Fundamentos da Programação

O Crivo de Eratóstenes

Aula 11

José Monteiro

(slides adaptados do Prof. Alberto Abad)

Listas: _List Comprehensions_

- O Python suporta um conceito chamado *list comprehensions* que pode ser usado para construir listas de uma maneira muito natural e fácil, parecido como na matemática.
- As *lists comprehensions* é uma das componentes de Python relacionados com **programação funcional** que iremos a ver em mais pormenor nas próximas semanas.
- A seguir estão formas comuns de descrever vetores (ou tuplos ou listas) em matemática:
  - $S = \{x^2 : x \text{ in } \{0 \ldots 9\}\}$
  - $V = (1, 2, 4, 8, \ldots, 2^{12})$
  - $M = \{x \mid x \text{ in } S \text{ and } x \text{ even}\}$

Listas: _List Comprehensions_

- Em Python, é possível escrever essas expressões quase exatamente como um matemático faria, sem precisar de se lembrar de nenhuma sintaxe críptica especial!

```
>>> S = [x**2 for x in range(10)]
>>> V = [2**i for i in range(13)]
>>> M = [x for x in S if x % 2 == 0]
```

In [ ]:

Crivo de Eratóstenes

Problema: Dado um número `n`, imprima todos os primos menores ou iguais a `n`. Por exemplo, se `n = 20`, a saída deve ser `2, 3, 5, 7, 11, 13, 17, 19`.

A crivo de Eratóstenes é uma das formas mais eficientes de encontrar todos os primos menores que `n` quando `n` for menor que 10 milhões. Algoritmo:

- Crie uma lista de inteiros consecutivos de 2 a n: `lista = [2, 3, 4, ..., n]`.
- Seleciona o primeiro elemento da lista, `c = 2`.
- Enquanto `c` não for maior que $\sqrt{n}$:
  - (a) removem-se da lista todos os múltiplos de c;
  - (b) passa-se ao número seguinte na lista.
- No final do algoritmo, a lista apenas contém números primos.

Crivo de Eratóstenes

Proposta 1

In [1]:

```
from math import sqrt

def crivo1(n):

    lista = list(range(2,n+1))

    i = 0
    while lista[i] <= sqrt(n):
        c = lista[i]
        j = i + 1

        while j < len(lista):
            if lista[j] % c == 0:
                del lista[j]
            else:
                j += 1

        i = i + 1

    return lista

print(crivo1(50))
%timeit -n 100 crivo1(50)
# Question1: Será que podemos começar com uma lista menor? Reparem que o úl
# Question2: Porque não podemos utilizar for?
# Question3: Será que podemos abstrair em uma função (mais eficiente) a eli
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
19.9 µs ± 1.5 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Crivo de Eratóstenes

Proposta 2

In [2]:

```
def crivo2(n):
    # reduzimos a lista inicial a metade tirando os pares
    lista = [2] + list(range(3, n+1, 2))

    i = 1    # no 0 está o 2 e já eliminámos os pares
    limite = sqrt(n)
    while lista[i] <= limite:
        # abstraímos a eliminação de múltiplos numa função que consiga uti
        elimina_multiplos(lista, i)
        i = i + 1

    return lista


def elimina_multiplos(lista, index):
    c = lista[index]
    fim = len(lista) -1
    for i in range(fim, index, -1):
        if lista[i] % c == 0:
            del lista[i]

print(crivo2(50))
%timeit -n 100 crivo2(50)
# Question 1: Será que podemos manter o tamanho da lista fixo e utilizar fo
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
6.42 µs ± 151 ns per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Crivo de Eratóstenes

Proposta 3

- Criar uma lista de tamanho n+1 com valores lógicos (inicializados a True) indicando se o inteiro correspondente ao índice é, ou não, um número primo
- A lista ter tamanho fixo permite não ter que testar todos os elementos, mas ir diretamente aos múltiplos do crivo

In [3]:

```
def crivo3(n):

    # lista de bools correspondente a inteiros de 0..n indicando se é ou nã
    lista = [True]*(n+1)
    lista[0], lista[1] = False, False

    # equivalente ao primeiro while anterior
    for i in range(2, int(sqrt(n)) + 1):
        # colocamos a False as posições multiplas de i
         if lista[i]:
            for j in range(i*i, n+1, i):
                lista[j] = False

    return [i for i in range(n+1) if lista[i]]

print(crivo3(50))
%timeit -n 100 crivo3(50)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
5.97 µs ± 110 ns per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Crivo de Eratóstenes

Proposta 4

- A versão anterior voltava a usar uma lista completa, se tirarmos os pares fica muito mais eficiente, mas os índices da lista deixam de corresponder aos valores, a relação é `valor = 3 + 2i` , em que `i` é o índice

In [4]:

```
def crivo4(n):
    # lista de bools correspondente aos ímpares de 3..n indicando se é ou n
    # índice i corresponde ao valor 3 + 2*i
    lista = [True] * ((n - 1) // 2)

    # equivalente ao primeiro while anterior
    for i in range(0, (int(sqrt(n)) - 1) // 2):
        if lista[i]:
            # colocamos a False as posições múltiplas de 3 + 2*i
            # começando na posição ((3+2i)**2-3)//2
            for j in range(3 + 6*i + 2*i*i, (n - 1) // 2, 3 + 2*i):
                lista[j] = False

    return [2] + [3 + 2*i for i in range((n - 1) // 2) if lista[i]]

print(crivo4(50))
%timeit -n 100 crivo4(50)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
4.73 µs ± 153 ns per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Crivo de Eratóstenes

Eficiência

In [11]:

```
n = 10000
%timeit -n 10 crivo1(n)
print()

%timeit -n 10 crivo2(n)
print()

%timeit -n 10 crivo3(n)
print()

%timeit -n 10 crivo4(n)
print()
```

```
11.6 ms ± 1.16 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

3.48 ms ± 202 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

1.16 ms ± 42.5 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

596 µs ± 74.9 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Listas - Tarefas próxima aula

- Trabalhar matéria apresentada hoje:
  - Experimentar todos os programas dos slides!