   "z"
"zap" "zip" "Zot"
>>> printtable( sorttable(data, column=1, reverse=True) )
"zap" "zip" "Zot"
""    "q"   "z"
"a"   "b"   "c"
>>> printtable( sorttable(data, ordering=lambda a,b: cmp(len(b),len(a))) )
"zap" "zip" "Zot"
"a"   "b"   "c"
""    "q"   "z"
>>>
```

See the Python entry in Named Arguments for a more comprehensive descriptio

Note that expression for a default argument of an optional parameter is eva

```
>>> def foo(x, lst=[]):
...   lst.append(x)
...   print lst
...
>>> foo(1)
[1]
>>> foo(2)
[1, 2]
>>> foo(3)
[1, 2, 3]
```

R[edit]

Python: Order_disjoint_list_items[edit]

```
from __future__ import print_function

def order_disjoint_list_items(data, items):
    #Modifies data list in-place
    itemindices = []
    for item in set(items):
        itemcount = items.count(item)
        #assert data.count(item) >= itemcount, 'More of %r than in data' %
        lastindex = [-1]
        for i in range(itemcount):
            lastindex.append(data.index(item, lastindex[-1] + 1))
        itemindices += lastindex[1:]
    itemindices.sort()
    for index, item in zip(itemindices, items):
        data[index] = item

if __name__ == '__main__':
    tostring = ' '.join
    for data, items in [ (str.split('the cat sat on the mat'), str.split('m
                         (str.split('the cat sat on the mat'), str.split('c
                         (list('ABCABCABC'), list('CACA')),
                         (list('ABCABDABE'), list('EADA')),
                         (list('AB'), list('B')),
                         (list('AB'), list('BA')),
                         (list('ABBA'), list('BA')),
                         (list(''), list('')),
                         (list('A'), list('A')),
                         (list('AB'), list('')),
                         (list('ABBA'), list('AB')),
                         (list('ABAB'), list('AB')),
                         (list('ABAB'), list('BABA')),
                         (list('ABCCBA'), list('ACAC')),
                         (list('ABCCBA'), list('CACA')),
```

```
                        ]:
        print('Data M: %-24r Order N: %-9r' % (tostring(data), tostring(ite
        order_disjoint_list_items(data, items)
        print("-> M' %r" % tostring(data))
```

Output:

```
Data M: 'the cat sat on the mat' Order N: 'mat cat' -> M' 'the mat sat on t
Data M: 'the cat sat on the mat' Order N: 'cat mat' -> M' 'the cat sat on t
Data M: 'A B C A B C A B C'      Order N: 'C A C A' -> M' 'C B A C B A A B 
Data M: 'A B C A B D A B E'      Order N: 'E A D A' -> M' 'E B C A B D A B 
Data M: 'A B'                    Order N: 'B'       -> M' 'A B'
Data M: 'A B'                    Order N: 'B A'     -> M' 'B A'
Data M: 'A B B A'                Order N: 'B A'     -> M' 'B A B A'
Data M: ''                       Order N: ''        -> M' ''
Data M: 'A'                      Order N: 'A'       -> M' 'A'
Data M: 'A B'                    Order N: ''        -> M' 'A B'
Data M: 'A B B A'                Order N: 'A B'     -> M' 'A B B A'
Data M: 'A B A B'                Order N: 'A B'     -> M' 'A B A B'
Data M: 'A B A B'                Order N: 'B A B A' -> M' 'B A B A'
Data M: 'A B C C B A'            Order N: 'A C A C' -> M' 'A B C A B C'
Data M: 'A B C C B A'            Order N: 'C A C A' -> M' 'C B A C B A'
```

Racket[edit]

Python: Order_two_numerical_lists[edit]

The built-in comparison operators already do this:

```
>>> [1,2,1,3,2] < [1,2,0,4,4,0,0,0]
False
```

Racket[edit]

Python: Ordered_Partitions[edit]

```
from itertools import combinations

def partitions(*args):
    def p(s, *args):
        if not args: return [[]]
        res = []
        for c in combinations(s, args[0]):
            s0 = [x for x in s if x not in c]
            for r in p(s0, *args[1:]):
                res.append([c] + r)
        return res
    s = range(sum(args))
    return p(s, *args)

print partitions(2, 0, 2)
```

An equivalent but terser solution.

Python: Ordered_words[edit]

```
import urllib.request

url = 'http://www.puzzlers.org/pub/wordlists/unixdict.txt'
words = urllib.request.urlopen(url).read().decode("utf-8").split()
ordered = [word for word in words if word==''.join(sorted(word))]
maxlen = len(max(ordered, key=len))
maxorderedwords = [word for word in ordered if len(word) == maxlen]
print(' '.join(maxorderedwords))
```

Alternate Solution

```
import urllib.request

mx, url = 0, 'http://www.puzzlers.org/pub/wordlists/unixdict.txt'

for word in urllib.request.urlopen(url).read().decode("utf-8").split():
    lenword = len(word)
    if lenword >= mx and word==''.join(sorted(word)):
        if lenword > mx:
            words, mx = [], lenword
        words.append(word)
print(' '.join(words))
```

Sample Output

abbott accent accept access accost almost bellow billow biopsy chilly choos

Short local version:

Python: Palindrome[edit]

Non-recursive

This one uses the *reversing the string* technique
(to reverse a string Python can use the odd
but right syntax string[::-1])

```
def is_palindrome(s):
  return s == s[::-1]
```

Recursive

```
def is_palindrome_r(s):
  if len(s) <= 1:
    return True
  elif s[0] != s[-1]:
    return False
  else:
    return is_palindrome_r(s[1:-1])
```

Python has short-circuit evaluation of Boolean operations
so a shorter and still easy to understand recursive function is

```
def is_palindrome_r2(s):
  return not s or s[0] == s[-1] and is_palindrome_r2(s[1:-1])
```

Testing

```
def test(f, good, bad):
  assert all(f(x) for x in good)
  assert not any(f(x) for x in bad)
```

```
  print '%s passed all %d tests' % (f.__name__, len(good)+len(bad))

pals = ('', 'a', 'aa', 'aba', 'abba')
notpals = ('aA', 'abA', 'abxBa', 'abxxBa')
for ispal in is_palindrome, is_palindrome_r, is_palindrome_r2:
  test(ispal, pals, notpals)
```

Python: Palindrome_detection[edit]

Non-recursive

This one uses the *reversing the string* technique
(to reverse a string Python can use the odd
but right syntax string[::-1])

```
def is_palindrome(s):
  return s == s[::-1]
```

Recursive

```
def is_palindrome_r(s):
  if len(s) <= 1:
    return True
  elif s[0] != s[-1]:
    return False
  else:
    return is_palindrome_r(s[1:-1])
```

Python has short-circuit evaluation of Boolean operations
so a shorter and still easy to understand recursive function is

```
def is_palindrome_r2(s):
  return not s or s[0] == s[-1] and is_palindrome_r2(s[1:-1])
```

Testing

```
def test(f, good, bad):
  assert all(f(x) for x in good)
  assert not any(f(x) for x in bad)
```

```
  print '%s passed all %d tests' % (f.__name__, len(good)+len(bad))

pals = ('', 'a', 'aa', 'aba', 'abba')
notpals = ('aA', 'abA', 'abxBa', 'abxxBa')
for ispal in is_palindrome, is_palindrome_r, is_palindrome_r2:
  test(ispal, pals, notpals)
```

Python: Pangram_checker[edit]

Using set arithmetic:

```
import string, sys
if sys.version_info[0] < 3:
    input = raw_input

def ispangram(sentence, alphabet=string.ascii_lowercase):
    alphaset = set(alphabet)
    return alphaset <= set(sentence.lower())

print ( ispangram(input('Sentence: ')) )
```

Output:

```
Sentence: The quick brown fox jumps over the lazy dog
True
```

R[edit]

Python: Paraffins[edit]

This version only counts different paraffins. The multi-precision integers

Translation of: C

```
try:
    import psyco
```

```
    psyco.full()
except ImportError:
    pass

MAX_N = 300
BRANCH = 4

ra = [0] * MAX_N
unrooted = [0] * MAX_N

def tree(br, n, l, sum = 1, cnt = 1):
    global ra, unrooted, MAX_N, BRANCH
    for b in xrange(br + 1, BRANCH + 1):
        sum += n
        if sum >= MAX_N:
            return

        # prevent unneeded long math
        if l * 2 >= sum and b >= BRANCH:
            return

        if b == br + 1:
            c = ra[n] * cnt
        else:
            c = c * (ra[n] + (b - br - 1)) / (b - br)

        if l * 2 < sum:
            unrooted[sum] += c

        if b < BRANCH:
            ra[sum] += c;
            for m in range(1, n):
                tree(b, m, l, sum, c)

def bicenter(s):
    global ra, unrooted
    if not (s & 1):
        aux = ra[s / 2]
        unrooted[s] += aux * (aux + 1) / 2


def main():
    global ra, unrooted, MAX_N
    ra[0] = ra[1] = unrooted[0] = unrooted[1] = 1

    for n in xrange(1, MAX_N):
        tree(0, n, n)
        bicenter(n)
        print "%d: %d" % (n, unrooted[n])

main()
```

Output (newlines added):

1: 1

Python: Parametrized_SQL_statement[edit]

Translation of: Ruby

```
import sqlite3

db = sqlite3.connect(':memory:')

# setup
db.execute('create temp table players (name, score, active, jerseyNum)')
db.execute('insert into players values ("name",0,"false",99)')
db.execute('insert into players values ("name",0,"false",100)')

# demonstrate parameterized SQL

# example 1 -- simple placeholders
db.execute('update players set name=?, score=?, active=? where jerseyNum=?'

# example 2 -- named placeholders
db.execute('update players set name=:name, score=:score, active=:active whe
    {'num': 100,
     'name': 'John Doe',
     'active': False,
     'score': -1}
)

# and show the results
for row in db.execute('select * from players'):
   print(row)
```

outputs

```
(u'Smith, Steve', 42, 1, 99)
(u'John Doe', -1, 0, 100)
```

Python: Parse_an_IP_Address[edit]

Library: pyparse

The following uses pyparse to parse the IP address. It's an attempt at usin

```
import string
from pyparsing import * # import antigravity

tests="""#
127.0.0.1                       # The "localhost" IPv4 address
127.0.0.1:80                    # The "localhost" IPv4 address, with a spec
::1                             # The "localhost" IPv6 address
[::1]:80                        # The "localhost" IPv6 address, with a spec
2605:2700:0:3::4713:93e3        # Rosetta Code's primary server's public IP
[2605:2700:0:3::4713:93e3]:80   # Rosetta Code's primary server's public IP
2001:db8:85a3:0:0:8a2e:370:7334 # doc, IPv6 for 555-1234
2001:db8:85a3::8a2e:370:7334    # doc
[2001:db8:85a3:8d3:1319:8a2e:370:7348]:443 # doc +port
192.168.0.1                     # private
::ffff:192.168.0.1              # private transitional
::ffff:71.19.147.227            # Rosetta Code's transitional
[::ffff:71.19.147.227]:80       # Rosetta Code's transitional +port
::                              # unspecified
256.0.0.0                       # invalid, octet > 255 (currently not detec
g::1                            # invalid
0000                                    Bad address
0000:0000                               Bad address
0000:0000:0000:0000:0000:0000:0000:0000 Good address
0000:0000:0000::0000:0000               Good Address
0000::0000::0000:0000                   Bad address
ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff Good address
ffff:ffff:ffff:fffg:ffff:ffff:ffff:ffff Bad address
fff:ffff:ffff:ffff:ffff:ffff:ffff:ffff  Good address
fff:ffff:0:ffff:ffff:ffff:ffff:ffff     Good address
"""

def print_args(args):
  print "print_args:", args

def join(args):
  args[0]="".join(args)
  del args[1:]

def replace(val):
  def lambda_replace(args):
    args[0]=val
    del args[1:]
  return lambda_replace

def atoi(args): args[0]=string.atoi(args[0])
def itohex2(args): args[0]="%02x"%args[0]

def hextoi(args): args[0]=string.atoi(args[0], 16)
def itohex4(args): args[0]="%04x"%args[0]

def assert_in_range(lwb, upb):
  def range_check(args):
    return # turn range checking off
    if args[0] < lwb:
```

```
      raise ValueError,"value %d < %d"%(args[0], lwb)
    if args[0] > upb:
      raise ValueError,"value %d > %d"%(args[0], upb)
  return range_check

dot = Literal(".").suppress()("dot"); colon = Literal(":").suppress()("colo
octet = Word(nums).setParseAction(atoi,assert_in_range(0,255),itohex2)("oct

port = Word(nums).setParseAction(atoi,assert_in_range(0,256*256-1))("port")
ipv4 = (octet + (dot+octet)*3)("addr")
ipv4.setParseAction(join) #,hextoi)

ipv4_port = ipv4+colon.suppress()+port

a2f = "abcdef"
hex = oneOf(" ".join(nums+a2f));

hexet = (hex*(0,4))("hexet")
hexet.setParseAction(join, hextoi, itohex4)

max=8; stop=max+1

xXXXX_etc = [None, hexet]; xXXXX_etc.extend([hexet + (colon+hexet)*n for n
x0000_etc = [ Literal("::").setParseAction(replace("0000"*num_x0000s)) for

ipv6=xXXXX_etc[-1]+x0000_etc[0] | xXXXX_etc[-1]

# Build a table of rules for IPv6, in particular the double colon
for num_prefix in range(max-1, -1, -1):
  for num_x0000s in range(0,stop-num_prefix):
    x0000 = x0000_etc[num_x0000s]
    num_suffix=max-num_prefix-num_x0000s
    if num_prefix:
      if num_suffix: pat = xXXXX_etc[num_prefix]+x0000+xXXXX_etc[num_suffix
      else:          pat = xXXXX_etc[num_prefix]+x0000
    elif num_suffix: pat =                       x0000+xXXXX_etc[num_suffix
    else: pat=x0000
    ipv6 = ipv6 | pat

ipv6.setParseAction(join) # ,hextoi)
ipv6_port = Literal("[").suppress() + ipv6 + Literal("]").suppress()+colon+

ipv6_transitional = (Literal("::ffff:").setParseAction(replace("0"*20+"ffff
ipv6_transitional_port = Literal("[").suppress() + ipv6_transitional + Lite

ip_fmt = (
           (ipv4_port|ipv4)("ipv4") |
           (ipv6_transitional_port|ipv6_transitional|ipv6_port|ipv6)("ipv6"
         ) + LineEnd()

class IPAddr(object):
  def __init__(self, string):
    self.service = dict(zip(("address","port"), ip_fmt.parseString(string)[
  def __getitem__(self, key): return self.service[key]
  def __contains__(self, key): return key in self.service
  def __repr__(self): return `self.service` # "".join(self.service)
```

```
  address=property(lambda self: self.service["address"])
  port=property(lambda self: self.service["port"])
  is_service=property(lambda self: "port" in self.service)
  version=property(lambda self: {False:4, True:6}[len(self.address)>8])

for test in tests.splitlines():
  if not test.startswith("#"):
    ip_str, desc = test.split(None,1)
    print ip_str,"=>",
    try:
      ip=IPAddr(ip_str)
      print ip, "IP Version:",ip.version,"- Address is OK!",
    except (ParseException,ValueError), details: print "Bad! IP address syn
    print "- Actually:",desc
```

Output:

