Project 1 - restructured

April 24, 2021

1 Project 1

1.1 0. Table of contents

1.2 1. Introduction

1.2.1 1.A. Prime numbers

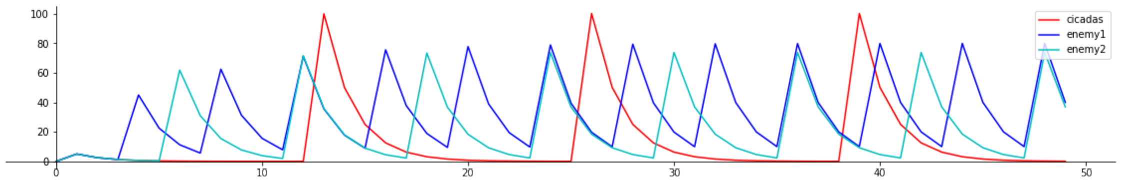
1.A.a. What are prime numbers? A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers. Natural numbers greater than 1 that are not prime numbers are called composite numbers. In other words: a number n is a prime number if it is greater than 1 and may only be divided by 1 or itself without leaving a remainder. Christlieb von Clausberg: https://books.google.de/books?id=X-wVBuaPiPsC&pg=PA86#v=onepage&q&f=false

Examples: - For prime numbers: - 2 - 3 - 11 - For composite numbers: - 4, factors: 2,2 - 6, factors: 2,3 - 22, factors: 2,11

Ther are infinitely many primes, but there is no known simple formula that allows us to predict and separate prime numbers from composite numbers. To determine whether a number is a prime number or not, we need to test each number by using a trial and error approach, that gets quite time expensive the larger the numbers get. Furthermore, the larger the numbers get, the scarcer the prime numbers are.

1.A.b. Prime numbers in nature

[1]:
```
import matplotlib.pyplot as plt

def life_cycle(interval, growth, decline):
    life = [0]
    populus = 10
    for i in range(1,50,1):
        if i%interval == 0:
            populus += 10
            populus *= growth
        else:
            populus *= decline
        if populus < 0:
            populus = 0
        life.append(populus)
    return life


cicadas = life_cycle(13, 10, 0.5)
enemy1 = life_cycle(4, 4, 0.5)
enemy2 = life_cycle(6, 6, 0.5)

fig = plt.figure()
fig.set_figwidth(20)
fig.set_figheight(3)
ax = fig.add_subplot(1, 1, 1)

ax.spines['left'].set_position('zero')
ax.spines['bottom'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

plt.plot(cicadas,'r', label='cicadas')
plt.plot(enemy1,'b', label='enemy1')
plt.plot(enemy2,'c', label='enemy2')

plt.legend(loc='upper right')

plt.show()
```



Singing cicadas live in the United States and only mate every 13 or 17 years. For example, the American Magicicada septendecim only leaves its underground hiding place after exactly 17 years in order to reproduce within a period of about three weeks. The larvae that hatch from the eggs live underground until they crawl to the surface of the earth almost on the same day in 17 years. A Chilean-German research team has found out why it only crawls out of its underground hiding place after 17 years. 13 and 17 are prime numbers. Since their enemies and competitors usually live in 2, 4 or 6 year rhythms, *the cicadas can increase their chances of survival by reproducing in the "low birth" cohorts of their predators.*

This theory is supported by our diagram. Animals with even reproduction circles often overlap, while cicadas only overlap with other species every (frequency of the cicada cycle * frequency of the enemies cycle) cycles. This makes reproduction for cicadas very reliable when compared to their natural opposition.

During their short aboveground life from mid-May to June, the cicadas do not cause any damage despite their massive occurrence.

Source: Gene Kritsky, PhD, Periodical Cicadas: The Brood X Edition, https://www.amazon.com/gp/product/B08X3XKRW7/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&cre
20&linkId=1b83fa112465e182eabdfa1616119d8f

1.A.c. Prime numbers in information systems Prime numbers are used in cryptographic algorithms, particularly in RSA (Rivest - Shamir - Adlerman). RSA is asymetric, meaning while how to encode messages for the reciever is public kwnoledge, only the reciever itself knows how to deciver that message. Contrary to symetric encodings like the enigma, where both the sender and the reciever had to posses kwnoledge of a designatet de- and encryption process, no prior agreements between sender and reciever need be made than a broad understanding of the RSA - encryption.

