Wstęp do informatyki - notatki

Piotr Karamon

Contents

1 Sqrt

```
def sqrt(n):
    i = 0
    while (n := n - (2*i+1)) >= 0:
        i += 1
    return i

print(sqrt(100))
```

```
10
```

2 Bisection method

2.1 Square root

```
EPSILON = 1e-10

def sqrt(n):
    a,b = 0,n
    midpoint =(a+b)/2
    while (errorabs := abs(midpoint*midpoint - n)) > EPSILON:
        midpoint =(a+b)/2
        if (midpoint*midpoint - n) < 0:
            a = midpoint
        else:
            b = midpoint
    return midpoint

print(sqrt(2))
```

```
1.4142135623842478
```

2.2 For a general function

```
f = lambda x: x**x - 2022

a,b = 1,5
while abs(y:=f(mid:=(a+b)/2)) > 1e-16:
    if y < 0:
        a,b = mid, b
    elif y > 0:
        a,b = a, mid

print((a+b)/2)
```

```
4.832071392109466
```

3 IsPrime

3.1 Basic

```
import math
def is_prime(n):
    if n <= 1: return False
    i = 2
    while i*i <= n:
```

```python
        if n % i == 0:
            return False
        i+=1
    return True

for i in range(20):
    print(i, is_prime(i))
```

```
0 False
1 False
2 True
3 True
4 False
5 True
6 False
7 True
8 False
9 False
10 False
11 True
12 False
13 True
14 False
15 False
16 False
17 True
18 False
19 True
```

3.2 How it should look like

```python
import math

def is_prime(n):
    if n == 2 or n == 3:
        return True
    if n % 2 == 0 or n % 3 == 0 or n <= 1:
        return False
    i = 5
    while i <= math.isqrt(n):
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 4
    return True

def is_prime_basic(n):
    if n <= 1: return False
```

```
    i = 2
    while i*i <= n:
        if n % i == 0:
            return False
        i+=1
    return True

print(all(is_prime(x) == is_prime_basic(x) for x in range(-2, 10_000)))
```

```
True
```

4 Sum of digits

```
def sum_of_digits(n):
    total = 0
    while n != 0:
        total += n % 10
        n = n // 10
    return total

print(sum_of_digits(35201))
```

```
11
```

5 GCD AND LCM

5.1 Introduction

The greatest common divisor(NWD in polish) of two integers a,b is the biggest integer which divides both of them. Meaning $a = de$ and $b = df$, the d is the $GCD(a, b)$. There is also a special case when one of the a and b is equal to 0. Then the value of the GCD is equal to the absolute value of the non-zero integer.

$$GCD(a, 0) = GCD(0, a) = |a|$$

$GCD(0, 0)$ is **commonly** defined as 0, but some authors leave it as undefined. GCD is **always** positive.

5.2 Example

$$54 = 27 * 2 = 3^3 * 2^1$$

The divisors of 54 are: $1, 2, 3, 6, 9, 18, 27, 54$

$$24 = 2^3 * 3^1$$

The divisors of 24 are: $1, 2, 3, 4, 6, 8, 12, 24$
Common divisors: $1, 2, 3, 6$
So the $GCD(54, 24) = 6$

5.3 Coprime number(względnie pierwsze)

a and b are said to be coprime if and only if $gcd(a, b) = 1$

5.4 LCM(NWW najmniejsza wspólna wielokrotoność)

$$LCM(a, b) = \frac{|a * b|}{GCD(a, b)}$$

6 Prime factorization

$$54 = 27 * 2 = 3^3 * 2^1$$

$$3^3 * 2^1$$

is the prime factorization of 54.

Prime factorization is a reduction of a number to its prime factors(with powers attached).

```
from collections import defaultdict

def prime_factors(n):
    i, factors = 2, []
    while n != 1:
        m = 0
        while n%i == 0:
            m += 1
            n //= i
        if m > 0:
            factors.append((i,m))
        i+=1
    return factors

print(prime_factors(3**3*2**3*7**2))
```

```
[(2, 3), (3, 3), (7, 2)]
```

6.1 Linear combination

$$gcd(a, b) = ax + by$$

Where $a, b \in Z$

6.2 Code example

```
def gcd(p,q):
    if q == 0:
        return p
    return gcd(q, p%q)
print(gcd(2*3*5*7*11*13,3*5*11))
```

```
165
```

```
def gcd(a,b):
    while b != 0:
        a,b = b, a%b
    return a
print(gcd(27*3, 54*3))
```

```
81
```