```
127.0.0.1 => {'address': '7f000001'} IP Version: 4 - Address is OK! - Actua
127.0.0.1:80 => {'port': 80, 'address': '7f000001'} IP Version: 4 - Address
d port (80)
::1 => {'address': '00000000000000000000000000000001'} IP Version: 6 - Addr
[::1]:80 => {'port': 80, 'address': '00000000000000000000000000000001'} IP 
ess, with a specified port (80)
2605:2700:0:3::4713:93e3 => {'address': '260527000000000300000000471393e3'}
y server's public IPv6 address
[2605:2700:0:3::4713:93e3]:80 => {'port': 80, 'address': '2605270000000003
tta Code's primary server's public IPv6 address, +port (80)
2001:db8:85a3:0:0:8a2e:370:7334 => {'address': '20010db885a3000000008a2e037
555-1234
2001:db8:85a3::8a2e:370:7334 => {'address': '20010db885a3000000008a2e03707
[2001:db8:85a3:8d3:1319:8a2e:370:7348]:443 => {'port': 443, 'address': '200
tually: # doc +port
192.168.0.1 => {'address': 'c0a80001'} IP Version: 4 - Address is OK! - Act
::ffff:192.168.0.1 => {'address': '00000000000000000000ffffc0a80001'} IP Ve
::ffff:71.19.147.227 => {'address': '00000000000000000000ffff471393e3'} IP 
al
[::ffff:71.19.147.227]:80 => {'port': 80, 'address': '00000000000000000000f
Code's transitional +port
:: => {'address': '00000000000000000000000000000000'} IP Version: 6 - Addre
256.0.0.0 => {'address': '100000000'} IP Version: 6 - Address is OK! - Actu
g::1 => Bad! IP address syntax error detected:  (at char 4), (line:1, col:5
0000 => Bad! IP address syntax error detected: Expected "." (at char 4), (l
0000:0000 => Bad! IP address syntax error detected: Expected ":" (at char 9
0000:0000:0000:0000:0000:0000:0000:0000 => {'address': '000000000000000000
ress
0000:0000:0000::0000:0000 => {'address': '00000000000000000000000000000000'
0000::0000::0000:0000 => Bad! IP address syntax error detected: Expected en
ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff => {'address': 'ffffffffffffffffff
ress
ffff:ffff:ffff:fffg:ffff:ffff:ffff:ffff => Bad! IP address syntax error det
address
fff:ffff:ffff:ffff:ffff:ffff:ffff:ffff => {'address': '0fffffffffffffffffff
```

```
ess
fff:ffff:0:ffff:ffff:ffff:ffff:ffff => {'address': '0ffffff0000ffffffffff
```

Python: Parse_command-line_arguments[edit]

Version 2.3+

```
from optparse import OptionParser
[...]
parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="write report to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")

(options, args) = parser.parse_args()

example:

<yourscript> --file=outfile -q
```

PicoLisp[edit]

Python: Parsing_command-line_arguments[edit]

Version 2.3+

```
from optparse import OptionParser
[...]
parser = OptionParser()
parser.add_option("-f", "--file", dest="filename",
                  help="write report to FILE", metavar="FILE")
parser.add_option("-q", "--quiet",
                  action="store_false", dest="verbose", default=True,
                  help="don't print status messages to stdout")
```

```
(options, args) = parser.parse_args()

example:

<yourscript> --file=outfile -q
```

Python: Partial_function_application[edit]

```
from functools import partial

def fs(f, s): return [f(value) for value in s]

def f1(value): return value * 2

def f2(value): return value ** 2

fsf1 = partial(fs, f1)
fsf2 = partial(fs, f2)

s = [0, 1, 2, 3]
assert fs(f1, s) == fsf1(s) # ==  [0, 2, 4, 6]
assert fs(f2, s) == fsf2(s) # ==  [0, 1, 4, 9]

s = [2, 4, 6, 8]
assert fs(f1, s) == fsf1(s) # ==  [4, 8, 12, 16]
assert fs(f2, s) == fsf2(s) # ==  [4, 16, 36, 64]
```

The program runs without triggering the assertions.

Python: Pascal's_Triangle[edit]

```
def pascal(n):
   """Prints out n rows of Pascal's triangle.
   It returns False for failure and True for success."""
   row = [1]
   k = [0]
   for x in range(max(n,0)):
      print row
      row=[l+r for l,r in zip(row+k,k+row)]
   return n>=1
```

Or, by creating a scan function:

```
def scan(op, seq, it):
  a = []
  result = it
  a.append(it)
  for x in seq:
    result = op(result, x)
    a.append(result)
  return a

def pascal(n):
    def nextrow(row, x):
        return [l+r for l,r in zip(row+[0,],[0,]+row)]

    return scan(nextrow, range(n-1), [1,])

for row in pascal(4):
    print(row)
```

Python: Pendulum_Animation[edit]

Library: pygame
[edit]

Translation of: C

```
import pygame, sys
from pygame.locals import *
from math import sin, cos, radians

pygame.init()

WINDOWSIZE = 250
TIMETICK = 100
BOBSIZE = 15

window = pygame.display.set_mode((WINDOWSIZE, WINDOWSIZE))
pygame.display.set_caption("Pendulum")

screen = pygame.display.get_surface()
screen.fill((255,255,255))

PIVOT = (WINDOWSIZE/2, WINDOWSIZE/10)
SWINGLENGTH = PIVOT[1]*4

class BobMass(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
```

```
        self.theta = 45
        self.dtheta = 0
        self.rect = pygame.Rect(PIVOT[0]-SWINGLENGTH*cos(radians(self.theta
                                PIVOT[1]+SWINGLENGTH*sin(radians(self.theta
                                1,1)
        self.draw()

    def recomputeAngle(self):
        scaling = 3000.0/(SWINGLENGTH**2)

        firstDDtheta = -sin(radians(self.theta))*scaling
        midDtheta = self.dtheta + firstDDtheta
        midtheta = self.theta + (self.dtheta + midDtheta)/2.0

        midDDtheta = -sin(radians(midtheta))*scaling
        midDtheta = self.dtheta + (firstDDtheta + midDDtheta)/2
        midtheta = self.theta + (self.dtheta + midDtheta)/2

        midDDtheta = -sin(radians(midtheta)) * scaling
        lastDtheta = midDtheta + midDDtheta
        lasttheta = midtheta + (midDtheta + lastDtheta)/2.0

        lastDDtheta = -sin(radians(lasttheta)) * scaling
        lastDtheta = midDtheta + (midDDtheta + lastDDtheta)/2.0
        lasttheta = midtheta + (midDtheta + lastDtheta)/2.0

        self.dtheta = lastDtheta
        self.theta = lasttheta
        self.rect = pygame.Rect(PIVOT[0]-
                                SWINGLENGTH*sin(radians(self.theta)),
                                PIVOT[1]+
                                SWINGLENGTH*cos(radians(self.theta)),1,1)


    def draw(self):
        pygame.draw.circle(screen, (0,0,0), PIVOT, 5, 0)
        pygame.draw.circle(screen, (0,0,0), self.rect.center, BOBSIZE, 0)
        pygame.draw.aaline(screen, (0,0,0), PIVOT, self.rect.center)
        pygame.draw.line(screen, (0,0,0), (0, PIVOT[1]), (WINDOWSIZE, PIVOT

    def update(self):
        self.recomputeAngle()
        screen.fill((255,255,255))
        self.draw()

bob = BobMass()

TICK = USEREVENT + 2
pygame.time.set_timer(TICK, TIMETICK)

def input(events):
    for event in events:
        if event.type == QUIT:
            sys.exit(0)
        elif event.type == TICK:
            bob.update()
```

```
while True:
    input(pygame.event.get())
    pygame.display.flip()
```

Python: Penney's_game[edit]

```
from __future__ import print_function
import random
from time import sleep

first = random.choice([True, False])

you = ''
if first:
    me = ''.join(random.sample('HT'*3, 3))
    print('I choose first and will win on first seeing {} in the list of to
    while len(you) != 3 or any(ch not in 'HT' for ch in you) or you == me:
        you = input('What sequence of three Heads/Tails will you win with:
else:
    while len(you) != 3 or any(ch not in 'HT' for ch in you):
        you = input('After you: What sequence of three Heads/Tails will you
    me = ('H' if you[1] == 'T' else 'T') + you[:2]
    print('I win on first seeing {} in the list of tosses'.format(me))

print('Rolling:\n  ', end='')
rolled = ''
while True:
    rolled += random.choice('HT')
    print(rolled[-1], end='')
    if rolled.endswith(you):
        print('\n  You win!')
        break
    if rolled.endswith(me):
        print('\n  I win!')
        break
    sleep(1)    # For dramatic effect
```

Output:

Python: Pentagram[edit]

Works with: Python version 3.4.1

```
import turtle

turtle.bgcolor("green")
t = turtle.Turtle()
t.color("red", "blue")
t.begin_fill()
for i in range(0, 5):
    t.forward(200)
    t.right(144)
t.end_fill()
```

Racket[edit]

Python: Percentage_difference_between_images[edit

You must install the Python Imaging Library to use this example.

Python: Perfect_Numbers[edit]

```
def perf(n):
    sum = 0
    for i in xrange(1, n):
        if n % i == 0:
            sum += i
    return sum == n
```

Functional style:

```
perf = lambda n: n == sum(i for i in xrange(1, n) if n % i == 0)
```

Python: Perfect_numbers[edit]

```
def perf(n):
    sum = 0
    for i in xrange(1, n):
        if n % i == 0:
```

```
            sum += i
    return sum == n
```

Functional style:

```
perf = lambda n: n == sum(i for i in xrange(1, n) if n % i == 0)
```

Python: Perfect_shuffle[edit]

```
import doctest
import random


def flatten(lst):
    """
    >>> flatten([[3,2],[1,2]])
    [3, 2, 1, 2]
    """
    return [i for sublst in lst for i in sublst]

def magic_shuffle(deck):
    """
    >>> magic_shuffle([1,2,3,4])
    [1, 3, 2, 4]
    """
    half = len(deck) // 2
    return flatten(zip(deck[:half], deck[half:]))

def after_how_many_is_equal(shuffle_type,start,end):
    """
    >>> after_how_many_is_equal(magic_shuffle,[1,2,3,4],[1,2,3,4])
    2
    """

    start = shuffle_type(start)
    counter = 1
    while start != end:
        start = shuffle_type(start)
        counter += 1
    return counter

def main():
    doctest.testmod()

    print("Length of the deck of cards | Perfect shuffles needed to obtain t
    for length in (8, 24, 52, 100, 1020, 1024, 10000):
        deck = list(range(length))
```

```
        shuffles_needed = after_how_many_is_equal(magic_shuffle,deck,deck)
        print("{} | {}".format(length,shuffles_needed))


if __name__ == "__main__":
    main()
```

Reversed shuffle or just calculate how many shuffles are needed:

```
def mul_ord2(n):
        # directly calculate how many shuffles are needed to restore
        # initial order: 2^o mod(n-1) == 1
        if n == 2: return 1

        n,t,o = n-1,2,1
        while t != 1:
                t,o = (t*2)%n,o+1
        return o

def shuffles(n):
        a,c = list(range(n)), 0
        b = a

        while True:
                # Reverse shuffle; a[i] can be taken as the current
                # position of the card with value i.  This is faster.
                a = a[0:n:2] + a[1:n:2]
                c += 1
                if b == a: break
        return c

for n in range(2, 10000, 2):
        #print(n, mul_ord2(n))
        print(n, shuffles(n))
```

Python: Perlin_noise[edit]

Translation of: Java

```
def perlin_noise(x, y, z):
    X = int(x) & 255                  # FIND UNIT CUBE THAT
    Y = int(y) & 255                  # CONTAINS POINT.
    Z = int(z) & 255
    x -= int(x)                                # FIND RELATIVE X,Y,Z
    y -= int(y)                                # OF POINT IN CUBE.
    z -= int(z)
```

```
    u = fade(x)                                # COMPUTE FADE CURVES
    v = fade(y)                                # FOR EACH OF X,Y,Z.
    w = fade(z)
    A = p[X  ]+Y; AA = p[A]+Z; AB = p[A+1]+Z      # HASH COORDINATES OF
    B = p[X+1]+Y; BA = p[B]+Z; BB = p[B+1]+Z      # THE 8 CUBE CORNERS,

    return lerp(w, lerp(v, lerp(u, grad(p[AA  ], x  , y  , z   ),  # AND AD
                                   grad(p[BA  ], x-1, y  , z   )), # BLENDE
                           lerp(u, grad(p[AB  ], x  , y-1, z   ),  # RESULT
                                   grad(p[BB  ], x-1, y-1, z   ))),# FROM  
                   lerp(v, lerp(u, grad(p[AA+1], x  , y  , z-1 ),  # CORNER
                                   grad(p[BA+1], x-1, y  , z-1 )), # OF CUB
                           lerp(u, grad(p[AB+1], x  , y-1, z-1 ),
                                   grad(p[BB+1], x-1, y-1, z-1 ))))

def fade(t):
    return t ** 3 * (t * (t * 6 - 15) + 10)

def lerp(t, a, b):
    return a + t * (b - a)

def grad(hash, x, y, z):
    h = hash & 15                      # CONVERT LO 4 BITS OF HASH CODE
    u = x if h<8 else y                # INTO 12 GRADIENT DIRECTIONS.
    v = y if h<4 else (x if h in (12, 14) else z)
    return (u if (h&1) == 0 else -u) + (v if (h&2) == 0 else -v)

p = [None] * 512
permutation = [151,160,137,91,90,15,
   131,13,201,95,96,53,194,233,7,225,140,36,103,30,69,142,8,99,37,240,21,10
   190, 6,148,247,120,234,75,0,26,197,62,94,252,219,203,117,35,11,32,57,177
   88,237,149,56,87,174,20,125,136,171,168, 68,175,74,165,71,134,139,48,27,
   77,146,158,231,83,111,229,122,60,211,133,230,220,105,92,41,55,46,245,40,
   102,143,54, 65,25,63,161, 1,216,80,73,209,76,132,187,208, 89,18,169,200,
   135,130,116,188,159,86,164,100,109,198,173,186, 3,64,52,217,226,250,124,
   5,202,38,147,118,126,255,82,85,212,207,206,59,227,47,16,58,17,182,189,28
   223,183,170,213,119,248,152, 2,44,154,163, 70,221,153,101,155,167, 43,17
   129,22,39,253, 19,98,108,110,79,113,224,232,178,185, 112,104,218,246,97,
   251,34,242,193,238,210,144,12,191,179,162,241, 81,51,145,235,249,14,239,
   49,192,214, 31,181,199,106,157,184, 84,204,176,115,121,50,45,127, 4,150,
   138,236,205,93,222,114,67,29,24,72,243,141,128,195,78,66,215,61,156,180]
for i in range(256):
    p[256+i] = p[i] = permutation[i]

if __name__ == '__main__':
    print("%1.17f" % perlin_noise(3.14, 42, 7))
```

Output:

Python: Permutation_test[edit]

Translation of: Tcl

```
from itertools import combinations as comb

def statistic(ab, a):
    sumab, suma = sum(ab), sum(a)
    return ( suma / len(a) -
             (sumab -suma) / (len(ab) - len(a)) )

def permutationTest(a, b):
    ab = a + b
    Tobs = statistic(ab, a)
    under = 0
    for count, perm in enumerate(comb(ab, len(a)), 1):
        if statistic(ab, perm) <= Tobs:
            under += 1
    return under * 100. / count

treatmentGroup = [85, 88, 75, 66, 25, 29, 83, 39, 97]
controlGroup   = [68, 41, 10, 49, 16, 65, 32, 92, 28, 98]
under = permutationTest(treatmentGroup, controlGroup)
print("under=%.2f%%, over=%.2f%%" % (under, 100. - under))
```

Output:

```
under=89.11%, over=10.89%
```

Python: Permutations_with_repetitions[edit]

```
from itertools import product

# check permutations until we find the word 'crack'
for x in product('ACRK', repeat=5):
    w = ''.join(x)
    print w
    if w.lower() == 'crack': break
```

Python: Pernicious_numbers[edit]

```
>>> def popcount(n): return bin(n).count("1")
```

```
>>> primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 5
>>> p, i = [], 0
>>> while len(p) < 25:
        if popcount(i) in primes: p.append(i)
        i += 1



>>> p
[3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 17, 18, 19, 20, 21, 22, 24, 25, 26, 28,
>>> p, i = [], 888888877
>>> while i <= 888888888:
        if popcount(i) in primes: p.append(i)
        i += 1



>>> p
[888888877, 888888878, 888888880, 888888883, 888888885, 888888886]
>>>
```

Python: Phrase_reversals[edit]

These examples use the extended slicing notation of [::-1] to reverse strin

```
>>> phrase = "rosetta code phrase reversal"
>>> phrase[::-1]                                          # Reversed.
'lasrever esarhp edoc attesor'
>>> ' '.join(word[::-1] for word in phrase.split())       # Words reversed.
'attesor edoc esarhp lasrever'
>>> ' '.join(word for word in phrase.split()[::-1])       # Word order reve
'reversal phrase code rosetta'
>>>
```

Python: Pi[edit]

```
def calcPi():
    q, r, t, k, n, l = 1, 0, 1, 1, 3, 3
    while True:
        if 4*q+r-t < n*t:
            yield n
            nr = 10*(r-n*t)
            n  = ((10*(3*q+r))//t)-10*n
            q  *= 10
            r  = nr
        else:
```

```
            nr = (2*q+r)*l
            nn = (q*(7*k)+2+(r*l))//(t*l)
            q  *= k
            t  *= l
            l  += 2
            k += 1
            n  = nn
            r  = nr

import sys
pi_digits = calcPi()
i = 0
for d in pi_digits:
    sys.stdout.write(str(d))
    i += 1
    if i == 40: print(""); i = 0
```

output

```
3141592653589793238462643383279502884197
```

Python: Pick_random_element[edit]

```
>>> import random
>>> random.choice(['foo', 'bar', 'baz'])
'baz'
```

R[edit]

Python: Play_recorded_sounds[edit]

Works with: Python version 2.6

Library: pygame

Pygame is a library for cross-platform game development and depends on the

```
import time
from pygame import mixer

mixer.init(frequency=16000) #set frequency for wav file
s1 = mixer.Sound('test.wav')
s2 = mixer.Sound('test2.wav')
```

```
#individual
s1.play(-1)         #loops indefinitely
time.sleep(0.5)

#simultaneously
s2.play()          #play once
time.sleep(2)
s2.play(2)         #optional parameter loops three times
time.sleep(10)

#set volume down
s1.set_volume(0.1)
time.sleep(5)

#set volume up
s1.set_volume(1)
time.sleep(5)

s1.stop()
s2.stop()
mixer.quit()
```

To play back .mp3 (or .ogg) files, the music import is used.