The Process

1. Select to prime numbers q and p.
2. Let N = q x p and (N) = (p - 1) x (q - 1).
3. Choose an e with 1 < e < (N) and where the greatest common divisor of e and (N) = 1.
4. Calculate a d with the values e and (N) that satisfies e x d - k x (N) = 1. There is only one possible solution. The variable k is calculated but not needed.
5. To encode a number: G = (T^e) % N
6. To decode a number: T = (G^d) % N

Why does it work? (((T^e) % N)^d) % N = T The private key (d, N) cannot be constructed from the publik key (e, N) without considerable effort. To calculate d, (N) is needed. Finding q and p from N takes longer the larger numbers are used. Should someone find (N), the message can be decoded, but since the process is asymetric every message can simply be encoded slightly different, making long term communication fast, as easy as possible and as secure as possible.

Possible problems This algorithm depends on the lack of understanding about prime numbers. If that chages, major parts of the internet are not secure anymore. Also, the encrytion does not change the pattern of the message, so a black-and-white image still shows the same shapes. Techniques like Alphabetical Probability are still able to find possible matches faster, the bigger the message gets.

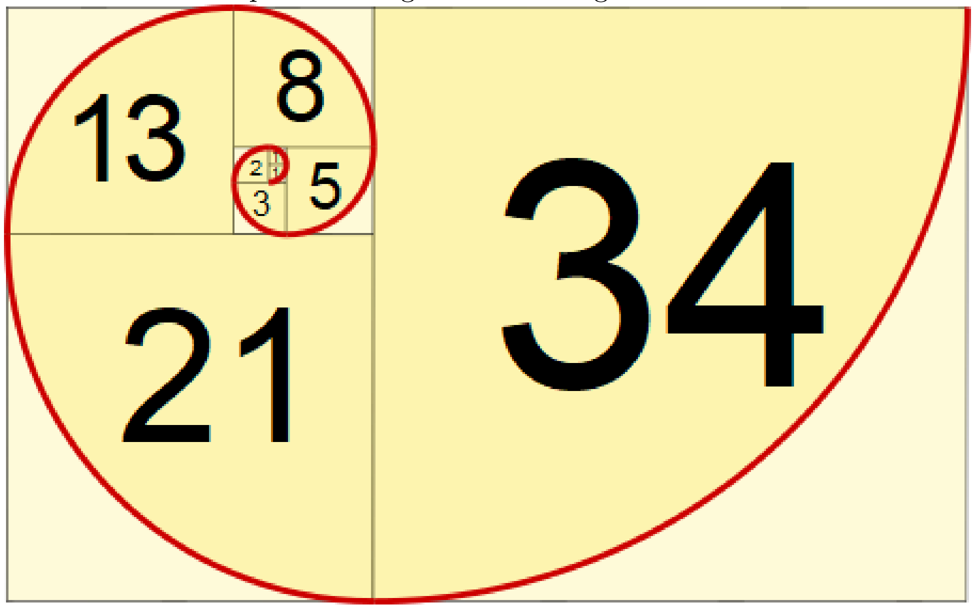
Sources: Christian Spannagel 2012, zur Verfügung gestellt von der Technischen Informationsbibliothek: https://doi.org/10.5446/19815, https://doi.org/10.5446/19816, https://doi.org/10.5446/19817, https://doi.org/10.5446/19813, https://doi.org/10.5446/19814, https://www.inf.hs-flensburg.de/lang/krypto/algo/euklid.htm

1.2.2 1.B. Fibonacci numbers

1.A.a. What are Fibonacci numbers Fibonacci numbers are numbers that form a sequence called the Fibonacci sequence. A new Fibonacci number is calculated by the sum of its two preceding ones. The sequence starts with 0 and 1.

Example: 0, 1, 1, 2, 3, 5, 8, 13, ...

