Project

April 2, 2021

1 Primes

Prime numbers are natural numbers greater than 1 that is not a product of two natural numbers. A number n is prime if it is greater than 1 and no number 2,..n-1 divide n evenly.

Exercise 1.1 calculates all primes smaler that x = 1,000 ''' all primes = a boldface capital P '''für latex Every x between 10000 and 2 is a prime if no y bigger than 2 and smaler than x/2 is able to divide x without reminder.

[2]:
```
# Exercise 1.1

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

```
[2, 3, 4, 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]
247506
```

In case the program finds a y bigger than 2 and smaler than x/2 that divides x without reminder, we can define x as a composite number.

[3]:
```python
# Exercise 1.2

sol = []
circles2 = 0
for i in range(2, 1000, 1):
    prime = True
    for j in range(2, int(i/2), 1):
        circles2 += 1
        if i % j == 0:
            prime = False
            break                           # not a prime no more runs
    if prime:
        sol.append(i)
print(sol)
print(circles2)
```

```
[2, 3, 4, 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]
39875
```

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

[16]:
```python
# 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: 247,506, with simple optimization: 39,875.
We saved 207,631 loops.
```

[4]:
```python
# Exercise 1.4

def is_prime(x):
    """
    return: Boolean
    True if prime, else False
```

```
    """
    for j in range(2, int(i/2), 1):
        if i % j == 0:
            return False
    return True
```

[5]:
```
# Validity test

valid = []
for i in range(2, 1000, 1):
    func = is_prime(i)                    # should already know the answer
    prime = True
    for j in range(2, int(i/2), 1):
        if i % j == 0:
            prime = False
            if not func and not prime: # did both get the same result?
                valid.append(True)
            break
    if func and prime:                 # did both get the same result?
        valid.append(True)

for i in valid:
    if not i:
        print("ähem")
```

1.1 Prime Numbers in Cryptography

Prime numbers are used in cryptographic algorithms, particularly in RSA, Source: https://de.wikipedia.org/wiki/RSA-Kryptosystem.

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,

1.2 Prime Numbers in Nature

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. 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&crea
20&linkId=1b83fa112465e182eabdfa1616119d8f

2 Fibonachi

[6]:
```
# Exercise 1.3

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
```

[7]:
```
print(format(tiles_area(10000), ","))
```

```
45,765,226
```

[8]:
```
# 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
```

[ ]:
```
Recherchieren Sie je ein Beispiel für Anwendung von Primzahlen in der Informatik␣
 ↪und in der Natur und stellen Sie es kurz ca. 1/4 Seite plus Diagramme oder␣
 ↪Bilder. Zitieren Sie Ihre Quellen.
```

```
Recherchieren Sie je ein weiteres Beispiel der Fibbonacci Fliesen/Spiralen in
Natur und Infomatik (Mathematik) und stellen Sie es kurz dar.
```