```
import time
from pygame import mixer
from pygame.mixer import music

mixer.init()
music.load('test.mp3')

music.play()
time.sleep(10)

music.stop()
mixer.quit()
```

Python: Playfair_cipher[edit]

Translation of: Perl 6

```
from string import ascii_uppercase
from itertools import product
from re import findall

def uniq(seq):
    seen = {}
    return [seen.setdefault(x, x) for x in seq if x not in seen]
```

```
def partition(seq, n):
    return [seq[i : i + n] for i in xrange(0, len(seq), n)]


"""Instantiate a specific encoder/decoder."""
def playfair(key, from_ = 'J', to = None):
    if to is None:
        to = 'I' if from_ == 'J' else ''

    def canonicalize(s):
        return filter(str.isupper, s.upper()).replace(from_, to)

    # Build 5x5 matrix.
    m = partition(uniq(canonicalize(key + ascii_uppercase)), 5)

    # Pregenerate all forward translations.
    enc = {}

    # Map pairs in same row.
    for row in m:
        for i, j in product(xrange(5), repeat=2):
            if i != j:
                enc[row[i] + row[j]] = row[(i + 1) % 5] + row[(j + 1) % 5]

    # Map pairs in same column.
    for c in zip(*m):
        for i, j in product(xrange(5), repeat=2):
            if i != j:
                enc[c[i] + c[j]] = c[(i + 1) % 5] + c[(j + 1) % 5]

    # Map pairs with cross-connections.
    for i1, j1, i2, j2 in product(xrange(5), repeat=4):
        if i1 != i2 and j1 != j2:
            enc[m[i1][j1] + m[i2][j2]] = m[i1][j2] + m[i2][j1]

    # Generate reverse translations.
    dec = dict((v, k) for k, v in enc.iteritems())

    def sub_enc(txt):
        lst = findall(r"(.)(?:(?!\1)(.))?", canonicalize(txt))
        return " ".join(enc[a + (b if b else 'X')] for a, b in lst)

    def sub_dec(encoded):
        return " ".join(dec[p] for p in partition(canonicalize(encoded), 2))

    return sub_enc, sub_dec


(encode, decode) = playfair("Playfair example")
orig = "Hide the gold in...the TREESTUMP!!!"
print "Original:", orig
enc = encode(orig)
print "Encoded:", enc
print "Decoded:", decode(enc)
```

Output:

```
Original: Hide the gold in...the TREESTUMP!!!
Encoded: BM OD ZB XD NA BE KU DM UI XM MO UV IF
Decoded: HI DE TH EG OL DI NT HE TR EX ES TU MP
```

REXX[edit]

Python: Plot_x,_y_arrays[edit]

Library: matplotlib
[edit]

matplotlib plot of x,y arrays

Interactive session:

```
>>> x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> y = [2.7, 2.8, 31.4, 38.1, 58.0, 76.2, 100.5, 130.0, 149.3, 180.0]

>>> import pylab
>>> pylab.plot(x, y, 'bo')
>>> pylab.savefig('qsort-range-10-9.png')
```

See some other examples:

[Python: Point_in_polygon_(ray_casting_algorithm)]

```
from collections import namedtuple
from pprint import pprint as pp
import sys

Pt = namedtuple('Pt', 'x, y')                   # Point
Edge = namedtuple('Edge', 'a, b')               # Polygon edge from a to b
Poly = namedtuple('Poly', 'name, edges')        # Polygon

_eps = 0.00001
_huge = sys.float_info.max
_tiny = sys.float_info.min

def rayintersectseg(p, edge):
    ''' takes a point p=Pt() and an edge of two endpoints a,b=Pt() of a lin
    '''
    a,b = edge
    if a.y > b.y:
        a,b = b,a
    if p.y == a.y or p.y == b.y:
        p = Pt(p.x, p.y + _eps)

    intersect = False

    if (p.y > b.y or p.y < a.y) or (
        p.x > max(a.x, b.x)):
        return False

    if p.x < min(a.x, b.x):
        intersect = True
    else:
        if abs(a.x - b.x) > _tiny:
            m_red = (b.y - a.y) / float(b.x - a.x)
        else:
            m_red = _huge
        if abs(a.x - p.x) > _tiny:
            m_blue = (p.y - a.y) / float(p.x - a.x)
        else:
            m_blue = _huge
        intersect = m_blue >= m_red
    return intersect

def _odd(x): return x%2 == 1

def ispointinside(p, poly):
    ln = len(poly)
```

```
    return _odd(sum(rayintersectseg(p, edge)
                    for edge in poly.edges ))

def polypp(poly):
    print ("\n  Polygon(name='%s', edges=(" % poly.name)
    print ('   ', ',\n    '.join(str(e) for e in poly.edges) + '\n    ))')

if __name__ == '__main__':
    polys = [
      Poly(name='square', edges=(
        Edge(a=Pt(x=0, y=0), b=Pt(x=10, y=0)),
        Edge(a=Pt(x=10, y=0), b=Pt(x=10, y=10)),
        Edge(a=Pt(x=10, y=10), b=Pt(x=0, y=10)),
        Edge(a=Pt(x=0, y=10), b=Pt(x=0, y=0))
        )),
      Poly(name='square_hole', edges=(
        Edge(a=Pt(x=0, y=0), b=Pt(x=10, y=0)),
        Edge(a=Pt(x=10, y=0), b=Pt(x=10, y=10)),
        Edge(a=Pt(x=10, y=10), b=Pt(x=0, y=10)),
        Edge(a=Pt(x=0, y=10), b=Pt(x=0, y=0)),
        Edge(a=Pt(x=2.5, y=2.5), b=Pt(x=7.5, y=2.5)),
        Edge(a=Pt(x=7.5, y=2.5), b=Pt(x=7.5, y=7.5)),
        Edge(a=Pt(x=7.5, y=7.5), b=Pt(x=2.5, y=7.5)),
        Edge(a=Pt(x=2.5, y=7.5), b=Pt(x=2.5, y=2.5))
        )),
      Poly(name='strange', edges=(
        Edge(a=Pt(x=0, y=0), b=Pt(x=2.5, y=2.5)),
        Edge(a=Pt(x=2.5, y=2.5), b=Pt(x=0, y=10)),
        Edge(a=Pt(x=0, y=10), b=Pt(x=2.5, y=7.5)),
        Edge(a=Pt(x=2.5, y=7.5), b=Pt(x=7.5, y=7.5)),
        Edge(a=Pt(x=7.5, y=7.5), b=Pt(x=10, y=10)),
        Edge(a=Pt(x=10, y=10), b=Pt(x=10, y=0)),
        Edge(a=Pt(x=10, y=0), b=Pt(x=2.5, y=2.5))
        )),
      Poly(name='exagon', edges=(
        Edge(a=Pt(x=3, y=0), b=Pt(x=7, y=0)),
        Edge(a=Pt(x=7, y=0), b=Pt(x=10, y=5)),
        Edge(a=Pt(x=10, y=5), b=Pt(x=7, y=10)),
        Edge(a=Pt(x=7, y=10), b=Pt(x=3, y=10)),
        Edge(a=Pt(x=3, y=10), b=Pt(x=0, y=5)),
        Edge(a=Pt(x=0, y=5), b=Pt(x=3, y=0))
        )),
      ]
    testpoints = (Pt(x=5, y=5), Pt(x=5, y=8),
                  Pt(x=-10, y=5), Pt(x=0, y=5),
                  Pt(x=10, y=5), Pt(x=8, y=5),
                  Pt(x=10, y=10))

    print ("\n TESTING WHETHER POINTS ARE WITHIN POLYGONS")
    for poly in polys:
        polypp(poly)
        print ('   ', '\t'.join("%s: %s" % (p, ispointinside(p, poly))
                               for p in testpoints[:3]))
        print ('   ', '\t'.join("%s: %s" % (p, ispointinside(p, poly))
                               for p in testpoints[3:6]))
        print ('   ', '\t'.join("%s: %s" % (p, ispointinside(p, poly))
```

```
                                for p in testpoints[6:]))
```

Sample output

```
 TESTING WHETHER POINTS ARE WITHIN POLYGONS

  Polygon(name='square', edges=(
    Edge(a=Pt(x=0, y=0), b=Pt(x=10, y=0)),
    Edge(a=Pt(x=10, y=0), b=Pt(x=10, y=10)),
    Edge(a=Pt(x=10, y=10), b=Pt(x=0, y=10)),
    Edge(a=Pt(x=0, y=10), b=Pt(x=0, y=0))
    ))
    Pt(x=5, y=5): True  Pt(x=5, y=8): True      Pt(x=-10, y=5): False
    Pt(x=0, y=5): False Pt(x=10, y=5): True     Pt(x=8, y=5): True
    Pt(x=10, y=10): False

  Polygon(name='square_hole', edges=(
    Edge(a=Pt(x=0, y=0), b=Pt(x=10, y=0)),
    Edge(a=Pt(x=10, y=0), b=Pt(x=10, y=10)),
    Edge(a=Pt(x=10, y=10), b=Pt(x=0, y=10)),
```

Helper routine to convert Fortran Polygons and points to Python

```
def _convert_fortran_shapes():
    point = Pt
    pts = (point(0,0), point(10,0), point(10,10), point(0,10),
           point(2.5,2.5), point(7.5,2.5), point(7.5,7.5), point(2.5,7.5),
           point(0,5), point(10,5),
           point(3,0), point(7,0), point(7,10), point(3,10))
    p = (point(5,5), point(5, 8), point(-10, 5), point(0,5), point(10,5),
         point(8,5), point(10,10) )

    def create_polygon(pts,vertexindex):
        return [tuple(Edge(pts[vertexindex[i]-1], pts[vertexindex[i+1]-1])
                       for i in range(0, len(vertexindex), 2) )]
    polys=[]
    polys += create_polygon(pts, ( 1,2, 2,3, 3,4, 4,1 ) )
    polys += create_polygon(pts, ( 1,2, 2,3, 3,4, 4,1, 5,6, 6,7, 7,8, 8,5 )
    polys += create_polygon(pts, ( 1,5, 5,4, 4,8, 8,7, 7,3, 3,2, 2,5 ) )
    polys += create_polygon(pts, ( 11,12, 12,10, 10,13, 13,14, 14,9, 9,11 )

    names = ( "square", "square_hole", "strange", "exagon" )
    polys = [Poly(name, edges)
             for name, edges in zip(names, polys)]
    print 'polys = ['
    for p in polys:
        print "  Poly(name='%s', edges=(" % p.name
```

```
        print '    ', ',\n    '.join(str(e) for e in p.edges) + '\n    )),'
    print '  ]'
_convert_fortran_shapes()
```

Python: Pointers_and_references[edit]

Python does not have pointers and all Python names (variables) are implicit

```
# Bind a literal string object to a name:
a = "foo"
# Bind an empty list to another name:
b = []
# Classes are "factories" for creating new objects: invoke class name as a
class Foo(object):
    pass
c = Foo()
# Again, but with optional initialization:
class Bar(object):
    def __init__(self, initializer = None)
        # "initializer is an arbitrary identifier, and "None" is an arbitr
        if initializer is not None:
           self.value = initializer
d = Bar(10)
print d.value
# Test if two names are references to the same object:
if a is b: pass
# Alternatively:
if id(a) == id(b): pass
# Re-bind a previous used name to a function:
def a(fmt, *args):
    if fmt is None:
        fmt = "%s"
     print fmt % (args)
# Append reference to a list:
b.append(a)
# Unbind a reference:
del(a)
# Call (anymous function object) from inside a list
b[0]("foo")  # Note that the function object we original bound to the name
             # even if its name is unbound or rebound to some other object
```

[Note: in some ways this task is meaningless for Python given the nature of

Racket[edit]

Python: Poker_hand_analyser[edit]

Goes a little further in also giving the ordered tie-breaker information fr

```
from collections import namedtuple

class Card(namedtuple('Card', 'face, suit')):
    def __repr__(self):
        return ''.join(self)


suit = '♥ ♦ ♣ ♠'.split()
# ordered strings of faces
faces   = '2 3 4 5 6 7 8 9 10 j q k a'
lowaces = 'a 2 3 4 5 6 7 8 9 10 j q k'
# faces as lists
face   = faces.split()
lowace = lowaces.split()


def straightflush(hand):
    f,fs = ( (lowace, lowaces) if any(card.face == '2' for card in hand)
             else (face, faces) )
    ordered = sorted(hand, key=lambda card: (f.index(card.face), card.suit)
    first, rest = ordered[0], ordered[1:]
    if ( all(card.suit == first.suit for card in rest) and
         ' '.join(card.face for card in ordered) in fs ):
        return 'straight-flush', ordered[-1].face
    return False

def fourofakind(hand):
    allfaces = [f for f,s in hand]
    allftypes = set(allfaces)
    if len(allftypes) != 2:
        return False
    for f in allftypes:
        if allfaces.count(f) == 4:
            allftypes.remove(f)
            return 'four-of-a-kind', [f, allftypes.pop()]
    else:
        return False

def fullhouse(hand):
    allfaces = [f for f,s in hand]
    allftypes = set(allfaces)
    if len(allftypes) != 2:
        return False
    for f in allftypes:
        if allfaces.count(f) == 3:
```