These numbers may also be displayed graphically for a better visualization. We create squares based on the successive Fibonacci numbers. And by drawing circular arcs connecting the opposite corners of squares we get a resulting line which is called the



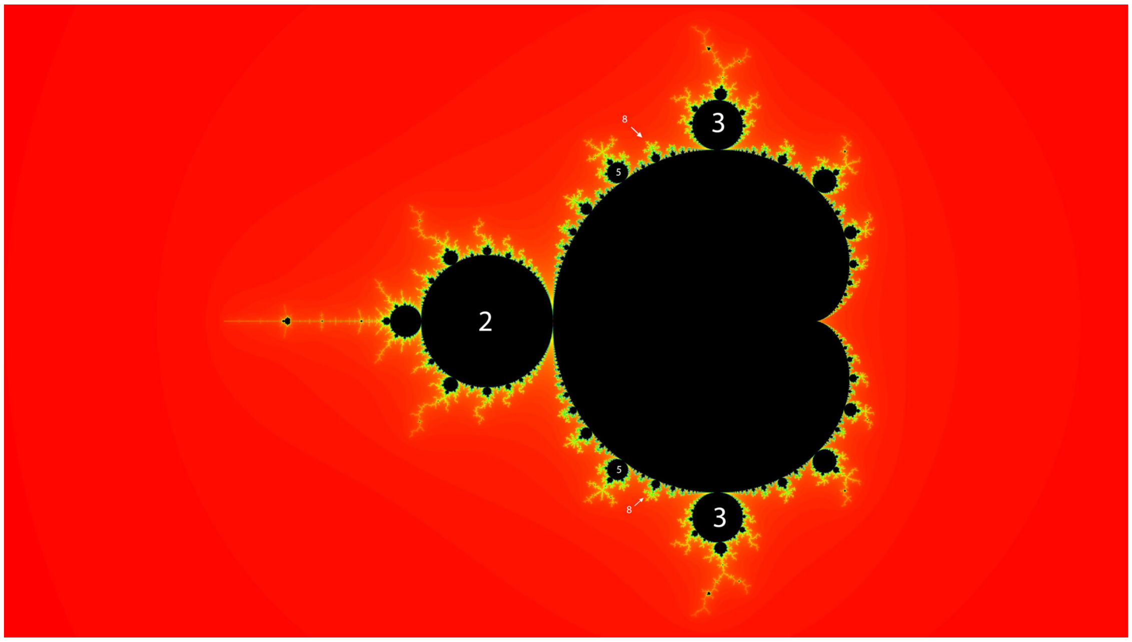
Fibonacci spiral.
[]https://www.mathsisfun.com/numbers/fibonacci-sequence.html

1.A.b. Fibonacci numbers in nature The Fibonacci numbers can often be observed throughout nature. One example of this occurrence would be the sunflowers seed arrangements as they closely resemble the Fibonacci numbers. It contains spiral patterns in 2 opposite directions which are typically dependent on each other. For example the seed arrangement from the left to the inside contains the x position of the Fibonacci numbers while from the right it takes the x + 1 position (usually 13 seeds left-oriented and 21 right-oriented). But the sunflower isn't the only species of plants that has those properties also another one, called Anonium, contains a leaf arrangement similiar to that of the Fiibonacci numbers. In fact an estimate by R. Jean even estimates that 92 percent of plants with displaying spirals or multijugate phyllotaxis have Fibonacci similar properties. Also there is the shell of a snail known as Nautilus pompilius it produces Chambers in a way that closely resembles the golden ratio.

Source 1: Richard A. Dunlap: The Golden Ratio and Fibonacci Numbers. World Scientific, Singapur 1999, ISBN 981-02-3264-0, P. 130–134 Source 2: Alfred S. Posamentier, Ingmar Lehmann: The (Fabulous) FIBONACCI Numbers, Prometheus Books, New York 2007 Source 3: Roger V. Jean: Phyllotaxis: A Systemic Study in Plant Morphogenesis, Cambridge University Press, Cambridge 1994

1.A.c. Fibonacci numbers in physics In science the numbers can be observed in an experiment called LEED which stands for Low-energy electron diffraction and is a very common experiment for surface experiments as it is inexpensive and easy to perform. And as such has been used for a very long time for determining surface geometries. During this process data is acquired most commonly through a camera connected to a Pc. When observing the quasicrystalline structures on quasicrystals, the diffraction pattern is similar to the Fourier transform of a Fibonacci array where there is a linear sequence of one long and one short element. This is also referred to as an one- dimensional aperiodic order as there consists an order but without periodicity.

