18 Reunião 19: 24/06/2021

Figure 1: Dilbert de Scott Adams

Reuniões passadas

- listas
- criar listas
- acessar itens da lista através de **índices**
- percorrer listas
- função `len()`
- comando de repetição `for ... in range(ini, fim, passo):...`
- função `sum()`
- fatias
- fatias em python
- apelidos × clones

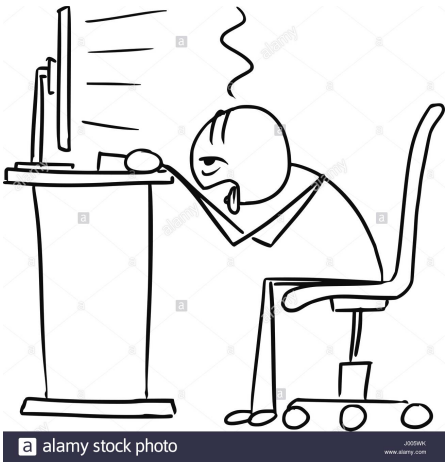

Figure 2: Fonte: www.alamy.com

18.1 Hoje

Figure 3: Fonte: Dreamstime.com

- Mutabilidade
- Listas são bichos mutáveis
- Fatias são clones
- Atribuições só criam ou alteram apelidos, variáveis
- Atribuições não criam clones

18.2 Exercício: risque multiplos

Escreva uma função `risque_multiplos()` que *recebe* um inteiro positivo `k` e uma lista `crivo` e *altera* a lista atribuindo `False` a toda posição de `crivo` cujo índice é um múltiplo de `k` maior que `k`. Por exemplo,

```
In [2]: crivo = [True] * 10
In [3]: crivo
Out[3]: [True, True, True, True, True, True, True, True, True, T
In [4]: risque_multiplos(2, crivo)
In [5]: crivo
Out[5]: [True, True, True, True, False, True, False, True, False
```

Figure 4: Fonte: Dreamstime.com

18.2.1 Solução

Está função é **mutadora** pois altera a lista que é recebida como parâmetro.

```
#-------------------------------------------------------------------
def risque_multiplos(k, crivo):
    '''(int, list) -> None

    RECEBE um inteiro k > 0 e uma lista crivo.
    ALTERA a lista atribuindo False a toda posição de crivo
      cujo índice é um múltiplo de k maior que k.

    EXEMPLO:

    In [2]: crivo = [True]*10
    In [3]: crivo
    Out[3]: [True, True, True, True, True, True, True, True, True,
    In [4]: risque_multiplos(2,crivo)
    In [5]: crivo
    Out[6]: [True, True, True, True, False, True, False, True, Fal
    '''
    n = len(crivo)
    for i in range(k+k, n, k):
        crivo[i] = False
```

Figure 5: Fonte: Dreamstime.com

18.3 Apelidos versus Clones

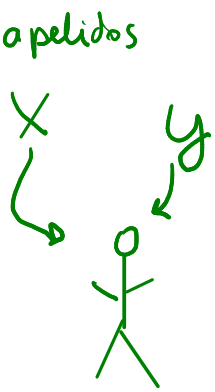

Figure 6: Fonte: sciencenews.org

Listas são coisas, objetos **mutáveis**. Podemos alterar seu componentes ou acrescentar novos componentes.

```
In [1]: lstA = [1, 2, 3, 'oi']

In [2]: lstA
Out[2]: [1, 2, 3, 'oi']

In [3]: lstA[1] = None

In [4]: lstA
Out[4]: [1, None, 3, 'oi']

In [5]: lstA = lstA + [True]

In [6]: lstA
Out[6]: [1, None, 3, 'oi', True]
```

Fatias são cópias idênticas ou **clones** de uma sublista

```
In [9]: cloneA = lstA[:]

In [10]: cloneA
Out[10]: [1, None, 3, 'oi', True]
```

```
In [11]: cloneA[4] = 3.14

In [12]: cloneA
Out[12]: [1, None, 3, 'oi', 3.14]

In [13]: lstA
Out[13]: [1, None, 3, 'oi', True]
```