```
            allftypes.remove(f)
            return 'full-house', [f, allftypes.pop()]
    else:
        return False

def flush(hand):
    allstypes = {s for f, s in hand}
    if len(allstypes) == 1:
        allfaces = [f for f,s in hand]
        return 'flush', sorted(allfaces,
                               key=lambda f: face.index(f),
                               reverse=True)
    return False

def straight(hand):
    f,fs = ( (lowace, lowaces) if any(card.face == '2' for card in hand)
             else (face, faces) )
    ordered = sorted(hand, key=lambda card: (f.index(card.face), card.suit)
    first, rest = ordered[0], ordered[1:]
    if ' '.join(card.face for card in ordered) in fs:
        return 'straight', ordered[-1].face
    return False

def threeofakind(hand):
    allfaces = [f for f,s in hand]
    allftypes = set(allfaces)
    if len(allftypes) <= 2:
        return False
    for f in allftypes:
        if allfaces.count(f) == 3:
            allftypes.remove(f)
            return ('three-of-a-kind', [f] +
                     sorted(allftypes,
                            key=lambda f: face.index(f),
                            reverse=True))
    else:
        return False

def twopair(hand):
    allfaces = [f for f,s in hand]
    allftypes = set(allfaces)
    pairs = [f for f in allftypes if allfaces.count(f) == 2]
    if len(pairs) != 2:
        return False
    p0, p1 = pairs
    other = [(allftypes - set(pairs)).pop()]
    return 'two-pair', pairs + other if face.index(p0) > face.index(p1) els

def onepair(hand):
    allfaces = [f for f,s in hand]
    allftypes = set(allfaces)
    pairs = [f for f in allftypes if allfaces.count(f) == 2]
    if len(pairs) != 1:
        return False
    allftypes.remove(pairs[0])
    return 'one-pair', pairs + sorted(allftypes,
```

```
                                        key=lambda f: face.index(f),
                                        reverse=True)

def highcard(hand):
    allfaces = [f for f,s in hand]
    return 'high-card', sorted(allfaces,
                               key=lambda f: face.index(f),
                               reverse=True)

handrankorder =  (straightflush, fourofakind, fullhouse,
                  flush, straight, threeofakind,
                  twopair, onepair, highcard)

def rank(cards):
    hand = handy(cards)
    for ranker in handrankorder:
        rank = ranker(hand)
        if rank:
            break
    assert rank, "Invalid: Failed to rank cards: %r" % cards
    return rank

def handy(cards='2♥ 2♦ 2♣ k♣ q♦'):
    hand = []
    for card in cards.split():
        f, s = card[:-1], card[-1]
        assert f in face, "Invalid: Don't understand card face %r" % f
        assert s in suit, "Invalid: Don't understand card suit %r" % s
        hand.append(Card(f, s))
    assert len(hand) == 5, "Invalid: Must be 5 cards in a hand, not %i" % le
    assert len(set(hand)) == 5, "Invalid: All cards in the hand must be uni
    return hand


if __name__ == '__main__':
    hands = ["2♥ 2♦ 2♣ k♣ q♦",
     "2♥ 5♥ 7♦ 8♣ 9♠",
     "a♥ 2♦ 3♣ 4♣ 5♦",
     "2♥ 3♥ 2♦ 3♣ 3♦",
     "2♥ 7♥ 2♦ 3♣ 3♦",
     "2♥ 7♥ 7♦ 7♣ 7♠",
     "10♥ j♥ q♥ k♥ a♥"] + [
     "4♥ 4♠ k♠ 5♦ 10♠",
     "q♣ 10♣ 7♣ 6♣ 4♣",
     ]
    print("%-18s %-15s %s" % ("HAND", "CATEGORY", "TIE-BREAKER"))
    for cards in hands:
        r = rank(cards)
        print("%-18r %-15s %r" % (cards, r[0], r[1]))
```

Output:

Python: Polymorphic_copy[edit]

```
import copy

class T:
   def classname(self):
      return self.__class__.__name__

   def __init__(self):
      self.myValue = "I'm a T."

   def speak(self):
      print self.classname(), 'Hello', self.myValue

   def clone(self):
      return copy.copy(self)

class S1(T):
   def speak(self):
      print self.classname(),"Meow", self.myValue

class S2(T):
   def speak(self):
      print self.classname(),"Woof", self.myValue


print "creating initial objects of types S1, S2, and T"
a = S1()
a.myValue = 'Green'
a.speak()

b = S2()
b.myValue = 'Blue'
b.speak()

u = T()
u.myValue = 'Purple'
u.speak()

print "Making copy of a as u, colors and types should match"
u = a.clone()
u.speak()
a.speak()
print "Assigning new color to u, A's color should be unchanged."
u.myValue = "Orange"
u.speak()
a.speak()

print "Assigning u to reference same object as b, colors and types should m
u = b
u.speak()
b.speak()
print "Assigning new color to u. Since u,b references same object b's color
```

```
u.myValue = "Yellow"
u.speak()
b.speak()
```

Output:

```
creating initial objects of types S1, S2, and T
S1 Meow Green
S2 Woof Blue
T Hello Purple
Making copy of a as u, colors and types should match
S1 Meow Green
S1 Meow Green
Assigning new color to u, A's color should be unchanged.
S1 Meow Orange
S1 Meow Green
Assigning u to reference same object as b, colors and types should match
S2 Woof Blue
S2 Woof Blue
Assigning new color to u. Since u,b references same object b's color change
S2 Woof Yellow
S2 Woof Yellow
```

The foregoing example uses the Python standard library *copy* module.
The task, as stated,does not provide insight as to what should happen shoul
It could be necessary to use "deep copy" instead of copy. (using *copy.deepc*
The distinction is important for complex objects containing references to o
The described task, as presented, offers no guidance on this matter.

In many cases the most portable and robust "copy" would be made by serializ
Under Python this would best be done with the *pickle* or *cPickle* standard mo

```
import cPickle as pickle

source = {'a': [1, 2.0, 3, 4+6j],
          'b': ('string', u'Unicode string'),
          'c': None}

target = pickle.loads(pickle.dumps(source))
```

In this example we use the *cPickle* module which is an implementation of the
We import it as *pickle* since we intend to use only those features which are
(The pure Python implementation is retained for those who which to create t
The *dumps()* and *loads()* methods dump the data structures to a string and lo
For those we'd use the *pickle.dump()* and *pickle.load()* methods).

For the simplest cases one can use simple Python introspection to copy simp

```
target = source.__class__()  # Create an object of the same type
if hasattr(source, 'items') and callable(source.items):
    for key,value in source.items:
        target[key] = value
elif hasattr(source, '__len__'):
    target = source[:]
else:  # Following is not recommended. (see below).
    target = source
```

This example handles dictionaries (and anything that implements a sufficien
Similarly this code tests if an item is a sequence (one can call the "len()
For any other type of object a simple binding is performed.
Technically this last case will not "copy" anything ... it will create a ne
The earlier binding of a "blank" instance of the source's __class__ will be
So the trick of creating the blank object of the same type is only meaningf
In the cases of strings, integers and other numbers the objects themselves

Python: Polynomial_Fitting[edit]

Library: numpy

```
>>> x = [0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]
>>> y = [1,   6,  17,  34,  57,  86, 121, 162, 209, 262, 321]
>>> coeffs = numpy.polyfit(x,y,deg=2)
>>> coeffs
array([ 3.,  2.,  1.])
```

Substitute back received coefficients.

```
>>> yf = numpy.polyval(numpy.poly1d(coeffs), x)
>>> yf
array([   1.,    6.,   17.,   34.,   57.,   86.,  121.,  162.,  209.,  262.,
```

Find max absolute error:

```
>>> '%.1g' % max(y-yf)
'1e-013'
```

Example[edit]

For input arrays `x' and `y':

```
>>> x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> y = [2.7, 2.8, 31.4, 38.1, 58.0, 76.2, 100.5, 130.0, 149.3, 180.0]
```

```
>>> p = numpy.poly1d(numpy.polyfit(x, y, deg=2), variable='N')
>>> print p
       2
1.085 N + 10.36 N - 0.6164
```

Thus we confirm once more that for already sorted sequences the considered quick sort implementation has quadratic dependence on sequence length (see **Example** section for Python language on *Query Performance* page).

R[edit]

Python: Polynomial_long_division[edit]

Works with: Python 2.x

```
# -*- coding: utf-8 -*-

from itertools import izip
from math import fabs

def degree(poly):
    while poly and poly[-1] == 0:
        poly.pop()   # normalize
    return len(poly)-1

def poly_div(N, D):
    dD = degree(D)
    dN = degree(N)
    if dD < 0: raise ZeroDivisionError
    if dN >= dD:
        q = [0] * dN
        while dN >= dD:
```

```
            d = [0]*(dN - dD) + D
            mult = q[dN - dD] = N[-1] / float(d[-1])
            d = [coeff*mult for coeff in d]
            N = [fabs ( coeffN - coeffd ) for coeffN, coeffd in izip(N, d)]
            dN = degree(N)
        r = N
    else:
        q = [0]
        r = N
    return q, r

if __name__ == '__main__':
    print "POLYNOMIAL LONG DIVISION"
    N = [-42, 0, -12, 1]
    D = [-3, 1, 0, 0]
    print "  %s / %s =" % (N,D),
    print " %s remainder %s" % poly_div(N, D)
```

Sample output:

```
POLYNOMIAL LONG DIVISION
  [-42, 0, -12, 1] / [-3, 1, 0, 0] =  [-27.0, -9.0, 1.0] remainder [-123.0]
```

R[edit]

Python: Polynomial_regression[edit]

Library: numpy

```
>>> x = [0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]
>>> y = [1,   6,  17,  34,  57,  86, 121, 162, 209, 262, 321]
>>> coeffs = numpy.polyfit(x,y,deg=2)
>>> coeffs
array([ 3.,  2.,  1.])
```

Substitute back received coefficients.

```
>>> yf = numpy.polyval(numpy.poly1d(coeffs), x)
>>> yf
array([   1.,    6.,   17.,   34.,   57.,   86.,  121.,  162.,  209.,  262.,
```

Find max absolute error:

Python: Polynomial_synthetic_division[edit]

Here is an extended synthetic division algorithm, which means that it suppo

Works with: Python 2.x

```
# -*- coding: utf-8 -*-

def extended_synthetic_division(dividend, divisor):
    '''Fast polynomial division by using Extended Synthetic Division. Also 
    # dividend and divisor are both polynomials, which are here simply list

    out = list(dividend) # Copy the dividend
    normalizer = divisor[0]
    for i in xrange(len(dividend)-(len(divisor)-1)):
        out[i] /= normalizer # for general polynomial division (when polyno
                                 # we need to normalize by dividing the coe
        coef = out[i]
        if coef != 0: # useless to multiply if coef is 0
            for j in xrange(1, len(divisor)): # in synthetic division, we a
                                             # because it's only used to n
                out[i + j] += -divisor[j] * coef

    # The resulting out contains both the quotient and the remainder, the r
    # has necessarily the same degree as the divisor since it's what we cou
    # where this separation is, and return the quotient and remainder.
    separator = -(len(divisor)-1)
    return out[:separator], out[separator:] # return quotient, remainder.

if __name__ == '__main__':
    print "POLYNOMIAL SYNTHETIC DIVISION"
    N = [1, -12, 0, -42]
    D = [1, -3]
    print "  %s / %s =" % (N,D),
    print " %s remainder %s" % extended_synthetic_division(N, D)
```

Sample output:

```
POLYNOMIAL SYNTHETIC DIVISION
  [1, -12, 0, -42] / [1, -3] =  [1, -9, -27] remainder [-123]
```

Racket[edit]

Python: Population_count[edit]

```
>>> def popcount(n): return bin(n).count("1")
...
>>> [popcount(3**i) for i in range(30)]
[1, 2, 2, 4, 3, 6, 6, 5, 6, 8, 9, 13, 10, 11, 14, 15, 11, 14, 14, 17, 17, 2
>>> evil, odious, i = [], [], 0
>>> while len(evil) < 30 or len(odious) < 30:
...     p = popcount(i)
...     if p % 2: odious.append(i)
...     else: evil.append(i)
...     i += 1
...
>>> evil[:30]
[0, 3, 5, 6, 9, 10, 12, 15, 17, 18, 20, 23, 24, 27, 29, 30, 33, 34, 36, 39,
>>> odious[:30]
[1, 2, 4, 7, 8, 11, 13, 14, 16, 19, 21, 22, 25, 26, 28, 31, 32, 35, 37, 38,
>>>
```

Python: Pragmatic_directives[edit]

Python has the __future__ module which controls certain features:

Python 3.2

```
Python 3.2 (r32:88445, Feb 20 2011, 21:30:00) [MSC v.1500 64 bit (AMD64)] o
Type "copyright", "credits" or "license()" for more information.
>>> import __future__
>>> __future__.all_feature_names
['nested_scopes', 'generators', 'division', 'absolute_import', 'with_statem
>>>
```

('barry_as_FLUFL' is an April fools joke)

Python 2.7

```
Python 2.7.2 (default, Jun 12 2011, 14:24:46) [MSC v.1500 64 bit (AMD64)] o
Type "copyright", "credits" or "license()" for more information.
>>> import __future__
>>> __future__.all_feature_names
['nested_scopes', 'generators', 'division', 'absolute_import', 'with_statem
>>>
```

Python: Price_Fraction[edit]

Using the bisect standard module to reduce the comparisons with members of

```
>>> import bisect
>>> _cin  = [.06, .11, .16, .21, .26, .31, .36, .41, .46, .51, .56, .61, .6
>>> _cout = [.10, .18, .26, .32, .38, .44, .50, .54, .58, .62, .66, .70, .7
>>> def pricerounder(pricein):
        return _cout[ bisect.bisect_right(_cin, pricein) ]
```

When dealing with money it is good to think about possible loss of precisio

```
>>> import bisect
>>> _cin  = [ 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81
>>> _cout = [10, 18, 26, 32, 38, 44, 50, 54, 58, 62, 66, 70, 74, 78, 82, 86
>>> def centsrounder(centsin):
        return _cout[ bisect.bisect_right(_cin, centsin) ]
```

Other options are to use the fractions or decimals modules for calculating

Bisection library code

The bisect Python standard library function uses the following code th

Python: Price_fraction[edit]

Using the bisect standard module to reduce the comparisons with members of

```
>>> import bisect
>>> _cin  = [.06, .11, .16, .21, .26, .31, .36, .41, .46, .51, .56, .61, .6
>>> _cout = [.10, .18, .26, .32, .38, .44, .50, .54, .58, .62, .66, .70, .7
>>> def pricerounder(pricein):
        return _cout[ bisect.bisect_right(_cin, pricein) ]
```

When dealing with money it is good to think about possible loss of precisio

```
>>> import bisect
>>> _cin  = [ 6, 11, 16, 21, 26, 31, 36, 41, 46, 51, 56, 61, 66, 71, 76, 81
>>> _cout = [10, 18, 26, 32, 38, 44, 50, 54, 58, 62, 66, 70, 74, 78, 82, 86
>>> def centsrounder(centsin):
        return _cout[ bisect.bisect_right(_cin, centsin) ]
```

Other options are to use the fractions or decimals modules for calculating

Bisection library code

The bisect Python standard library function uses the following code th

```
def bisect_right(a, x, lo=0, hi=None):
    """Return the index where to insert item x in list a, assuming a i

    The return value i is such that all e in a[:i] have e <= x, and al
    a[i:] have e > x.  So if x already appears in the list, a.insert(x
    insert just after the rightmost x already there.

    Optional args lo (default 0) and hi (default len(a)) bound the
    slice of a to be searched.
    """

    if lo < 0:
        raise ValueError('lo must be non-negative')
    if hi is None:
        hi = len(a)
    while lo < hi:
        mid = (lo+hi)//2
        if x < a[mid]: hi = mid
        else: lo = mid+1
    return lo
```

Python: Primality_by_Trial_Division[edit]

The simplest primality test, using trial division:

Works with: Python version 2.5

```
def prime(a):
    return not (a < 2 or any(a % x == 0 for x in xrange(2, int(a**0.5) + 1)
```

Another test. Exclude even numbers first:

```
def prime2(a):
    if a == 2: return True
    if a < 2 or a % 2 == 0: return False
    return not any(a % x == 0 for x in xrange(3, int(a**0.5) + 1, 2))
```

Yet another test. Exclude multiples of 2 and 3, see http://www.devx.com/vb2

Works with: Python version 2.4

```
def prime3(a):
    if a < 2: return False
    if a == 2 or a == 3: return True # manually test 2 and 3
    if a % 2 == 0 or a % 3 == 0: return False # exclude multiples of 2 and 

    maxDivisor = a**0.5
    d, i = 5, 2
    while d <= maxDivisor:
        if a % d == 0: return False
        d += i
        i = 6 - i # this modifies 2 into 4 and viceversa

    return True
```

Python: Primality_by_trial_division[edit]

The simplest primality test, using trial division:

Works with: Python version 2.5

```
def prime(a):
```

```
    return not (a < 2 or any(a % x == 0 for x in xrange(2, int(a**0.5) + 1)
```

Another test. Exclude even numbers first:

```
def prime2(a):
    if a == 2: return True
    if a < 2 or a % 2 == 0: return False
    return not any(a % x == 0 for x in xrange(3, int(a**0.5) + 1, 2))
```

Yet another test. Exclude multiples of 2 and 3, see http://www.devx.com/vb2

Works with: Python version 2.4

```
def prime3(a):
    if a < 2: return False
    if a == 2 or a == 3: return True # manually test 2 and 3
    if a % 2 == 0 or a % 3 == 0: return False # exclude multiples of 2 and 

    maxDivisor = a**0.5
    d, i = 5, 2
    while d <= maxDivisor:
        if a % d == 0: return False
        d += i
        i = 6 - i # this modifies 2 into 4 and viceversa

    return True
```

By Regular Expression[edit]

Regular expression by "Abigail".

(An explanation is given in "The Story of the Regexp and the Primes").