Source 1: R D Diehl, J Ledieu, N Ferralis, A W Szmodis and R McGrath: Low-energy electron diffraction from quasicrystal surfaces. Journal of Physics Condensed Matter, 13 January 2003, URL: http://stacks.iop.org/JPhysCM/15/R63 Source 2: Michael Baake David Damanik und Uwe Grimm: Aperiodic Order and Spectral Properties. Mathematisches Forschungsinstitut Oberwolfach, 2017, URL: https://imaginary.org/sites/default/files/snapshots/snapshot-2017-003.pdf #### 1.A.d. Fibonacci numbers in mathematics In the Mandelbrot set there are also different parts where the Fibonaccci numbers can be observed. One example of this phenomenon would be when observing the branches leaving the "circles" as those divide in specific numbers so for example the biggest one has (including the one going back to the "circle") two branches while the second biggest has 3 the third in between 5 the fourth in between 8 and so on. Source: Dr Holly Krieger from Murray Edwards College, University of Cambridge 2017



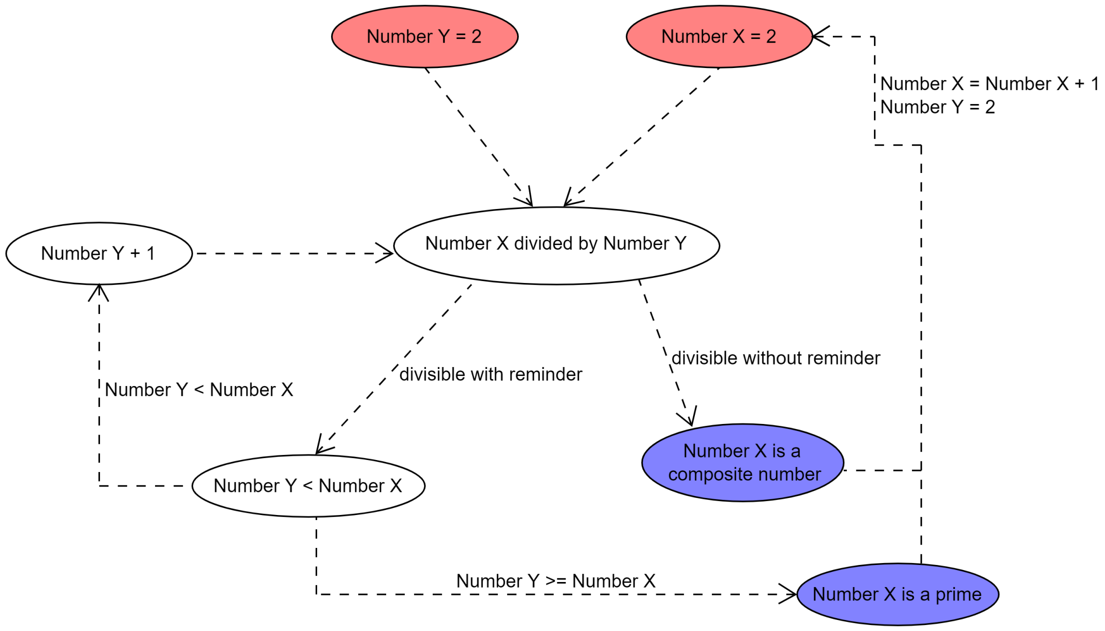
build with: Mandelbrot Set generator, URL: https://math.hws.edu/eck/js/mandelbrot/MB.html

1.3 2. Methods and approach

1.3.1 2.A. Exercise 1 - Prime numbers in systems

The program of exercise 1 calculates all primes smaler than x = 1,000. Every x between 1,000 and 2 is a prime if no y bigger than 2 and smaler than x/2 is able to divide x without reminder.

As seen in our flowchart, the process of determining primes can be quiet simple but tedious.



[2]:
```python
# Exercise 1

primes = []
circles1 = 0
for integer in range(2, 1000, 1):                 #Start, stop, step
    prime = True
    for j in range(2, int(integer/2) + 1, 1):
        circles1 += 1                              #How often we check for a prime
        if integer % j == 0:                       #Is there a number that divides i?
            prime = False
    if prime:
        primes.append(integer)
print(primes)
print(circles1)
```

```
[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, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647,
653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757,
761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983,
991, 997]
248502
```

1.3.2 2.B. Exercise 2 - Prime number algorithm - simple optimization

The algorithm in exercise 1 does not check for a possible early exit of the loop. But in case the program finds a j bigger than 2 and smaler than i/2 that divides i without reminder, we can define i as a composite number and exit the inner loop immediatly for increased performance.

