MAC 115 — Introdução à Ciência da Computação

Aula 16

Nelson Lago

IF noturno – 2023

Previously on MAC 115...

Coleções

A principal coleção em python é a *lista*:

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
```

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
```

vermelho

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

vermelho

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

---

vermelho

amarelo

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

---

vermelho

amarelo

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

---

vermelho

amarelo

Coleções

A principal coleção em python é a *lista*:

```
cores = ["vermelho", "azul", "amarelo"]
```

Uma lista é um conjunto ordenado:

```
print(cores[0])
print(cores[2])
```

---

vermelho

amarelo

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in primos:
  print("O número", p, "é primo")
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O número", primos[n], "é primo")
  n += 1
```

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in primos:
  print("O número", p, "é primo")
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O {}o primo é {}".format(n+1, primos[n]))
  n += 1
```

Repetições com coleções

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 0
while n < len(primos):
  print("O {}o primo é {}".format(n+1, primos[n]))
  n += 1
```

```
primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
n = 1
for p in primos:
  print("O {}o primo é {}".format(n, p))
  n += 1
```

A função range()

```
range(início, final, passo)
```

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**
- **O passo pode ser omitido; o padrão é um**

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**
- **O passo pode ser omitido; o padrão é um**
- **Se há dois parâmetros, eles são `início` e `final`**

A função `range()`

```
range(início, final, passo)
```

- **O início pode ser omitido; o padrão é zero**
- **O passo pode ser omitido; o padrão é um**
- **Se há dois parâmetros, eles são `início` e `final`**

O intervalo é sempre
fechado no início e aberto no final

A função range()

```
range(início, final, passo)
```

A função range()

```
range(início, final, passo)
```

O intervalo é sempre
fechado no início e aberto no final

A função range()

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

primeiro = início

A função range()

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

primeiro = início

$|\text{último}| < |\text{final}|$

A função range()

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

primeiro = início

|último| < |final|

A função range()

```
range(início, final, passo)
```

A função range()

```
range(início, final, passo)
```

O intervalo é sempre
fechado no início e aberto no final

A função `range()`

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

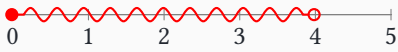
`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)

A função `range()`

`range(início, final, passo)`

O intervalo é sempre
fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)

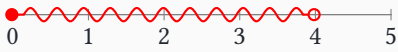


A função `range()`

`range(início, final, passo)`

O intervalo é sempre fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)



`range(2, 7)` → de dois a seis! (cinco elementos)

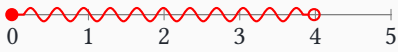
A função `range()`

`range(início, final, passo)`

O intervalo é sempre

fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)



`range(2, 7)` → de dois a seis! (cinco elementos)

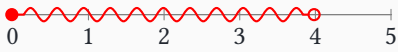
`range(1, 10, 2)` → de um a nove! (cinco elementos)

A função range()

`range(início, final, passo)`

O intervalo é sempre fechado no início e aberto no final

`range(4)` → `range(0, 4)` → de zero a três! (quatro elementos)



`range(2, 7)` → de dois a seis! (cinco elementos)

`range(1, 10, 2)` → de um a nove! (cinco elementos)

`range(1, 11, 2)` → de um a nove! (cinco elementos)

A função range()

```
range(início, final, passo)
```

A função `range()`

```
range(início, final, passo)
```

O intervalo é sempre

fechado no início e aberto no final

A função `range()`

`range(início, final, passo)`

O intervalo é sempre
fechado no início e aberto no final

O número total de elementos é $\left\lceil \frac{\text{final} - \text{início}}{\text{passo}} \right\rceil$

Coleções

Escreva uma função que recebe um número natural n como parâmetro e devolve uma lista com os n primeiros ímpares

Coleções

Escreva uma função que recebe um número natural n como parâmetro e devolve uma lista com os n primeiros ímpares

```
def ímpares(n):
    lista = []
    for i in range(         ):
        lista.append(i)
    return lista
```

Coleções

Escreva uma função que recebe um número natural n como parâmetro e devolve uma lista com os n primeiros ímpares

```
def ímpares(n):
    lista = []
    for i in range(1, 2*n, 2):
        lista.append(i)
    return lista
```

Coleções

Escreva uma função que recebe um número natural n e imprime uma contagem regressiva de n até zero

Coleções

Escreva uma função que recebe um número natural `n` e imprime uma contagem regressiva de `n` até zero

```
def regressiva(n):
```

Coleções

Escreva uma função que recebe um número natural `n` e imprime uma contagem regressiva de `n` até zero

```
def regressiva(n):

        print(i)
```

Coleções

Escreva uma função que recebe um número natural `n` e imprime uma contagem regressiva de `n` até zero

```
def regressiva(n):
    for i in range(         ):
        print(i)
```

Coleções

Escreva uma função que recebe um número natural `n` e imprime uma contagem regressiva de `n` até zero

```
def regressiva(n):
    for i in range(n, -1, -1):
        print(i)
```