```
>>> import re
>>> def isprime(n):
    return not re.match(r'^1?$|^(11+?)\1+$', '1' * n)

>>> # A quick test
>>> [i for i in range(40) if isprime(i)]
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

Python: Primes_-_allocate_descendants_to_their_an

Python is very flexible, concise and effective with lists.

```
from __future__ import print_function
from itertools import takewhile

maxsum = 99

def get_primes(max):
    if max < 2: return []
    lprimes = [2]
    for x in range(3, max+1, 2):
        for p in lprimes:
            if x%p == 0: break
        else: lprimes.append(x)
    return lprimes

descendants = [[] for _ in range(maxsum + 1)]
ancestors = [[] for _ in range(maxsum + 1)]

primes = get_primes(maxsum)

for p in primes:
    descendants[p].append(p)
    for s in range(1, len(descendants)-p):
        descendants[s+p] += [p*pr for pr in descendants[s]]

for p in primes + [4]: descendants[p].pop()

total = 0
for s in range(1, maxsum + 1):
    descendants[s].sort()
    for d in takewhile(lambda x: x <= maxsum, descendants[s]):
        ancestors[d] = ancestors[s] + [s]
    print([s], "Level:", len(ancestors[s]))
    print("Ancestors:", ancestors[s] if len(ancestors[s]) else "None")
    print("Descendants:", len(descendants[s]) if len(descendants[s]) else "
    if len(descendants[s]): print(descendants[s])
    print()
    total += len(descendants[s])

print("Total descendants", total)
```

Python: Primorial_numbers[edit]

Uses the pure python library pyprimes .

```
from pyprimes import nprimes
from functools import reduce


primelist = list(nprimes(1000001))    # [2, 3, 5, ...]

def primorial(n):
    return reduce(int.__mul__, primelist[:n], 1)

if __name__ == '__main__':
    print('First ten primorals:', [primorial(n) for n in range(10)])
    for e in range(7):
        n = 10**e
        print('primorial(%i) has %i digits' % (n, len(str(primorial(n)))))
```

Python: Print_a_Stack_Trace[edit]

See the traceback module

```
import traceback

def f(): return g()
def g(): traceback.print_stack()

f()
```

Sample output from a session in the Idle IDE:

```
  File "<string>", line 1, in <module>
  File "C:\Python26\lib\idlelib\run.py", line 93, in main
    ret = method(*args, **kwargs)
  File "C:\Python26\lib\idlelib\run.py", line 293, in runcode
    exec code in self.locals
  File "C:/Documents and Settings/All Users/Documents/Paddys/traceback.py",
    f()
  File "C:/Documents and Settings/All Users/Documents/Paddys/traceback.py",
```

Python: Priority_queue[edit]

Using PriorityQueue[edit]

Python has the class queue.PriorityQueue in its standard library.

The data structures in the "queue" module are synchronized multi-producer,

```
>>> import queue
>>> pq = queue.PriorityQueue()
>>> for item in ((3, "Clear drains"), (4, "Feed cat"), (5, "Make tea"), (1,
  pq.put(item)


>>> while not pq.empty():
  print(pq.get_nowait())


(1, 'Solve RC tasks')
(2, 'Tax return')
(3, 'Clear drains')
(4, 'Feed cat')
(5, 'Make tea')
>>>
```

Help text for queue.PriorityQueue

```
>>> import queue
>>> help(queue.PriorityQueue)
Help on class PriorityQueue in module queue:

class PriorityQueue(Queue)
 |  Variant of Queue that retrieves open entries in priority order (lowest
 |
 |  Entries are typically tuples of the form:  (priority number, data).
 |
 |  Method resolution order:
 |      PriorityQueue
 |      Queue
 |      builtins.object
 |
 |  Methods inherited from Queue:
 |
 |  __init__(self, maxsize=0)
 |
 |  empty(self)
 |      Return True if the queue is empty, False otherwise (not reliable!).
 |
 |      This method is likely to be removed at some point.  Use qsize() == 
 |      as a direct substitute, but be aware that either approach risks a r
 |      condition where a queue can grow before the result of empty() or
 |      qsize() can be used.
```

```
 |
 |      To create code that needs to wait for all queued tasks to be
 |      completed, the preferred technique is to use the join() method.
 |
 |  full(self)
 |      Return True if the queue is full, False otherwise (not reliable!).
 |
 |      This method is likely to be removed at some point.  Use qsize() >=
 |      as a direct substitute, but be aware that either approach risks a r
 |      condition where a queue can shrink before the result of full() or
 |      qsize() can be used.
 |
 |  get(self, block=True, timeout=None)
 |      Remove and return an item from the queue.
 |
 |      If optional args 'block' is true and 'timeout' is None (the default
 |      block if necessary until an item is available. If 'timeout' is
 |      a positive number, it blocks at most 'timeout' seconds and raises
 |      the Empty exception if no item was available within that time.
 |      Otherwise ('block' is false), return an item if one is immediately
 |      available, else raise the Empty exception ('timeout' is ignored
 |      in that case).
 |
 |  get_nowait(self)
 |      Remove and return an item from the queue without blocking.
 |
 |      Only get an item if one is immediately available. Otherwise
 |      raise the Empty exception.
 |
 |  join(self)
 |      Blocks until all items in the Queue have been gotten and processed.
 |
 |      The count of unfinished tasks goes up whenever an item is added to
 |      queue. The count goes down whenever a consumer thread calls task_do
 |      to indicate the item was retrieved and all work on it is complete.
 |
 |      When the count of unfinished tasks drops to zero, join() unblocks.
 |
 |  put(self, item, block=True, timeout=None)
 |      Put an item into the queue.
 |
 |      If optional args 'block' is true and 'timeout' is None (the default
 |      block if necessary until a free slot is available. If 'timeout' is
 |      a positive number, it blocks at most 'timeout' seconds and raises
 |      the Full exception if no free slot was available within that time.
 |      Otherwise ('block' is false), put an item on the queue if a free sl
 |      is immediately available, else raise the Full exception ('timeout'
 |      is ignored in that case).
 |
 |  put_nowait(self, item)
 |      Put an item into the queue without blocking.
 |
 |      Only enqueue the item if a free slot is immediately available.
 |      Otherwise raise the Full exception.
 |
 |  qsize(self)
```

```
 |      Return the approximate size of the queue (not reliable!).
 |
 |  task_done(self)
 |      Indicate that a formerly enqueued task is complete.
 |
 |      Used by Queue consumer threads.  For each get() used to fetch a tas
 |      a subsequent call to task_done() tells the queue that the processin
 |      on the task is complete.
 |
 |      If a join() is currently blocking, it will resume when all items
 |      have been processed (meaning that a task_done() call was received
 |      for every item that had been put() into the queue).
 |
 |      Raises a ValueError if called more times than there were items
 |      placed in the queue.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Queue:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

>>>
```

Using heapq[edit]

Python has the heapq module in its standard library.

Although one can use the heappush method to add items individually to a hea

```
>>> from heapq import heappush, heappop, heapify
>>> items = [(3, "Clear drains"), (4, "Feed cat"), (5, "Make tea"), (1, "So
>>> heapify(items)
>>> while items:
  print(heappop(items))


(1, 'Solve RC tasks')
(2, 'Tax return')
(3, 'Clear drains')
(4, 'Feed cat')
(5, 'Make tea')
>>>
```

Help text for module heapq

```
>>> help('heapq')
Help on module heapq:

NAME
    heapq - Heap queue algorithm (a.k.a. priority queue).

DESCRIPTION
    Heaps are arrays for which a[k] <= a[2*k+1] and a[k] <= a[2*k+2] for
    all k, counting elements from 0.  For the sake of comparison,
    non-existing elements are considered to be infinite.  The interesting
    property of a heap is that a[0] is always its smallest element.

    Usage:

    heap = []            # creates an empty heap
    heappush(heap, item) # pushes a new item on the heap
    item = heappop(heap) # pops the smallest item from the heap
    item = heap[0]       # smallest item on the heap without popping it
    heapify(x)           # transforms list into a heap, in-place, in linear
    item = heapreplace(heap, item) # pops and returns smallest item, and ad
                                   # new item; the heap size is unchanged

    Our API differs from textbook heap algorithms as follows:

    - We use 0-based indexing.  This makes the relationship between the
      index for a node and the indexes for its children slightly less
      obvious, but is more suitable since Python uses 0-based indexing.

    - Our heappop() method returns the smallest item, not the largest.

    These two make it possible to view the heap as a regular Python list
    without surprises: heap[0] is the smallest item, and heap.sort()
    maintains the heap invariant!

FUNCTIONS
    heapify(...)
        Transform list into a heap, in-place, in O(len(heap)) time.

    heappop(...)
        Pop the smallest item off the heap, maintaining the heap invariant.

    heappush(...)
        Push item onto heap, maintaining the heap invariant.

    heappushpop(...)
        Push item on the heap, then pop and return the smallest item
        from the heap. The combined action runs more efficiently than
        heappush() followed by a separate call to heappop().

    heapreplace(...)
        Pop and return the current smallest value, and add the new item.
```

```
        This is more efficient than heappop() followed by heappush(), and c
        more appropriate when using a fixed-size heap.  Note that the value
        returned may be larger than item!  That constrains reasonable uses o
        this routine unless written as part of a conditional replacement:

            if item > heap[0]:
                item = heapreplace(heap, item)

    merge(*iterables)
        Merge multiple sorted inputs into a single sorted output.

        Similar to sorted(itertools.chain(*iterables)) but returns a genera
        does not pull the data into memory all at once, and assumes that ea
        the input streams is already sorted (smallest to largest).

        >>> list(merge([1,3,5,7], [0,2,4,8], [5,10,15,20], [], [25]))
        [0, 1, 2, 3, 4, 5, 5, 7, 8, 10, 15, 20, 25]

    nlargest(n, iterable, key=None)
        Find the n largest elements in a dataset.

        Equivalent to:  sorted(iterable, key=key, reverse=True)[:n]

    nsmallest(n, iterable, key=None)
        Find the n smallest elements in a dataset.

        Equivalent to:  sorted(iterable, key=key)[:n]

DATA
    __about__ = 'Heap queues\n\n[explanation by François Pinard]\n\nH... t.
    __all__ = ['heappush', 'heappop', 'heapify', 'heapreplace', 'merge', '.

FILE
    c:\python32\lib\heapq.py


>>>
```

R[edit]

Python: Probabilistic_Choice[edit]

Two different algorithms are coded.

```
import random, bisect

def probchoice(items, probs):
```

```
  '''\
  Splits the interval 0.0-1.0 in proportion to probs
  then finds where each random.random() choice lies
  '''

  prob_accumulator = 0
  accumulator = []
  for p in probs:
    prob_accumulator += p
    accumulator.append(prob_accumulator)

  while True:
    r = random.random()
    yield items[bisect.bisect(accumulator, r)]

def probchoice2(items, probs, bincount=10000):
  '''\
  Puts items in bins in proportion to probs
  then uses random.choice() to select items.

  Larger bincount for more memory use but
  higher accuracy (on avarage).
  '''

  bins = []
  for item,prob in zip(items, probs):
    bins += [item]*int(bincount*prob)
  while True:
    yield random.choice(bins)


def tester(func=probchoice, items='good bad ugly'.split(),
                    probs=[0.5, 0.3, 0.2],
                    trials = 100000
                    ):
  def problist2string(probs):
    '''\
    Turns a list of probabilities into a string
    Also rounds FP values
    '''
    return ",".join('%8.6f' % (p,) for p in probs)

  from collections import defaultdict

  counter = defaultdict(int)
  it = func(items, probs)
  for dummy in xrange(trials):
    counter[it.next()] += 1
  print "\n##\n## %s\n##" % func.func_name.upper()
  print "Trials:              ", trials
  print "Items:               ", ' '.join(items)
  print "Target probability:  ", problist2string(probs)
  print "Attained probability:", problist2string(
    counter[x]/float(trials) for x in items)

if __name__ == '__main__':
```

```
  items = 'aleph beth gimel daleth he waw zayin heth'.split()
  probs = [1/(float(n)+5) for n in range(len(items))]
  probs[-1] = 1-sum(probs[:-1])
  tester(probchoice, items, probs, 1000000)
  tester(probchoice2, items, probs, 1000000)
```

Sample output:

```
##
## PROBCHOICE
##
Trials:                   1000000
Items:                    aleph beth gimel daleth he waw zayin heth
Target probability:   0.200000,0.166667,0.142857,0.125000,0.111111,0.100000
Attained probability: 0.200050,0.167109,0.143364,0.124690,0.111237,0.099661

##
## PROBCHOICE2
```

Python: Probabilistic_choice[edit]

Two different algorithms are coded.

```
import random, bisect

def probchoice(items, probs):
  '''\
  Splits the interval 0.0-1.0 in proportion to probs
  then finds where each random.random() choice lies
  '''

  prob_accumulator = 0
  accumulator = []
  for p in probs:
    prob_accumulator += p
    accumulator.append(prob_accumulator)

  while True:
    r = random.random()
    yield items[bisect.bisect(accumulator, r)]

def probchoice2(items, probs, bincount=10000):
  '''\
  Puts items in bins in proportion to probs
  then uses random.choice() to select items.

  Larger bincount for more memory use but
  higher accuracy (on avarage).
```

```
  '''

  bins = []
  for item,prob in zip(items, probs):
    bins += [item]*int(bincount*prob)
  while True:
    yield random.choice(bins)


def tester(func=probchoice, items='good bad ugly'.split(),
                    probs=[0.5, 0.3, 0.2],
                    trials = 100000
                    ):
  def problist2string(probs):
    '''\
    Turns a list of probabilities into a string
    Also rounds FP values
    '''
    return ",".join('%8.6f' % (p,) for p in probs)

  from collections import defaultdict

  counter = defaultdict(int)
  it = func(items, probs)
  for dummy in xrange(trials):
    counter[it.next()] += 1
  print "\n##\n## %s\n##" % func.func_name.upper()
  print "Trials:              ", trials
  print "Items:               ", ' '.join(items)
  print "Target probability:  ", problist2string(probs)
  print "Attained probability:", problist2string(
    counter[x]/float(trials) for x in items)

if __name__ == '__main__':
  items = 'aleph beth gimel daleth he waw zayin heth'.split()
  probs = [1/(float(n)+5) for n in range(len(items))]
  probs[-1] = 1-sum(probs[:-1])
  tester(probchoice, items, probs, 1000000)
  tester(probchoice2, items, probs, 1000000)
```

Sample output:

```
##
## PROBCHOICE
##
Trials:               1000000
Items:                aleph beth gimel daleth he waw zayin heth
Target probability:   0.200000,0.166667,0.142857,0.125000,0.111111,0.100000
Attained probability: 0.200050,0.167109,0.143364,0.124690,0.111237,0.099661

##
## PROBCHOICE2
```

```
##
Trials:                  1000000
Items:                   aleph beth gimel daleth he waw zayin heth
Target probability:   0.200000,0.166667,0.142857,0.125000,0.111111,0.100000
Attained probability: 0.199720,0.166424,0.142474,0.124561,0.111511,0.100313
```