This aproach will take more and more time the more integer and j grows. To save even more computing power we use our already found prime numbers to search for large numbers that are not divisable by those primes. We use prime factorization. This saves many more calculations since we do not divide by even numbers exept 2 and other unneccesesary numbers.

[3]:
```python
# Exercise 2

#This algorith calculates prime numbers and collects them in a list.
#It is trimmed for performance, especially when the integers get larger.
integer = 2
primes = []
circles2 = 0
while(integer < 1000):
    j = 0
    isPrimeNumber = True
    while(j < len(primes) and isPrimeNumber):
        #We only test if the current integer is divisible by any of the
        #already collected prime numbers. If it is not divisible we have
        #found a new prime number.
        circles2 = circles2 + 1    #How often we check for a prime
        if(integer % primes[j] == 0 and primes[j] < integer/2 + 1):
            isPrimeNumber = False #Ensures an early exit of the loop
        j = j + 1

    if(isPrimeNumber):
        primes.append(integer) #Add the new prime number to the list

    integer = integer + 1

print(primes)
print(circles2)
```

```
[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, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,
167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251,
257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557,
563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647,
653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757,
761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983,
991, 997]
15619
```

This greatly reduces the neccessery number of calculations and in turn saves time and memory.

[4]:
```
# Optimisation Check
print("Without optimization: " + format(circles1,",") + ", with simple␣
→optimization: "+ format(circles2,",") + ".")
if circles1 > circles2:
    print("We saved " + format(circles1 - circles2,",") + " loops.")
else:
    print("Why even bother?")
```

```
Without optimization: 248,502, with simple optimization: 15,619.
We saved 232,883 loops.
```

1.3.3 2.C. Exercise 3 - Fibonacci in systems

[5]:
```python
# Exercise 3
import math

def fibs_up_to(num1, num2, limit):
    """
    return: Tuble
    last two digits of the Fibonachi sequence up to limit
    """
    num1b, num2b = num2, num2 + num1   # Fibonachi logic
    if num2b < limit:                  # Recursive Condition
        num1, num2 = fibs_up_to(num1b, num2b, limit) # Recursive Call
    return num1, num2

def tiles_area(x):
    a, b = fibs_up_to(0, 1, x)         # fibonachi up to x
    return (a*(a+b))                   # area formula square

def fibs_sqrt(x):
    counter = 0
    Fibonacci_list = [0,1]
    while counter <= x:
        last = int(Fibonacci_list[len(Fibonacci_list)-1])
        last_2 = int(Fibonacci_list[len(Fibonacci_list)-2])
        current = last + last_2
        root = math.sqrt(current)
        if root%1 == 0:
            if root**2 == current:               # testing if root is a real␣
 ↪squareroot
                print(current,int(root))         # print the number as well as␣
 ↪the root
        Fibonacci_list.append(int(current))
        counter += 1
```

[6]:
```python
print(format(tiles_area(10000), ","))
```

```
45,765,226
```

[7]:
```python
fibs_sqrt(100)
```

```
1 1
144 12
```

[8]:
```python
# Area of the Fibonachi spiral

# Area Formula for each square: math.pi * pow(num2,2) / 4
# import math

def spiral_area(num1, num2, limit):
    area = 0
    # Fibonachi logic
    num1b, num2b = num2, num2 + num1
    # Recursive Condition
    if num2b < limit:
        # Recursive Call
        area = (math.pi * pow(num2, 2) / 4) + spiral_area(num1b, num2b, limit)
    # Stop
    return area

print(spiral_area(0, 1, 23))
```

```
214.41369860750336
```

1.3.4 2.D. Exercise 4 - Prime numbers and functions

Written as a function, our aproach looks like this.

This is still very much inefficent though, if you want to find out more about high-performance prime generation, check out D. J. Bernstein's: https://cr.yp.to/primegen.html and Daniel Scocco's: https://www.programminglogic.com/testing-if-a-number-is-prime-efficiently/ for more information.