Atribuições só criam ou modificam **apelidos**, não criam clones. Váriáveis são apelidos, nomes que damos a coisas, objetos que pretendemos usar mais tarde.

```
In [14]: lstB = lstA # lstB e lstA agora são apelidos para uma

In [15]: lstA
Out[15]: [1, None, 3, 'oi', True]

In [16]: lstB
Out[16]: [1, None, 3, 'oi', True]

In [17]: lstA[1] = False

In [18]: lstA
Out[18]: [1, False, 3, 'oi', True]

In [19]: lstB
Out[19]: [1, False, 3, 'oi', True]
```

Para identificar se duas variáveis são apelidos para o mesmo objeto podemo usar o operador `is`.

```
In [20]: lstA is lstB
Out[20]: True

In [21]: lstA == lstB
Out[21]: True
```

Dois objetos pode ser iguais sem ser o mesmo objeto.

```
In [22]: lstC = lstB[:] # lstC é apelido para um clone de B

In [23]: lstC
Out[23]: [1, False, 3, 'oi', True]

In [24]: lstB
Out[24]: [1, False, 3, 'oi', True]

In [25]: lstB == lstC
Out[25]: True

In [26]: lstB is lstC
Out[26]: False
```

É evidente de duas variáveis são apelidos para um mesmo objetos então elas são iguais.

```
In [31]: lstD = lstC # apelidos para uma mesma lista

In [32]: lstD
Out[32]: [1, False, 3, 'oi', True]

In [33]: lstC
Out[33]: [1, False, 3, 'oi', True]

In [34]: lstD == lstC
Out[34]: True

In [35]: lstD is lstC
Out[35]: True
```

18.4 Exercício: crivo de Eratóstenes

Escreva um programa que leia um número natural `n` e imprima todos os primos menores ou iguais a `n`.

Exemplos

```
Programa que imprime todos os primos menores que ou igual a n
Digite n: 20
Primos: 2 3 5 7 11 13 17 19

Programa que imprime todos os primos menores que ou igual a n
Digite n: 100
Primos: 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
```

Crivo de Erastótenes

Se você está em busca de inspiração, talvez o que segue ajude.

No século três A.C., o astrônomo grego Eratóstenes desenvolveu um algoritmo para determinar todos os números primos até um dado número inteiro positivo `n`. Para aplicar o algoritmo, inicialmente, escrevemos a lista dos inteiros entre 2 e `n`. Por exemplo, se `n` fosse 20 teríamos a lista

```
2 3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
```

Em seguida, selecione o primeiro número da lista indicando que encontramos um primo. Agora, percorremos a lista riscando todos os múltiplos do número selecionado, já que nenhum deles é primo.

Após executarmos o primeiro passo do algoritmo teríamos selecionado 2 e riscado todos os múltiplos de 2:

```
2 3  X  5  X  7  X  9  XX  11  XX  13  XX  15  XX  17  XX  19  XX
```

Agora simplesmente repetimos o processo selecionando o primeiro número da lista que não tenha sido selecionado e que não tenha sido riscado. No caso selecionamos e riscamos todos os múltiplos desse número. No exemplo, selecionamos 3 e riscamos os seus múltiplos obtendo:

```
2 3  X  5  X  7  X  X  XX  11  XX  13  XX  XX  XX  17  XX  19  XX
```

Repetindo esse processo até que todo número na lista tenha sido selecionado ou tenha sido riscado chegamos a:

```
2 3  X  5  X  7  X  X  XX  11  XX  13  XX  XX  XX  17  XX  19  XX
```

Os números que não foram riscados são primos e os demais são compostos.

Esse algoritmo para gerar essa lista de primos é chamado de **Crivo de Era-tóstenes**.

Veja a seguir uma animação desse algoritmo copiada da página Crivo de Eratóstenes na Wikipédia.