Python: Problem_of_Apollonius[edit]

Translation of: Java
. Although a Circle class is defined, the solveApollonius function is defin

```
from collections import namedtuple
import math

Circle = namedtuple('Circle', 'x, y, r')

def solveApollonius(c1, c2, c3, s1, s2, s3):
    '''
    >>> solveApollonius((0, 0, 1), (4, 0, 1), (2, 4, 2), 1,1,1)
    Circle(x=2.0, y=2.1, r=3.9)
    >>> solveApollonius((0, 0, 1), (4, 0, 1), (2, 4, 2), -1,-1,-1)
    Circle(x=2.0, y=0.8333333333333333, r=1.1666666666666667)
    '''
    x1, y1, r1 = c1
    x2, y2, r2 = c2
    x3, y3, r3 = c3

    v11 = 2*x2 - 2*x1
    v12 = 2*y2 - 2*y1
    v13 = x1*x1 - x2*x2 + y1*y1 - y2*y2 - r1*r1 + r2*r2
    v14 = 2*s2*r2 - 2*s1*r1

    v21 = 2*x3 - 2*x2
    v22 = 2*y3 - 2*y2
    v23 = x2*x2 - x3*x3 + y2*y2 - y3*y3 - r2*r2 + r3*r3
    v24 = 2*s3*r3 - 2*s2*r2

    w12 = v12/v11
    w13 = v13/v11
    w14 = v14/v11

    w22 = v22/v21-w12
    w23 = v23/v21-w13
    w24 = v24/v21-w14

    P = -w23/w22
    Q = w24/w22
    M = -w12*P-w13
    N = w14 - w12*Q

    a = N*N + Q*Q - 1
```

```
    b = 2*M*N - 2*N*x1 + 2*P*Q - 2*Q*y1 + 2*s1*r1
    c = x1*x1 + M*M - 2*M*x1 + P*P + y1*y1 - 2*P*y1 - r1*r1

    # Find a root of a quadratic equation. This requires the circle centers
    D = b*b-4*a*c
    rs = (-b-math.sqrt(D))/(2*a)

    xs = M+N*rs
    ys = P+Q*rs

    return Circle(xs, ys, rs)

if __name__ == '__main__':
    c1, c2, c3 = Circle(0, 0, 1), Circle(4, 0, 1), Circle(2, 4, 2)
    print(solveApollonius(c1, c2, c3, 1, 1, 1))    #Expects "Circle[x=2.00,
    print(solveApollonius(c1, c2, c3, -1, -1, -1)) #Expects "Circle[x=2.00,
```

Sample Output

```
Circle(x=2.0, y=2.1, r=3.9)
Circle(x=2.0, y=0.8333333333333333, r=1.1666666666666667)
```

Racket[edit]

Python: Program_name[edit]

Python has at least two ways to get the script name: the traditional ARGV a

```
#!/usr/bin/env python

import sys

def main():
        program = sys.argv[0]
        print "Program: %s" % program

if __name__=="__main__":
        main()
```

```
#!/usr/bin/env python

import inspect

def main():
        program = inspect.getfile(inspect.currentframe())
        print "Program: %s" % program

if __name__=="__main__":
        main()
```

Python: Program_termination[edit]

Polite

```
import sys
if problem:
    sys.exit(1)
```

The atexit module allows you to register functions to be run when the progr

As soon as possible

Python: Pyramid_of_numbers[edit]

Works with: Python version 2.4+

```
# Pyramid solver
#            [151]
#         [   ] [   ]
#      [ 40] [   ] [   ]
#   [   ] [   ] [   ] [   ]
#[ X ] [ 11] [ Y ] [ 4 ] [ Z ]
#  X -Y + Z = 0

def combine( snl, snr ):

        cl = {}
        if isinstance(snl, int):
```

```
		cl['1'] = snl
	elif isinstance(snl, string):
		cl[snl] = 1
	else:
		cl.update( snl)

	if isinstance(snr, int):
		n = cl.get('1', 0)
		cl['1'] = n + snr
	elif isinstance(snr, string):
		n = cl.get(snr, 0)
		cl[snr] = n + 1
	else:
		for k,v in snr.items():
			n = cl.get(k, 0)
			cl[k] = n+v
	return cl


def constrain(nsum, vn ):
	nn = {}
	nn.update(vn)
	n = nn.get('1', 0)
	nn['1'] = n - nsum
	return nn

def makeMatrix( constraints ):
	vmap = set()
	for c in constraints:
		vmap.update( c.keys())
	vmap.remove('1')
	nvars = len(vmap)
	vmap = sorted(vmap)		# sort here so output is in sorted 
	mtx = []
	for c in constraints:
		row = []
		for vv in vmap:
			row.append(float(c.get(vv, 0)))
		row.append(-float(c.get('1',0)))
		mtx.append(row)

	if len(constraints) == nvars:
		print 'System appears solvable'
	elif len(constraints) < nvars:
		print 'System is not solvable - needs more constraints.'
	return mtx, vmap


def SolvePyramid( vl, cnstr ):

	vl.reverse()
	constraints = [cnstr]
	lvls = len(vl)
	for lvln in range(1,lvls):
		lvd = vl[lvln]
		for k in range(lvls - lvln):
```

```
                sn = lvd[k]
                ll = vl[lvln-1]
                vn = combine(ll[k], ll[k+1])
                if sn is None:
                        lvd[k] = vn
                else:
                        constraints.append(constrain( sn, vn ))

        print 'Constraint Equations:'
        for cstr in constraints:
                fset = ('%d*%s'%(v,k) for k,v in cstr.items() )
                print ' + '.join(fset), ' = 0'

        mtx,vmap = makeMatrix(constraints)

        MtxSolve(mtx)

        d = len(vmap)
        for j in range(d):
                print vmap[j],'=', mtx[j][d]


def MtxSolve(mtx):
        # Simple Matrix solver...

        mDim = len(mtx)                 # dimension---
        for j in range(mDim):
                rw0= mtx[j]
                f = 1.0/rw0[j]
                for k in range(j, mDim+1):
                        rw0[k] *= f

                for l in range(1+j,mDim):
                        rwl = mtx[l]
                        f = -rwl[j]
                        for k in range(j, mDim+1):
                                rwl[k] += f * rw0[k]

        # backsolve part ---
        for j1 in range(1,mDim):
                j = mDim - j1
                rw0= mtx[j]
                for l in range(0, j):
                        rwl = mtx[l]
                        f = -rwl[j]
                        rwl[j]    += f * rw0[j]
                        rwl[mDim] += f * rw0[mDim]

        return mtx


p = [ [151], [None,None], [40,None,None], [None,None,None,None], ['X', 11,
addlConstraint = { 'X':1, 'Y':-1, 'Z':1, '1':0 }
SolvePyramid( p, addlConstraint)
```

```
Output:
```

```
Constraint Equations:
-1*Y + 1*X + 0*1 + 1*Z  = 0
-18*1 + 1*X + 1*Y  = 0
-73*1 + 5*Y + 1*Z  = 0
System appears solvable
X = 5.0
Y = 13.0
Z = 8.0
```

The Pyramid solver is not restricted to solving for 3 variables, or just th

Alternative solution using the csp module (based on code by Gustavo Niemeye
http://www.fantascienza.net/leonardo/so/csp.zip

```
from csp import Problem

p = Problem()
pvars = "R2 R3 R5 R6 R7 R8 R9 R10 X Y Z".split()
# 0-151 is the possible finite range of the variables
p.addvars(pvars, xrange(152))
p.addrule("R7 == X + 11")
p.addrule("R8 == Y + 11")
p.addrule("R9 == Y + 4")
p.addrule("R10 == Z + 4")
p.addrule("R7 + R8 == 40")
p.addrule("R5 == R8 + R9")
p.addrule("R6 == R9 + R10")
p.addrule("R2 == 40 + R5")
p.addrule("R3 == R5 + R6")
p.addrule("R2 + R3 == 151")
p.addrule("Y == X + Z")
for sol in p.xsolutions():
    print [sol[k] for k in "XYZ"]
```

Output:

```
[5, 13, 8]
```

Python: Pythagorean_triples[edit]

Two methods, the second of which is much faster

```
from fractions import gcd


def pt1(maxperimeter=100):
    '''
# Naive method
    '''
    trips = []
    for a in range(1, maxperimeter):
        aa = a*a
        for b in range(a, maxperimeter-a+1):
            bb = b*b
            for c in range(b, maxperimeter-b-a+1):
                cc = c*c
                if a+b+c > maxperimeter or cc > aa + bb: break
                if aa + bb == cc:
                    trips.append((a,b,c, gcd(a, b) == 1))
    return trips

def pytrip(trip=(3,4,5),perim=100, prim=1):
    a0, b0, c0 = a, b, c = sorted(trip)
    t, firstprim = set(), prim>0
    while a + b + c <= perim:
        t.add((a, b, c, firstprim>0))
        a, b, c, firstprim = a+a0, b+b0, c+c0, False
    #
    t2 = set()
    for a, b, c, firstprim in t:
        a2, a5, b2, b5, c2, c3, c7 = a*2, a*5, b*2, b*5, c*2, c*3, c*7
        if  a5 - b5 + c7 <= perim:
            t2 |= pytrip(( a - b2 + c2,  a2 - b + c2,  a2 - b2 + c3), perim
        if  a5 + b5 + c7 <= perim:
            t2 |= pytrip(( a + b2 + c2,  a2 + b + c2,  a2 + b2 + c3), perim
        if -a5 + b5 + c7 <= perim:
            t2 |= pytrip((-a + b2 + c2, -a2 + b + c2, -a2 + b2 + c3), perim
    return t | t2

def pt2(maxperimeter=100):
    '''
# Parent/child relationship method:
# http://en.wikipedia.org/wiki/Formulas_for_generating_Pythagorean_triples#
    '''
    trips = pytrip((3,4,5), maxperimeter, 1)
    return trips

def printit(maxperimeter=100, pt=pt1):
    trips = pt(maxperimeter)
    print("  Up to a perimeter of %i there are %i triples, of which %i are
          % (maxperimeter,
             len(trips),
```

```
            len([prim for a,b,c,prim in trips if prim])))

for algo, mn, mx in ((pt1, 250, 2500), (pt2, 500, 20000)):
    print(algo.__doc__)
    for maxperimeter in range(mn, mx+1, mn):
        printit(maxperimeter, algo)
```

Output

```
# Naive method

  Up to a perimeter of 250 there are 56 triples, of which 18 are primitive
  Up to a perimeter of 500 there are 137 triples, of which 35 are primitive
  Up to a perimeter of 750 there are 227 triples, of which 52 are primitive
  Up to a perimeter of 1000 there are 325 triples, of which 70 are primitiv
  Up to a perimeter of 1250 there are 425 triples, of which 88 are primitiv
  Up to a perimeter of 1500 there are 527 triples, of which 104 are primiti
  Up to a perimeter of 1750 there are 637 triples, of which 123 are primiti
  Up to a perimeter of 2000 there are 744 triples, of which 140 are primiti
  Up to a perimeter of 2250 there are 858 triples, of which 156 are primiti
  Up to a perimeter of 2500 there are 969 triples, of which 175 are primiti

# Parent/child relationship method:
# http://en.wikipedia.org/wiki/Formulas_for_generating_Pythagorean_triples#
```

Python: QR_decomposition[edit]

Library: numpy

Numpy has a qr function but here is a reimplementation to show construction

```
#!/usr/bin/env python3

import numpy as np

def qr(A):
    m, n = A.shape
    Q = np.eye(m)
    for i in range(n - (m == n)):
        H = np.eye(m)
        H[i:, i:] = make_householder(A[i:, i])
        Q = np.dot(Q, H)
        A = np.dot(H, A)
    return Q, A

def make_householder(a):
```

```
    v = a / (a[0] + np.copysign(np.linalg.norm(a), a[0]))
    v[0] = 1
    H = np.eye(a.shape[0])
    H -= (2 / np.dot(v, v)) * np.dot(v[:, None], v[None, :])
    return H

# task 1: show qr decomp of wp example
a = np.array(((
    (12, -51,   4),
    ( 6, 167, -68),
    (-4,  24, -41),
)))

q, r = qr(a)
print('q:\n', q.round(6))
print('r:\n', r.round(6))

# task 2: use qr decomp for polynomial regression example
def polyfit(x, y, n):
    return lsqr(x[:, None]**np.arange(n + 1), y.T)

def lsqr(a, b):
    q, r = qr(a)
    _, n = r.shape
    return np.linalg.solve(r[:n, :], np.dot(q.T, b)[:n])

x = np.array((0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
y = np.array((1, 6, 17, 34, 57, 86, 121, 162, 209, 262, 321))

print('\npolyfit:\n', polyfit(x, y, 2))
```

Output:

```
q:
 [[-0.857143  0.394286  0.331429]
 [-0.428571 -0.902857 -0.034286]
 [ 0.285714 -0.171429  0.942857]]
r:
 [[ -14.   -21.    14.]
 [   0. -175.    70.]
 [   0.    0.   -35.]]

polyfit:
 [ 1.  2.  3.]
```

Python: Quadratic_Equation[edit]

Library: numpy

This solution compares the naïve method with three "better" methods.

```
#!/usr/bin/env python3

import math
import cmath
import numpy

def quad_discriminating_roots(a,b,c, entier = 1e-5):
    """For reference, the naive algorithm which shows complete loss of
    precision on the quadratic in question.  (This function also returns a
    characterization of the roots.)"""
    discriminant = b*b - 4*a*c
    a,b,c,d =complex(a), complex(b), complex(c), complex(discriminant)
    root1 = (-b + cmath.sqrt(d))/2./a
    root2 = (-b - cmath.sqrt(d))/2./a
    if abs(discriminant) < entier:
        return "real and equal", abs(root1), abs(root1)
    if discriminant > 0:
        return "real", root1.real, root2.real
    return "complex", root1, root2

def middlebrook(a, b, c):
    try:
        q = math.sqrt(a*c)/b
        f = .5+ math.sqrt(1-4*q*q)/2
    except ValueError:
        q = cmath.sqrt(a*c)/b
        f = .5+ cmath.sqrt(1-4*q*q)/2
    return (-b/a)*f, -c/(b*f)

def whatevery(a, b, c):
    try:
        d = math.sqrt(b*b-4*a*c)
    except ValueError:
        d = cmath.sqrt(b*b-4*a*c)
    if b > 0:
        return div(2*c, (-b-d)), div((-b-d), 2*a)
    else:
        return div((-b+d), 2*a), div(2*c, (-b+d))

def div(n, d):
    """Divide, with a useful interpretation of division by zero."""
    try:
        return n/d
    except ZeroDivisionError:
        if n:
            return n*float('inf')
        return float('nan')

testcases = [
    (3, 4, 4/3),    # real, equal
```

```
    (3, 2, -1),     # real, unequal
    (3, 2, 1),      # complex
    (1, -1e9, 1),   # ill-conditioned "quadratic in question" required by t
    (1, -1e100, 1),
    (1, -1e200, 1),
    (1, -1e300, 1),
]

print('Naive:')
for c in testcases:
    print("{} {:.5} {:.5}".format(*quad_discriminating_roots(*c)))

print('\nMiddlebrook:')
for c in testcases:
    print(("{:.5} "*2).format(*middlebrook(*c)))

print('\nWhat Every...')
for c in testcases:
    print(("{:.5} "*2).format(*whatevery(*c)))

print('\nNumpy:')
for c in testcases:
    print(("{:.5} "*2).format(*numpy.roots(c)))
```

Output:

```
Naive:
real and equal 0.66667 0.66667
real 0.33333 -1.0
complex (-0.33333+0.4714j) (-0.33333-0.4714j)
real 1e+09 0.0
real 1e+100 0.0
real nan nan
real nan nan

Middlebrook:
-0.66667 -0.66667
(-1+0j) (0.33333+0j)
(-0.33333-0.4714j) (-0.33333+0.4714j)
1e+09 1e-09
1e+100 1e-100
1e+200 1e-200
1e+300 1e-300

What Every...
-0.66667 -0.66667
0.33333 -1.0
(-0.33333+0.4714j) (-0.33333-0.4714j)
1e+09 1e-09
1e+100 1e-100
inf 0.0
```

```
inf 0.0

Numpy:
-0.66667 -0.66667
-1.0 0.33333
(-0.33333+0.4714j) (-0.33333-0.4714j)
1e+09 1e-09
1e+100 1e-100
1e+200 1e-200
1e+300 0.0
```