[9]:
```
# Exercise 4

from math import isqrt

def is_prime(x):
    """
    return: Boolean
    True if prime, else False
    """
    for i in range(2, isqrt(x) + 1, 1):
        if x % i == 0:
            return False
    return True
```

1.4 3. Results

1.5 4. Discussion and application

1.5.1 Prime numbers in Cryptography - an example

Find two primes with x digits.

[10]:
```
from random import randint
```

```
def set_of_random_ints(hero):
    """
    finds two different numbers from 0 to hero (inclusive)
    param: int
    return: int, int
    """
    a = randint(0, hero)
    b = randint(0, hero)
    if a == b:  # to guarantee two unique numbers
        a, b = randint(hero)
    return a, b


def generate_primes(x):
    """
    finds all primes with x digits and selects two at random
    param: int
    return: int, int
    """
    sol = []
    for i in range(pow(10, x), pow(10, x+1), 1): # pow(10, x) gives a number␣
 ↪with x digits
        if is_prime(i):
            sol.append(i)
    a, b = set_of_random_ints(len(sol)-1) # guarantee two unique numbers
    return sol[a], sol[b]
```

Find an e with 1 < e < (N) and where the greatest common divisor of e and (N) = 1.

[11]:
```
# from random import randint as cryforhelp

def gcd(a, b):
    """
    eucledian algorithm by Hans Werner Lang, @https://www.inf.hs-flensburg.de/
 ↪lang/index.htm
    param: int, int
    return: int
    """
    while b != 0:
        c = a % b
        a = b
        b = c
    return a

def find_e(phi_of_N):
    """
    finds an random int with 1 < int < phi_of_N and where greatest common␣
↪divisor of int and phi_of_N = 1
    param: int
    return: int
    """
    e = []
    for i in range(2, phi_of_N, 1):
        if gcd(i, phi_of_N) == 1:
            e.append(i)
    return e[randint(0, len(e)-1)]
```

Find a d with the values e and (N) that satisfies e x d - k x (N) = 1.

[12]:
```
def extgcd(a, b):
    """
    extended eucledian algorithm by Hans Werner Lang, @https://www.inf.
↪hs-flensburg.de/lang/index.htm
    solves: gcd(a,b) = s * a + t * b, returns gcd(a,b), s, t
    param: int, int
    return: int, int, int
    """
    if b == 0:
        return a, 1, 0
    else:
        g, u, v = extgcd(b, a % b)
        q = a//b
        return g, v, u-q*v

def find_d(e, phi_of_N):
    """
    solve e * x - y * phi_of_N = 1 and return x
    param: int, int
    return: int
    """
    gcd, s, t = extgcd(e, phi_of_N)
    return s
```

Now, generate your keys.

[13]:
```
def rsa_keys(limit):
    """
    takes in digits of prime numbers to be used
    public key: {N, e}, private key: {q, p, N, phi_of_N, e, d}
    param: int
    return: dict, dict
    """
    q, p = generate_primes(limit)
    N, phi_of_N = q * p, (q-1) * (p-1)
    e = find_e(phi_of_N)
    d = find_d(e, phi_of_N)
    public, private = {"N": N, "e": e}, {"q": q, "p": p, "N": N, "phi_of_N":␣
↪phi_of_N, "e": e, "d": d}
    if d < 1:
        public, private = rsa_keys(limit)
    return public, private
```

This is how the RSA - encription works.

[14]:
```
public, private = rsa_keys(2)

message = 2649 # theoretical pin code

secret = pow(message, public["e"])%public["N"]
message = pow(secret, private["d"])%private["N"]

print("secret: "+ str(secret))
print("message: "+ str(message))

for key, value in private.items():
    print(key, ' : ', value)
```

```
secret: 52122
message: 2649
q  :  479
p  :  653
N  :  312787
phi_of_N  :  311656
e  :  38295
d  :  113399
```

1.6 5. Summary and lookout

All in all, prime numbers seem to play a pretty dominant role in our world, even though it may not be obvious at first. The consepts are mostly pretty simple, but most problems conserning primes do not seem to have smart solutions.

Fibonacci numbers seem to play quite a large role in nature too. The fascinating thing about the Fibonacci sequence are not its practical applications simillar to primes, but rather its surprising talent to be hidden in seemingly random or incoherent patterns.

But in order to find these coherencies we need to look for patterns and this will take time. Most likely prime and Fibonacci numbers are present in a few other fields as well, and these patterns we have yet to discover.

1.7 6. Appendices