6.3 GCD(a,b,c)

7 Calculating PI

```
factor = 0.5**0.5
product = 1
for _ in range(1000):
    product*= factor
    factor = (0.5 + 0.5*factor) ** 0.5

print(2/product)
```

```
3.1415926535897927
```

8 Calculating cube roots

```
def cbrt(k):
    error = lambda x: x**3 - k
    g = k/3 # guess
    while abs(err:=error(g)) > 1e-10:
```

```
        g = g - err / (2*g**2)
    return g

print(cbrt(8))
```

```
1.9999999999933642
```

9 Calculating e

```
factorial = 1
e = 0
for i in range(1, 10000):
    e += 1/factorial
    factorial *= i
print(e)
```

```
2.7182818284590455
```

10 IsPalindrome

```
def is_palindrome_decimal(n):
    orig_n = n
    rn = 0
    while n !=0:
        digit = n % 10
        rn = 10*rn + digit
        n = (n - digit) // 10
    return rn == orig_n

def is_palindrome_binary(n):
    orig_n = n
    rn = 0
    while n !=0:
        digit = n % 2
        rn = 2*rn + digit
        n = (n - digit) // 2
    return rn == orig_n

print(is_palindrome_decimal(321123))
print(is_palindrome_decimal(391123))
print(is_palindrome_binary(0b10101))
print(is_palindrome_binary(0b11101))
```

```
True
False
True
False
```

11 A_n sequence

```
def num_steps(start):
    a_n = start
    steps = 0
    while a_n != 0:
        steps += 1
        a_n = (a_n %2)*(3*a_n + 1) + (1-a_n%2) * a_n /2
    return s
```

12 Generating subsets

A subset of a n element set can be fully describe as a binary sequence of length n. This binary sequence indicates presence or absence of a element. Example:

```
Set A = 1,2,3,4
Subset s = 0110
meaning s contains 2,3 and does not contain 1,4
```

Generating subsets comes down to generating those binary sequences of length n

12.1 Generating all subsets

```
A = [1,2,3,4]
subset = [0]*len(A)

def subsets(n, p):
    if n == p:
        print('elements', [A[i] for i in range(len(subset)) if subset[i]
        ↪   == 1], subset)
        return
    subset[p] = 0
    subsets(n, p+1)
    subset[p] = 1
    subsets(n,p +1)

subsets(len(A), 0)
```

```
elements [] [0, 0, 0, 0]
elements [4] [0, 0, 0, 1]
elements [3] [0, 0, 1, 0]
elements [3, 4] [0, 0, 1, 1]
elements [2] [0, 1, 0, 0]
elements [2, 4] [0, 1, 0, 1]
elements [2, 3] [0, 1, 1, 0]
elements [2, 3, 4] [0, 1, 1, 1]
elements [1] [1, 0, 0, 0]
elements [1, 4] [1, 0, 0, 1]
elements [1, 3] [1, 0, 1, 0]
elements [1, 3, 4] [1, 0, 1, 1]
elements [1, 2] [1, 1, 0, 0]
elements [1, 2, 4] [1, 1, 0, 1]
elements [1, 2, 3] [1, 1, 1, 0]
elements [1, 2, 3, 4] [1, 1, 1, 1]
```

12.2 Next subset

```
A = [1,2,3,4]
subset = [0]*len(A)

def next_subset(subset):
    i = 0
    while i < len(subset) and subset[i] == 1:
        i+=1
    # looping behaviour
    if i==len(subset):
        for i in range(len(subset)):
            subset[i] = 0
        return subset
    subset[i] = 1
    for j in range(0, i):
        subset[j] = 0
    return subset


get_elements = lambda subset: [A[i] for i in range(len(subset)) if
↪   subset[i] == 1]
subset = [0,0,0,0]
for i in range(18):
    print(next_subset(subset), get_elements(subset))
```

```
[1, 0, 0, 0] [1]
[0, 1, 0, 0] [2]
[1, 1, 0, 0] [1, 2]
[0, 0, 1, 0] [3]
```

```
[1, 0, 1, 0] [1, 3]
[0, 1, 1, 0] [2, 3]
[1, 1, 1, 0] [1, 2, 3]
[0, 0, 0, 1] [4]
[1, 0, 0, 1] [1, 4]
[0, 1, 0, 1] [2, 4]
[1, 1, 0, 1] [1, 2, 4]
[0, 0, 1, 1] [3, 4]
[1, 0, 1, 1] [1, 3, 4]
[0, 1, 1, 1] [2, 3, 4]
[1, 1, 1, 1] [1, 2, 3, 4]
[0, 0, 0, 0] []
[1, 0, 0, 0] [1]
[0, 1, 0, 0] [2]
```

12.3 Subsets with specified size

Say we have an n element set and we are only interested in k element subsets of it. Any set with n elements we can represent as $A = \{1, 2, 3, ..., n\}$.