Python: Quaternion_type[edit]

This example extends Pythons namedtuples to add extra functionality.

```
from collections import namedtuple
import math

class Q(namedtuple('Quaternion', 'real, i, j, k')):
    'Quaternion type: Q(real=0.0, i=0.0, j=0.0, k=0.0)'

    __slots__ = ()

    def __new__(_cls, real=0.0, i=0.0, j=0.0, k=0.0):
        'Defaults all parts of quaternion to zero'
        return super().__new__(_cls, float(real), float(i), float(j), float

    def conjugate(self):
        return Q(self.real, -self.i, -self.j, -self.k)

    def _norm2(self):
        return sum( x*x for x in self)

    def norm(self):
        return math.sqrt(self._norm2())

    def reciprocal(self):
        n2 = self._norm2()
        return Q(*(x / n2 for x in self.conjugate()))

    def __str__(self):
        'Shorter form of Quaternion as string'
        return 'Q(%g, %g, %g, %g)' % self

    def __neg__(self):
        return Q(-self.real, -self.i, -self.j, -self.k)

    def __add__(self, other):
        if type(other) == Q:
```

```
            return Q( *(s+o for s,o in zip(self, other)) )
        try:
            f = float(other)
        except:
            return NotImplemented
        return Q(self.real + f, self.i, self.j, self.k)

    def __radd__(self, other):
        return Q.__add__(self, other)

    def __mul__(self, other):
        if type(other) == Q:
            a1,b1,c1,d1 = self
            a2,b2,c2,d2 = other
            return Q(
                 a1*a2 - b1*b2 - c1*c2 - d1*d2,
                 a1*b2 + b1*a2 + c1*d2 - d1*c2,
                 a1*c2 - b1*d2 + c1*a2 + d1*b2,
                 a1*d2 + b1*c2 - c1*b2 + d1*a2 )
        try:
            f = float(other)
        except:
            return NotImplemented
        return Q(self.real * f, self.i * f, self.j * f, self.k * f)

    def __rmul__(self, other):
        return Q.__mul__(self, other)

    def __truediv__(self, other):
        if type(other) == Q:
            return self.__mul__(other.reciprocal())
        try:
            f = float(other)
        except:
            return NotImplemented
        return Q(self.real / f, self.i / f, self.j / f, self.k / f)

    def __rtruediv__(self, other):
        return other * self.reciprocal()

    __div__, __rdiv__ = __truediv__, __rtruediv__

Quaternion = Q

q  = Q(1, 2, 3, 4)
q1 = Q(2, 3, 4, 5)
q2 = Q(3, 4, 5, 6)
r  = 7
```

Continued shell session

Run the above with the -i flag to python on the command line, or run with i

```
>>> q
Quaternion(real=1.0, i=2.0, j=3.0, k=4.0)
>>> q1
Quaternion(real=2.0, i=3.0, j=4.0, k=5.0)
>>> q2
Quaternion(real=3.0, i=4.0, j=5.0, k=6.0)
>>> r
7
>>> q.norm()
5.477225575051661
>>> q1.norm()
7.3484692283495345
>>> q2.norm()
9.273618495495704
>>> -q
Quaternion(real=-1.0, i=-2.0, j=-3.0, k=-4.0)
>>> q.conjugate()
Quaternion(real=1.0, i=-2.0, j=-3.0, k=-4.0)
>>> r + q
Quaternion(real=8.0, i=2.0, j=3.0, k=4.0)
>>> q + r
Quaternion(real=8.0, i=2.0, j=3.0, k=4.0)
>>> q1 + q2
Quaternion(real=5.0, i=7.0, j=9.0, k=11.0)
>>> q2 + q1
Quaternion(real=5.0, i=7.0, j=9.0, k=11.0)
>>> q * r
Quaternion(real=7.0, i=14.0, j=21.0, k=28.0)
>>> r * q
Quaternion(real=7.0, i=14.0, j=21.0, k=28.0)
>>> q1 * q2
Quaternion(real=-56.0, i=16.0, j=24.0, k=26.0)
>>> q2 * q1
Quaternion(real=-56.0, i=18.0, j=20.0, k=28.0)
>>> assert q1 * q2 != q2 * q1
>>>
>>> i, j, k = Q(0,1,0,0), Q(0,0,1,0), Q(0,0,0,1)
>>> i*i
Quaternion(real=-1.0, i=0.0, j=0.0, k=0.0)
>>> j*j
Quaternion(real=-1.0, i=0.0, j=0.0, k=0.0)
>>> k*k
Quaternion(real=-1.0, i=0.0, j=0.0, k=0.0)
>>> i*j*k
Quaternion(real=-1.0, i=0.0, j=0.0, k=0.0)
>>> q1 / q2
Quaternion(real=0.7906976744186047, i=0.023255813953488358, j=-2.775557561
>>> q1 / q2 * q2
Quaternion(real=2.0000000000000004, i=3.0000000000000004, j=4.00000000000000
>>> q2 * q1 / q2
Quaternion(real=2.0, i=3.465116279069768, j=3.906976744186047, k=4.7674418
>>> q1.reciprocal() * q1
Quaternion(real=0.9999999999999999, i=0.0, j=0.0, k=0.0)
>>> q1 * q1.reciprocal()
Quaternion(real=0.9999999999999999, i=0.0, j=0.0, k=0.0)
>>>
```

R[edit]

Python: Query_Performance[edit]

Given *function* and *arguments* return a time (in microseconds) it takes to ma

Note: There is an overhead in executing a function that does nothing.

```
import sys, timeit
def usec(function, arguments):
    modname, funcname = __name__, function.__name__
    timer = timeit.Timer(stmt='%(funcname)s(*args)' % vars(),
                         setup='from %(modname)s import %(funcname)s; args=
    try:
        t, N = 0, 1
        while t < 0.2:
            t = min(timer.repeat(repeat=3, number=N))
            N *= 10
        microseconds = round(10000000 * t / N, 1) # per loop
        return microseconds
    except:
        timer.print_exc(file=sys.stderr)
        raise

 def nothing(): pass
 def identity(x): return x
```

Example[edit]

```
>>> print usec(nothing, [])
1.7
>>> print usec(identity, [1])
2.2
```

Python: Queue.1[edit]

A python list can be used as a simple FIFO by simply using only it's *.appen*

To encapsulate this behavior into a class and provide the task's specific A

```
   class FIFO(object):
       def __init__(self, *args):
           self.contents = list(args)
       def __call__(self):
           return self.pop()
       def __len__(self):
           return len(self.contents)
       def pop(self):
           return self.contents.pop(0)
       def push(self, item):
           self.contents.append(item)
       def extend(self,*itemlist):
           self.contents += itemlist
       def empty(self):
           return bool(self.contents)
       def __iter__(self):
           return self
       def next(self):
           if self.empty():
               raise StopIteration
           return self.pop()

if __name__ == "__main__":
    # Sample usage:
    f = FIFO()
    f.push(3)
    f.push(2)
    f.push(1)
    while not f.empty():
        print f.pop(),
    # >>> 3 2 1
    # Another simple example gives the same results:
    f = FIFO(3,2,1)
    while not f.empty():
        print f(),
    # Another using the default "truth" value of the object
    # (implicitly calls on the length() of the object after
    # checking for a __nonzero__ method
    f = FIFO(3,2,1)
    while f:
        print f(),
    # Yet another, using more Pythonic iteration:
    f = FIFO(3,2,1)
    for i in f:
        print i,
```

This example does add to a couple of features which are easy in Python and

These additional methods could be omitted and some could have been dispatch

That sort of wrapper looks like:

```
class FIFO:  ## NOT a new-style class, must not derive from "object"
   def __init__(self,*args):
       self.contents = list(args)
   def __call__(self):
       return self.pop()
   def empty(self):
       return bool(self.contents)
   def pop(self):
       return self.contents.pop(0)
   def __getattr__(self, attr):
       return getattr(self.contents,attr)
   def next(self):
       if not self:
           raise StopIteration
       return self.pop()
```

As noted in the contents this must NOT be a new-style class, it must NOT bu

Works with: Python version 2.4+

Python 2.4 and later includes a deque class, supporting thread-safe, memory

```
from collections import deque
fifo = deque()
fifo. appendleft(value) # push
value = fifo.pop()
not fifo # empty
fifo.pop() # raises IndexError when empty
```

R[edit]

Python: Quickselect_algorithm[edit]

A direct implementation of the Wikipedia pseudo-code, using a random initia

```
import random
```

```
def partition(vector, left, right, pivotIndex):
    pivotValue = vector[pivotIndex]
    vector[pivotIndex], vector[right] = vector[right], vector[pivotIndex]  #
    storeIndex = left
    for i in range(left, right):
        if vector[i] < pivotValue:
            vector[storeIndex], vector[i] = vector[i], vector[storeIndex]
            storeIndex += 1
    vector[right], vector[storeIndex] = vector[storeIndex], vector[right]  #
    return storeIndex

def _select(vector, left, right, k):
    "Returns the k-th smallest, (k >= 0), element of vector within vector[le
    while True:
        pivotIndex = random.randint(left, right)     # select pivotIndex bet
        pivotNewIndex = partition(vector, left, right, pivotIndex)
        pivotDist = pivotNewIndex - left
        if pivotDist == k:
            return vector[pivotNewIndex]
        elif k < pivotDist:
            right = pivotNewIndex - 1
        else:
            k -= pivotDist + 1
            left = pivotNewIndex + 1

def select(vector, k, left=None, right=None):
    """\
    Returns the k-th smallest, (k >= 0), element of vector within vector[le
    left, right default to (0, len(vector) - 1) if omitted
    """
    if left is None:
        left = 0
    lv1 = len(vector) - 1
    if right is None:
        right = lv1
    assert vector and k >= 0, "Either null vector or k < 0 "
    assert 0 <= left <= lv1, "left is out of range"
    assert left <= right <= lv1, "right is out of range"
    return _select(vector, left, right, k)

if __name__ == '__main__':
    v = [9, 8, 7, 6, 5, 0, 1, 2, 3, 4]
    print([select(v, i) for i in range(10)])
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Python: Quicksort[edit]

```
def quickSort(arr):
    less = []
    pivotList = []
    more = []
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        for i in arr:
            if i < pivot:
                less.append(i)
            elif i > pivot:
                more.append(i)
            else:
                pivotList.append(i)
        less = quickSort(less)
        more = quickSort(more)
        return less + pivotList + more

a = [4, 65, 2, -31, 0, 99, 83, 782, 1]
a = quickSort(a)
```

In a Haskell fashion --

```
def qsort(L):
    return (qsort([y for y in L[1:] if y <  L[0]]) +
            L[:1] +
            qsort([y for y in L[1:] if y >= L[0]])) if len(L) > 1 else L
```

More readable, but still using list comprehensions:

Python: Quine[edit]

Works with: Python version 2.x and 3.x

Python's %r format conversion uses the repr() function to return a string c

```
x = 'x = %r\nprint(x %% x)'
print(x % x)
```

Works with: Python version 3.x and 2.6+

With the new str.format:

```
x = 'x = {!r};print(x.format(x))';print(x.format(x))
```

Works with: Python version 2.x and 3.x

After creating the file "Quine.py" with the following source, running the program will spit the code back out on a terminal window:

```
import sys; sys.stdout.write(open(sys.argv[0]).read())
```

Note: actually an empty file could be treated as python quine too.

Works with: Python version 2.x and 3.x

```
import sys,inspect;sys.stdout.write(inspect.getsource(inspect.currentframe(
```

Due to Leon Naley (name guessed) from devshed python forum

Python: Quotes[edit]

Python makes no distinction between single characters and strings.
One can use single or double quotes.

```
'c' == "c" # character
'text' == "text"
' " '
" ' "
'\x20' == ' '
u'unicode string'
u'\u05d0' # unicode literal
```

As shown in the last examples, Unicode strings are single or double quoted with a "u" or "U" prepended thereto.

Verbatim (a.k.a. "raw") strings are contained within either single or doubl
This is useful when defining regular expressions as it avoids the need to u

```
r'\x20' == '\\x20'
```

The Unicode and raw string modifiers can be combined to prefix a raw Unicod

Here-strings are denoted with triple quotes.

```
''' single triple quote '''
""" double triple quote """
```

The "u" and "r" prefixes can also be used with triple quoted strings.

Triple quoted strings can contain any mixture of double and single quotes a
They are terminated by unescaped triple quotes of the same type that initia
They are generally used for "doc strings" and other multi-line string expre

Python: RCSNUSP.1[edit]

Translation of: Go

```
#!/usr/bin/env python3

HW = r'''
/++++!/===========?\>++.>+.+++++++..+++\
\+++\ | /+>+++++++>/ /++++++++++<<.++>./
$+++/ | \+++++++++>\ \+++++.>.+++.-----\
      \==-<<<<+>+++/ /=.>.+>.--------.-/'''

def snusp(store, code):
    ds = bytearray(store)  # data store
    dp = 0                 # data pointer
    cs = code.splitlines() # 2 dimensional code store
    ipr, ipc = 0, 0        # instruction pointers in row and column
    for r, row in enumerate(cs):
        try:
            ipc = row.index('$')
            ipr = r
            break
        except ValueError:
            pass
    rt, dn, lt, up = range(4)
```

```
    id = rt  # instruction direction.  starting direction is always rt
    def step():
        nonlocal ipr, ipc
        if id&1:
            ipr += 1 - (id&2)
        else:
            ipc += 1 - (id&2)
    while ipr >= 0 and ipr < len(cs) and ipc >= 0 and ipc < len(cs[ipr]):
        op = cs[ipr][ipc]
        if op == '>':
            dp += 1
        elif op == '<':
            dp -= 1
        elif op == '+':
            ds[dp] += 1
        elif op == '-':
            ds[dp] -= 1
        elif op == '.':
            print(chr(ds[dp]), end='')
        elif op == ',':
            ds[dp] = input()
        elif op == '/':
            id = ~id
        elif op == '\\':
            id ^= 1
        elif op == '!':
            step()
        elif op == '?':
            if not ds[dp]:
                step()
        step()

if __name__ == '__main__':
    snusp(5, HW)
```

Output:

```
Hello World!
```

Python: REPL[edit]

Start the interpreter by typing python at the command line (or select it fr

```
python
Python 2.6.1 (r261:67517, Dec  4 2008, 16:51:00) [MSC v.1500 32 bit (Intel)
```

```
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def f(string1, string2, separator):
        return separator.join([string1, '', string2])

>>> f('Rosetta', 'Code', ':')
'Rosetta::Code'
>>>
```

Python: RIPEMD-160[edit]

```
Python 3.3.0 (v3.3.0:bd8afb90ebf2, Sep 29 2012, 10:57:17) [MSC v.1600 64 bi
Type "copyright", "credits" or "license()" for more information.
>>> import hashlib
>>> h = hashlib.new('ripemd160')
>>> h.update(b"Rosetta Code")
>>> h.hexdigest()
'b3be159860842cebaa7174c8fff0aa9e50a5199f'
>>>
```

Python: RLE[edit]

```
def encode(input_string):
    count = 1
    prev = ''
    lst = []
    for character in input_string:
        if character != prev:
            if prev:
                entry = (prev,count)
                lst.append(entry)
                #print lst
            count = 1
            prev = character
        else:
            count += 1
    else:
        entry = (character,count)
        lst.append(entry)
    return lst
```

```
def decode(lst):
    q = ""
    for character, count in lst:
        q += character * count
    return q

