Computational Physics Assignment № 1

Cristopher Cerda Puga, Universidad de Colima August 25, 2023

Problem 1

The program below checks whether a number n is a prime or not: if n is a prime it returns 1, 0 if it is not. For example $\mathbf{prime(5)}$ returns 1, whereas $\mathbf{prime(6)}$ returns 0.

Listing 1: A simple python program that checks whether a number n is a prime (can be improved).

```
1 def prime(n):
2     eta = 1
3     for i in range (n -2):
4         ratio = float ( n )/ float ( i +2)
5         fratio = math . floor ( ratio )
6     if ratio - fratio == 0.0:
7         eta = 0
8         break
9     return eta
```

We can check the performance of the program by verifying if 15485863 is prime: the command $\%\mathbf{time\ prime(15485863)}$ returns the time of execution. Of course, the timing depends on the hardware of the computer where the program is run and whether you are running multiple processes at the same time.

1. Explain the program.
   - The function takes an argument n which is the number to evaluate.
   - Then, it initializes a variable $\mathbf{eta}$ to 1. This variable will be use to keep track of whether the number is prime or not.
   - The *for* loop iterates over the variable i initialized to 0 and ending at $n-2$.
   - Next, the variable $\mathbf{ratio}$ is calculated dividing $\frac{n}{i+2}$. The reason to add 2 is because its nonsense to divide by 0 and 1.
   - The variable $\mathbf{fratio}$ removes the decimal point and give us the closest integer to that number given by $\mathbf{ratio}$ (for example $\mathbf{math.floor(1.2) = 1}$).
   - Finally, if the condition that the difference between $\mathbf{ratio}$ and $\mathbf{fratio}$ is equal to 0 is satisfied, this means that the loop found a divisor of n, assigning a value of 0 to $\mathbf{eta}$.

2. Can you think of an improvement to this program?
   Yes, there is a better algorithm that only takes into account the range of possible divisor of n in the range of $(n, n^{1/2} + 1)$. Also, another improvement could be using the modulus of n with each divisor as a condition to determine if n is prime or not:

Listing 2: Improved code.

```
def bestprime(n):
    if n<=1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

3. Assuming that you wrote your own program for checking if n is prime, compare its execution time with the time taken with the original:
   - The original one takes 1.9073486328125e-06 seconds; while the improved one takes only 1.430511474609375e-06 seconds.

Problem 2

Use the program in the previous problem and write a program that finds all the primes below a certain value n.

Listing 3: Finding primes below $\mathbf{n}$.

```
def bestprime(n):
    if n<=1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True


def findprimes(n):
    primes = [] # We create a list where the primes will be stored
    for i in range(2,n+1):
        if prime(i): # If the prime(n) function returns True for the ith ↩
            number, then is stored into the list
            primes.append(i)
    return primes
```

```
16
17 print(findprimes(n))
```

For example, if we set $n = 100$:

Listing 4: Output.

```
1 print(findprimes(100))
2 ## [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]
```

Problem 3

Now that we have managed to calculate the factorial of a number, let's consider a generalization: imagine that p_n is the n^{th} prime and consider the quantity

$$p_n\# \equiv p_n p_{n-1} \dots p_2 p_1 \tag{1}$$

This number is called the primorial of p_n. For instance $3\# = 6 = 3!$ and $5\# = 30 \neq 5!$

1. Write a program the calculates the primorial of a given prime.

   Using the functions written above, it is enough to perform iterative multiplications on the list $\mathbf{primes}$:

   Listing 5: Primorial of n

   ```
   1 def primorial(n):
   2     primes = findprimes(n)
   3     primorial=1
   4     for i in primes:
   5         primorial *= i
   6     return primorial
   ```

   For example, setting $n = 5$:

   ```
   1 print(primorial(5))
   2 ## 30
   ```

2. Starting at $p_1 = 2$ and going up to the some prime p_N, check wether $p_i\# \pm 1$ $(i = 1, \dots, N)$ are primes. These primes are called *primorial primes*.

Listing 6: Primorial primes

```
def primorial(n):
    if bestprime(n):
        primes = findprimes(n)
        primorial=1
        for i in primes:
            primorial *= i
        if bestprime(primorial + 1) or bestprime(primorial - 1):
            return n
        else:
            return 0
    else:
        return 0


primorials = []
n = 1
while n<50:
    n+=1
    if primorial(n) != 0:
        primorials.append(primorial(n))

print(primorials)
## [2, 3, 5, 7, 11, 13, 31, 41]
```

Using the same functions above and doing a little modification to our $\mathbf{primorial}()$ function. We get the primorial primes below an $N = 15$

Problem 4

1. Define your own function, let's call it $\mathbf{Q}(\mathbf{n}, \mathbf{d})$, that when applied to a number n, it returns the $\mathbf{d^{th}}$ digit (counting from the first digit to the left of the decimal and going to the left). For instance

$$Q(1032, 3) = 0$$

If you have a float number, and you want to consider also the decimals, you can easily do so by multiplying by an appropriate factor. For example:

$$Q(3.1415926535897932385 * 10^{10}, 3) = 5$$