Implementar é legal.
Vejam os passos

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers

Esqueleto de solução

Aqui vai uma proposta de esqueleto de solução

(a) Escreva uma função `risque_multiplos()` que *recebe* um inteiro positivo `k` e uma lista `crivo` e *altera* a lista atribuindo `False` a toda posição de `crivo` cujo índice é um múltiplo de `k` maior que `k`. Por exemplo,

```
In [2]: crivo = [True] * 10
In [3]: crivo
Out[3]: [True, True, True, True, True, True, True, True, True, T
In [4]: risque_multiplos(2, crivo)
In [5]: crivo
Out[5]: [True, True, True, True, False, True, False, True, False
```

(b) Escreva uma função `crivo_eratostenes()` que *recebe* um inteiro `n` $\geq$ `0` e *retorna* uma lista com todos os números primos até `n` inclusive. Os primos na lista devem estar em ordem crescente. Por exemplo,

```
In [6]: crivo_eratostenes(29)
Out[6]: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
In [7]: crivo_eratostenes(20)
Out[7]: [2, 3, 5, 7, 11, 13, 17, 19]
```

(c) Escreva uma função `main()` que *lê* um inteiro `n` $\geq$ `0` e *imprime* todos os números primos números primos até `n` inclusive.

Solução

```
def main():
    '''() -> None
    Programa que LÊ um número inteiro n >= 0.
    IMPRIME todos os primos menores ou igual a n.
    O programa é uma implementação do Crivo de Eratóstenes.

    Exemplo:

    Programa que imprime todos os primos menores que ou igual a n
    Digite n: 20
    Primos: 2 3 5 7 11 13 17 19
    '''
    print("Programa que imprime todos os primos menores que ou igual
    n = int(input("Digite n: "))
    print("Primos: ", end="")
    primos = crivo_eratostenes(n)
    for i in range(len(primos)):
        print(f"{primos[i]}", end=" ") # end="" para não mudar de l
    print() # muda de linha
```

```python
def crivo_eratostenes(n):
    '''(int) -> list
    RECEBE um inteiro n >= 0.
    RETORNA uma lista com todos os números primos até n _inclusive_
    Os primos na lista devem estar em ordem crescente.
    Esta função deve ser uma implementação do Crivo de Eratóstenes
    deve utilizar a função risque_multiplos().

    Exemplo:
    In [9]: crivo_eratostenes(10)
    Out[9]: [2, 3, 5, 7]
    '''
    primos = []
    crivo = [True]*(n+1)
    if n > 1:
        crivo[0] = False
        crivo[1] = False
    primo = 2
    while primo <= n:
        # coloque o primo na lista
        primos += [primo]

        # risque os múltiplos do primo
        risque_multiplos(primo, crivo)

        # encontre o próximo primo
        i = primo + 1
        while i < n+1 and not crivo[i]:
            i += 1

        # i é o próximo primo menor ou igual a n ou n+1
        primo = i
    return primos
```

a lista que será retornada

passo 1

passo 2

passo preparatório para próxima iteração

o mesmo que crivo[i] == False

novo ator

retorna (apelido para) lista

```
#-----------------------------------------------------------------
def risque_multiplos(k, crivo):
    '''(int, list) -> None

    RECEBE um inteiro k > 0 e uma lista crivo.
    ALTERA a lista atribuindo False a toda posição de crivo
        cujo índice é um múltiplo de k maior que k.

    EXEMPLO:

    In [2]: crivo = [True]*10
    In [3]: crivo
    Out[3]: [True, True, True, True, True, True, True, True, True,
    In [4]: risque_multiplos(2,crivo)
    In [5]: crivo
    Out[6]: [True, True, True, True, False, True, False, True, Fal
    '''
    n = len(crivo)
    for i in range(k+k, n, k):
        crivo[i] = False

#-----------------------------------------------------------------
if __name__ == "__main__":
    main()
```