#Method call
encode("aaaaahhhhhhmmmmmmmuiiiiiiiaaaaaa")
decode([('a', 5), ('h', 6), ('m', 7), ('u', 1), ('i', 7), ('a', 6)])
```

Functional

Python: RSA_code[edit]

This example **may be incorrect** due to a recent change in the task requirements or a lack of testing. Please verify it and remove this message. If the example does not match the requirements or does not work, replace this message with Template:incorrect or fix the code yourself.

This code will open up a simple Tkinter window which has space to type a me
blocks, separated by commas. To decrypt a message, simply press the decrypt
in the bottom box, while plaintext goes (and appears) in the topmost box. U
block of plaintext is not a single letter, for example, a, 01, encoded is 0

Note: the key given here is a toy key, it is easily broken.

Python: Radix_sort[edit]

Works with: Python version 2.6

This is the Wikipedia example code extended with an extra pass to sort nega

```
#python2.6 <
from math import log

def getDigit(num, base, digit_num):
    # pulls the selected digit
    return (num // base ** digit_num) % base

def makeBlanks(size):
    # create a list of empty lists to hold the split by digit
    return [ [] for i in range(size) ]

def split(a_list, base, digit_num):
    buckets = makeBlanks(base)
```

```
    for num in a_list:
        # append the number to the list selected by the digit
        buckets[getDigit(num, base, digit_num)].append(num)
    return buckets

# concatenate the lists back in order for the next step
def merge(a_list):
    new_list = []
    for sublist in a_list:
       new_list.extend(sublist)
    return new_list

def maxAbs(a_list):
    # largest abs value element of a list
    return max(abs(num) for num in a_list)

def split_by_sign(a_list):
    # splits values by sign - negative values go to the first bucket,
    # non-negative ones into the second
    buckets = [[], []]
    for num in a_list:
        if num < 0:
            buckets[0].append(num)
        else:
            buckets[1].append(num)
    return buckets

def radixSort(a_list, base):
    # there are as many passes as there are digits in the longest number
    passes = int(round(log(maxAbs(a_list), base)) + 1)
    new_list = list(a_list)
    for digit_num in range(passes):
        new_list = merge(split(new_list, base, digit_num))
    return merge(split_by_sign(new_list))
```

Python: Ramsey's_theorem[edit]

Works with: Python version 3.4.1

Translation of: C

```
range17 = range(17)
a = [['0'] * 17 for i in range17]
idx = [0] * 4


def find_group(mark, min_n, max_n, depth=1):
    if (depth == 4):
        prefix = "" if (mark == '1') else "un"
        print("Fail, found totally {}connected group:".format(prefix))
```

```
        for i in range(4):
            print(idx[i])
        return True

    for i in range(min_n, max_n):
        n = 0
        while (n < depth):
            if (a[idx[n]][i] != mark):
                break
            n += 1

        if (n == depth):
            idx[n] = i
            if (find_group(mark, 1, max_n, depth + 1)):
                return True

    return False


if __name__ == '__main__':
    for i in range17:
        a[i][i] = '-'
    for k in range(4):
        for i in range17:
            j = (i + pow(2, k)) % 17
            a[i][j] = a[j][i] = '1'

    # testcase breakage
    # a[2][1] = a[1][2] = '0'

    for row in a:
        print(' '.join(row))

    for i in range17:
        idx[0] = i
        if (find_group('1', i + 1, 17) or find_group('0', i + 1, 17)):
            print("no good")
            exit()

    print("all good")
```

```
Output same as C:
```

Python: Random_number_generator_(device)[edit]

```
import random
rand = random.SystemRandom()
rand.randint(1,10)
```

Python: Random_number_generator_(included)[edit]

Python uses the Mersenne twister algorithm accessed via the built-in random

Python: Random_numbers[edit]

Using random.gauss

Python: Range_expansion[edit]

```
def rangeexpand(txt):
    lst = []
    for r in txt.split(','):
        if '-' in r[1:]:
            r0, r1 = r[1:].split('-', 1)
            lst += range(int(r[0] + r0), int(r1) + 1)
        else:
            lst.append(int(r))
    return lst

print(rangeexpand('-6,-3--1,3-5,7-11,14,15,17-20'))
```

Another variant, using regular expressions to parse the ranges:

```
import re

def rangeexpand(txt):
    lst = []
    for rng in txt.split(','):
        start,end = re.match('^(-?\d+)(?:-(-?\d+))?$', rng).groups()
        if end:
            lst.extend(xrange(int(start),int(end)+1))
        else:
            lst.append(int(start))
    return lst
```

Output:

```
[-6, -3, -2, -1, 3, 4, 5, 7, 8, 9, 10, 11, 14, 15, 17, 18, 19, 20]
```

another variant, using a functional style to parse the ranges:

```
from functools import reduce
from operator import add

def rangeexpand(s):
    return reduce(add,
            map(lambda x: list(range(*map(int, x.split('-')))) if '-' in x e
```

Python: Range_extraction[edit]

```
def range_extract(lst):
    'Yield 2-tuple ranges or 1-tuple single elements from list of increasin
    lenlst = len(lst)
    i = 0
    while i< lenlst:
        low = lst[i]
        while i <lenlst-1 and lst[i]+1 == lst[i+1]: i +=1
        hi = lst[i]
        if   hi - low >= 2:
            yield (low, hi)
        elif hi - low == 1:
            yield (low,)
            yield (hi,)
        else:
            yield (low,)
        i += 1

def printr(ranges):
    print( ','.join( (('%i-%i' % r) if len(r) == 2 else '%i' % r)
                     for r in ranges ) )

if __name__ == '__main__':
    for lst in [[-8, -7, -6, -3, -2, -1, 0, 1, 3, 4, 5, 7,
                 8, 9, 10, 11, 14, 15, 17, 18, 19, 20],
                [0, 1, 2, 4, 6, 7, 8, 11, 12, 14, 15, 16, 17, 18, 19, 20, 2
                 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 35, 36, 37, 38, 39
        #print(list(range_extract(lst)))
        printr(range_extract(lst))
```

Output:

```
-8--6,-3-1,3-5,7-11,14,15,17-20
0-2,4,6-8,11,12,14-25,27-33,35-39
```

Output:

if the print**r**(...) statement is commented-out instead of the print(...) st

This shows the tuples yielded by generator function range_extract.

```
[(-8, -6), (-3, 1), (3, 5), (7, 11), (14,), (15,), (17, 20)]
[(0, 2), (4,), (6, 8), (11,), (12,), (14, 25), (27, 33), (35, 39)]
```

Qi[edit]

Python: Ranking_methods[edit]

```
def mc_rank(iterable, start=1):
    """Modified competition ranking"""
    lastresult, fifo = None, []
    for n, item in enumerate(iterable, start-1):
        if item[0] == lastresult:
            fifo += [item]
        else:
            while fifo:
                yield n, fifo.pop(0)
            lastresult, fifo = item[0], fifo + [item]
    while fifo:
        yield n+1, fifo.pop(0)


def sc_rank(iterable, start=1):
    """Standard competition ranking"""
    lastresult, lastrank = None, None
    for n, item in enumerate(iterable, start):
        if item[0] == lastresult:
            yield lastrank, item
        else:
            yield n, item
            lastresult, lastrank = item[0], n
```

```
def d_rank(iterable, start=1):
    """Dense ranking"""
    lastresult, lastrank = None, start - 1,
    for item in iterable:
        if item[0] == lastresult:
            yield lastrank, item
        else:
            lastresult, lastrank = item[0], lastrank + 1
            yield lastrank, item


def o_rank(iterable, start=1):
    """Ordinal  ranking"""
    yield from enumerate(iterable, start)


def f_rank(iterable, start=1):
    """Fractional ranking"""
    last, fifo = None, []
    for n, item in enumerate(iterable, start):
        if item[0] != last:
            if fifo:
                mean = sum(f[0] for f in fifo) / len(fifo)
                while fifo:
                    yield mean, fifo.pop(0)[1]
        last = item[0]
        fifo.append((n, item))
    if fifo:
        mean = sum(f[0] for f in fifo) / len(fifo)
        while fifo:
            yield mean, fifo.pop(0)[1]


if __name__ == '__main__':
    scores = [(44, 'Solomon'),
              (42, 'Jason'),
              (42, 'Errol'),
              (41, 'Garry'),
              (41, 'Bernard'),
              (41, 'Barry'),
              (39, 'Stephen')]

    print('\nScores to be ranked (best first):')
    for s in scores:
        print('        %2i %s' % (s ))
    for ranker in [sc_rank, mc_rank, d_rank, o_rank, f_rank]:
        print('\n%s:' % ranker.__doc__)
        for rank, score in ranker(scores):
            print('  %3g, %r' % (rank, score))
```

Output:

```
Scores to be ranked (best first):
        44 Solomon
        42 Jason
        42 Errol
        41 Garry
        41 Bernard
        41 Barry
        39 Stephen
```

Python: Rate_counter[edit]

```
import subprocess
import time

class Tlogger(object):
    def __init__(self):
        self.counts = 0
        self.tottime = 0.0
        self.laststart = 0.0
        self.lastreport = time.time()

    def logstart(self):
        self.laststart = time.time()

    def logend(self):
        self.counts +=1
        self.tottime += (time.time()-self.laststart)
        if (time.time()-self.lastreport)>5.0:   # report once every 5 secon
           self.report()

    def report(self):
        if ( self.counts > 4*self.tottime):
            print "Subtask execution rate: %f times/second"% (self.counts/se
        else:
            print "Average execution time: %f seconds"%(self.tottime/self.c
        self.lastreport = time.time()


def taskTimer( n, subproc_args ):
    logger = Tlogger()

    for x in range(n):
        logger.logstart()
        p = subprocess.Popen(subproc_args)
        p.wait()
        logger.logend()
    logger.report()


import timeit
import sys
```

```
def main( ):

    # for accurate timing of code segments
    s = """j = [4*n for n in range(50)]"""
    timer = timeit.Timer(s)
    rzlts = timer.repeat(5, 5000)
    for t in rzlts:
        print "Time for 5000 executions of statement = ",t

    # subprocess execution timing
    print "#times:",sys.argv[1]
    print "Command:",sys.argv[2:]
    print ""
    for k in range(3):
       taskTimer( int(sys.argv[1]), sys.argv[2:])

main()
```

Usage Example:
First argument is the number of times to iterate. Additional arguments are

```
C:>rateCounter.py 20 md5.exe
```

Racket[edit]

Python: Rational_Arithmetic.1[edit]

Works with: Python version 3.0

Python 3's standard library already implements a Fraction class:

```
from fractions import Fraction

for candidate in range(2, 2**19):
  sum = Fraction(1, candidate)
  for factor in range(2, int(candidate**0.5)+1):
    if candidate % factor == 0:
      sum += Fraction(1, factor) + Fraction(1, candidate // factor)
  if sum.denominator == 1:
    print("Sum of recipr. factors of %d = %d exactly %s" %
           (candidate, int(sum), "perfect!" if sum == 1 else ""))
```

It might be implemented like this:

Python: Ray-casting_algorithm[edit]

```
from collections import namedtuple
from pprint import pprint as pp
import sys

Pt = namedtuple('Pt', 'x, y')               # Point
Edge = namedtuple('Edge', 'a, b')           # Polygon edge from a to b
Poly = namedtuple('Poly', 'name, edges')    # Polygon

_eps = 0.00001
_huge = sys.float_info.max
_tiny = sys.float_info.min

def rayintersectseg(p, edge):
    ''' takes a point p=Pt() and an edge of two endpoints a,b=Pt() of a line
    '''
    a,b = edge
    if a.y > b.y:
        a,b = b,a
    if p.y == a.y or p.y == b.y:
        p = Pt(p.x, p.y + _eps)

    intersect = False

    if (p.y > b.y or p.y < a.y) or (
        p.x > max(a.x, b.x)):
        return False

    if p.x < min(a.x, b.x):
        intersect = True
    else:
        if abs(a.x - b.x) > _tiny:
            m_red = (b.y - a.y) / float(b.x - a.x)
        else:
            m_red = _huge
        if abs(a.x - p.x) > _tiny:
            m_blue = (p.y - a.y) / float(p.x - a.x)
        else:
            m_blue = _huge
        intersect = m_blue >= m_red
    return intersect

def _odd(x): return x%2 == 1

def ispointinside(p, poly):
    ln = len(poly)
    return _odd(sum(rayintersectseg(p, edge)
                    for edge in poly.edges ))
```

```
def polypp(poly):
    print ("\n  Polygon(name='%s', edges=(" % poly.name)
    print ('    ', ',\n    '.join(str(e) for e in poly.edges) + '\n    ))')

if __name__ == '__main__':
    polys = [
      Poly(name='square', edges=(
        Edge(a=Pt(x=0, y=0), b=Pt(x=10, y=0)),
        Edge(a=Pt(x=10, y=0), b=Pt(x=10, y=10)),
        Edge(a=Pt(x=10, y=10), b=Pt(x=0, y=10)),
        Edge(a=Pt(x=0, y=10), b=Pt(x=0, y=0))
        )),
      Poly(name='square_hole', edges=(
        Edge(a=Pt(x=0, y=0), b=Pt(x=10, y=0)),
        Edge(a=Pt(x=10, y=0), b=Pt(x=10, y=10)),
        Edge(a=Pt(x=10, y=10), b=Pt(x=0, y=10)),
        Edge(a=Pt(x=0, y=10), b=Pt(x=0, y=0)),
        Edge(a=Pt(x=2.5, y=2.5), b=Pt(x=7.5, y=2.5)),
        Edge(a=Pt(x=7.5, y=2.5), b=Pt(x=7.5, y=7.5)),
        Edge(a=Pt(x=7.5, y=7.5), b=Pt(x=2.5, y=7.5)),
        Edge(a=Pt(x=2.5, y=7.5), b=Pt(x=2.5, y=2.5))
        )),
      Poly(name='strange', edges=(
        Edge(a=Pt(x=0, y=0), b=Pt(x=2.5, y=2.5)),
        Edge(a=Pt(x=2.5, y=2.5), b=Pt(x=0, y=10)),
        Edge(a=Pt(x=0, y=10), b=Pt(x=2.5, y=7.5)),
        Edge(a=Pt(x=2.5, y=7.5), b=Pt(x=7.5, y=7.5)),
        Edge(a=Pt(x=7.5, y=7.5), b=Pt(x=10, y=10)),
        Edge(a=Pt(x=10, y=10), b=Pt(x=10, y=0)),
        Edge(a=Pt(x=10, y=0), b=Pt(x=2.5, y=2.5))
        )),
      Poly(name='exagon', edges=(
        Edge(a=Pt(x=3, y=0), b=Pt(x=7, y=0)),
        Edge(a=Pt(x=7, y=0), b=Pt(x=10, y=5)),
        Edge(a=Pt(x=10, y=5), b=Pt(x=7, y=10)),
        Edge(a=Pt(x=7, y=10), b=Pt(x=3, y=10)),
        Edge(a=Pt(x=3, y=10), b=Pt(x=0, y=5)),
        Edge(a=Pt(x=0, y=5), b=Pt(x=3, y=0))
        )),
      ]
    testpoints = (Pt(x=5, y=5), Pt(x=5, y=8),
                  Pt(x=-10, y=5), Pt(x=0, y=5),
                  Pt(x=10, y=5), Pt(x=8, y=5),
                  Pt(x=10, y=10))

    print ("\n TESTING WHETHER POINTS ARE WITHIN POLYGONS")
    for poly in polys:
        polypp(poly)
        print ('   ', '\t'.join("%s: %s" % (p, ispointinside(p, poly))
                               for p in testpoints[:3]))
        print ('   ', '\t'.join("%s: %s" % (p, ispointinside(p, poly))
                               for p in testpoints[3:6]))
        print ('   ', '\t'.join("%s: %s" % (p, ispointinside(p, poly))
                               for p in testpoints[6:]))
```

Sample output

```
 TESTING WHETHER POINTS ARE WITHIN POLYGONS

  Polygon(name='square', edges=(
```

Python: Read-eval-print_loop[edit]

```
Start the interpreter by typing python at the command line (or select it



python
Python 2.6.1 (r261:67517, Dec  4 2008, 16:51:00) [MSC v.1500 32 bit (Int
win32
Type "help", "copyright", "credits" or "license" for more information.
```