```
def Q(n,d):
    n = str(n)
    digits = list(n)

    if '.' in digits:
```

```
        index = digits.index('.')
        del digits[index+1:]
        digits.pop(index)
        digits = [int(i) for i in digits]
        return digits[(d-len(digits))*(-1)]
    else:
        digits = [int(i) for i in digits]
        return digits[(d-len(digits))*(-1)]


n = float(input('Enter a number: '))
d = int(input('Enter the digit: '))
print(Q(n,d))
```

For example, if we set $n = 1234.5678$, $d = 3$:

```
print(Q(1234.5678,3))
## 2
```

2. Imagine that n is a large integer (to make thing simpler): use your "Q" function to define a function that generates an array, whose components are the digits of n.

```
def lstdigits(n):
    n = str(n)
    if '.' in n:
      n = n.replace('.','')
    digits = list(n)
    digits = [int(i) for i in digits]
    return digits
```

For example, using the function above for the first 200 natural numbers:

```
n = 0
digits_list = []
while n<200:
    n+=1
    digits_list.append(Q(n,d))
print(digits_list)

## [[1], [2], [3], [4], [5], [6], [7], [8], [9], [1, 0], [1, 1], [1, ↩
   2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8], [1, 9], [2, ↩
   0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, ↩
   8], [2, 9], [3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, ↩
```

```
6], [3, 7], [3, 8], [3,9], [4, 0], [4, 1], [4, 2], [4, 3], [4, 4],↩
 [4, 5], [4, 6], [4, 7], [4, 8], [4, 9], [5, 0], [5, 1], [5, 2], ↩
[5, 3], [5, 4], [5, 5], [5, 6], [5, 7], [5, 8], [5, 9], [6, 0], ↩
[6, 1], [6,2], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8], [6,↩
 9], [7, 0], [7, 1], [7, 2], [7, 3], [7, 4][7, 5], [7, 6], [7, 7],↩
 [7, 8], [7, 9], [8, 0], [8, 1], [8, 2], [8, 3], [8, 4], [8, 5], ↩
[8, 6], [8, 7], [8, 8], [8, 9], [9, 0], [9, 1], [9, 2],[9, 3], [9,↩
 4], [9, 5], [9, 6], [9, 7], [9, 8], [9, 9], [1, 0, 0], [1, 0, 1],↩
 [1, 0, 2], [1, 0, 3], [1, 0, 4], [1, 0, 5], [1, 0, 6], [1, 0, ↩
7],[1, 0, 8], [1, 0, 9], [1, 1, 0], [1, 1, 1], [1, 1, 2], [1, 1, ↩
3], [1, 1, 4], [1, 1, 5], [1, 1, 6], [1, 1, 7], [1, 1, 8], [1, 1, ↩
9], [1, 2, 0],[1, 2, 1], [1, 2, 2], [1, 2, 3], [1, 2, 4], [1, 2, ↩
5], [1, 2, 6], [1, 2, 7], [1, 2, 8], [1, 2, 9], [1, 3, 0], [1, 3, ↩
1], [1, 3, 2], [1, 3, 3], [1, 3, 4], [1, 3, 5], [1, 3, 6], [1, 3, ↩
7], [1, 3, 8], [1, 3, 9], [1,4, 0], [1, 4, 1], [1, 4, 2], [1, 4, ↩
3], [1, 4, 4], [1, 4, 5], [1, 4, 6],[1, 4, 7], [1, 4, 8], [1, 4, ↩
9], [1, 5, 0], [1, 5, 1], [1, 5, 2], [1, 5, 3], [1, 5, 4], [1, 5, ↩
5], [1, 5, 6], [1, 5, 7], [1, 5, 8], [1, 5, 9], [1, 6, 0], [1, 6, ↩
1], [1, 6, 2], [1, 6, 3], [1, 6, 4], [1, 6, 5], [1, 6, 6], [1, 6, ↩
7], [1, 6, 8], [1, 6, 9], [1, 7, 0], [1, 7, 1], [1, 7, 2],[1, 7, ↩
3], [1, 7, 4], [1, 7, 5], [1, 7, 6], [1, 7, 7], [1, 7, 8], [1, 7, ↩
9], [1, 8, 0], [1, 8, 1], [1, 8,2], [1, 8, 3], [1, 8, 4], [1, 8, ↩
5],[1, 8, 6], [1, 8, 7], [1, 8, 8], [1, 8, 9], [1, 9, 0], [1, 9, ↩
1], [1, 9, 2], [1, 9, 3], [1, 9, 4], [1, 9, 5], [1, 9, 6], [1, 9, ↩
7], [1, 9, 8],[1, 9, 9], [2, 0, 0]]
```