Example:

```
A=1,2,3,4,5
We want only subsets with size 3.
1 2 3
1 2 4
1 2 5
1 3 4
1 3 5
1 4 5
2 3 4
2 3 5
2 4 5
3 4 5
```

```
def next_subset(n, subset):
    i = 0
    if subset[i] < n:
        subset[i]+=1
        return subset
    else:
        pass
    return subset

n, x = 5,[3,2,1]

print(x)
for i in range(10):
```

```
    x = next_subset(n,x)
    print(x)
```

```
[3, 2, 1]
[4, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
[5, 2, 1]
```

13 Divisors

```
from math import sqrt
def divisors(k):
    i = 1
    while i <= sqrt(k):
        if k % i == 0:
            print(i)
            print(-i)
            j = k/i
            if j != i:
                print(j)
                print(-j)
        i += 1

divisors(6)
```

```
1
-1
6.0
-6.0
2
-2
3.0
-3.0
```

14 Find divisors with the min sum

```
from math import sqrt
def find_ab(n):
    i = 1
    divisor = 1
    while i <= sqrt(n):
        if n % i == 0:
            divisor = i
        i += 1
    return divisor, n // divisor

print(find_ab(120))
```

```
(10, 12)
```

15 Convert to 2-16 system

```
def print_in_base(n, base):
    digits = []
    while n != 0:
        digits.append(n % base)
        n //= base
    for i in range(len(digits) - 1, -1, -1):
        digit = digits[i]
        if digit <= 9:
            print(digit, end='')
        else:
            print(chr(ord('A') + digit - 10), end='')
    print()
print_in_base(255, 16)
print_in_base(7, 2)
```

```
FF
111
```

16 Same digits

```
def same_digits(a,b):
    return get_digit_counts(a) == get_digit_counts(b)

def get_digit_counts(n):
    counts = [0 for _ in range(9)]
    while n != 0:
```

```
        counts[n % 10] += 1
        n //= 10
    return counts

print(same_digits(1122334, 4223311))
```

```
True
```

17 Sieve of Eratosthenes

```
def sieve(N):
    numbers = list(range(2, N))
    for i in range(len(numbers)):
        if numbers[i] != -1:
            cross_out_multiples(numbers, numbers[i], i+1)

    return [n for n in numbers if n != -1]

def cross_out_multiples(numbers, divisor, start_index):
    for i in range(start_index, len(numbers)):
        if numbers[i] % divisor == 0:
            numbers[i] = -1

print(sieve(100))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
↪ 71, 73, 79, 83, 89, 97]
```

18 Calculating e again

```
def calculate_e(N):
   nom = 1
   n, fact_n = 2, 2
   limit = 10**(N+3)
   while fact_n < limit:
      nom = nom*(n+1) + 1
      n, fact_n = n+1, fact_n * (n+1)

   a,b = nom, fact_n
   print('2.',end='')
   for i in range(N):
       print(10*a // b, end='')
       a = 10*a % b
calculate_e(30)
```

```
2.718281828459045235360287471352
```

19 Area under a graph

We can approximate the area under a graph using what is known as *rectangle method*. In the following example we are calculating the area under a graph of $y = 1/x$ in the interval $[1, k)$.

```
def area(start, end, f):
    x = start
    dx = 1e-6
    area = 0
    while x < end:
        area += dx * f(x)
        x += dx
    return area
print(area(1, 4, lambda x: 1/x))
```

```
1.3862947361296067
```

20 Different digits in other bases

```
def diff_digits(a, b):
    for base in range(2, 17):
        if not have_common_digit(a, b, base):
            return base
    return -1


def have_common_digit(a, b, base):
    digits_in_b = [0 for _ in range(base)]
    while b != 0:
        digits_in_b[b%base] = 1
        b //= base

    while a != 0:
        if digits_in_b[a % base] == 1:
            return True
        a //= base

    return False


print(diff_digits(123, 522))
```

```
11
```

21 $\mathbf{x^x = 2020}$

We have to solve the following equation $x^x = 2020$ using the Newton method. The derivative of x^x is equal to $x^x * (ln(x) + 1)$. We will use the following formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```python
from math import log
f = lambda x: x**x - 2020
f_prime = lambda x: (x**x) * (log(x) + 1)

x = 20
for _ in range(1000):
    x = x - f(x)/f_prime(x)
print(x)
```

```
4.831687113003211
```