MAC 110 — Introdução à Ciência da Computação

Aula 6

Nelson Lago

BMAC – 2024

Mais um exercício!!!!

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
```

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0
```

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0




print(n, "invertido é", invertido)
```

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:




print(n, "invertido é", invertido)
```

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10



print(n, "invertido é", invertido)
```

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10

print(n, "invertido é", invertido)
```

Exercício

Dado um número natural $n > 0$, imprima-o com os dígitos invertidos ("de trás para a frente"). Por exemplo, $6437 \rightarrow 7346$

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

Antes do início do laço:

invertido: 0
tmp: 6437

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido * 10`:

invertido: 0

tmp: 6437

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido + tmp % 10`:

invertido: 7

tmp: 6437

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`tmp = tmp // 10`:

invertido: 7

tmp: 643

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`while tmp > 0:`

invertido: 7

tmp: 643

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido * 10`:

invertido: 70

tmp: 643

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido + tmp % 10`:

invertido: 73

tmp: 643

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`tmp = tmp // 10`:

invertido: 73

tmp: 64

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`while tmp > 0:`

invertido: 73

tmp: 64

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido * 10`:

invertido: 730

tmp: 64

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido + tmp % 10`:

invertido: 734

tmp: 64

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`tmp = tmp // 10`:

invertido: 734

tmp: 6

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`while tmp > 0:`

invertido: 734

tmp: 6

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido * 10`:

invertido: 7340

tmp: 6

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`invertido = invertido + tmp % 10`:

invertido: 7346

tmp: 6

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`tmp = tmp // 10`:

invertido: 7346

tmp: 0

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

Exercício

Vamos experimentar com 6437?

`while tmp > 0:`

invertido: 7346

tmp: 0

```
n = int(input("Digite um número natural: "))
invertido = 0
tmp = n
while tmp > 0:
    invertido = invertido * 10
    invertido = invertido + tmp % 10
    tmp = tmp // 10
print(n, "invertido é", invertido)
```

and now for something completely different

Brincando com *strings*

- **Como imprimir estas frases?**
  - Preciso de um copo d'água

Brincando com *strings*

- **Como imprimir estas frases?**
  - Preciso de um copo d'água

```
print("Preciso de um copo d'água")
```

Brincando com *strings*

- **Como imprimir estas frases?**
  - Preciso de um copo d'água

    ```
    print("Preciso de um copo d'água")
    ```
  - Camões escreveu "Os Lusíadas" no século XVI

Brincando com *strings*

- **Como imprimir estas frases?**
  - Preciso de um copo d'água

    ```
    print("Preciso de um copo d'água")
    ```
  - Camões escreveu "Os Lusíadas" no século XVI

    ```
    print('Camões escreveu "Os Lusíadas" no século XVI')
    ```

Brincando com *strings*

- **Como imprimir estas frases?**

Brincando com *strings*

- **Como imprimir estas frases?**
  - Batatinha quando nasce
    Espalha a rama pelo chão

Brincando com *strings*

- **Como imprimir estas frases?**
  - Batatinha quando nasce

    Espalha a rama pelo chão

```
print("Batatinha quando nasce")
print("Espalha a rama pelo chão")
```

Brincando com *strings*

- **Como imprimir estas frases?**
  - Batatinha quando nasce

    Espalha a rama pelo chão

```
print("Batatinha quando nasce")
print("Espalha a rama pelo chão")
```

VEJA BEM...

Brincando com *strings*

- **Como imprimir estas frases?**
  - Batatinha quando nasce

    Espalha a rama pelo chão

```
print("Batatinha quando nasce")
print("Espalha a rama pelo chão")
```

VEJA BEM...

  - Camões bebeu muitos copos d'água escrevendo "Os Lusíadas"

Brincando com *strings*

- **Como imprimir estas frases?**
  - Batatinha quando nasce
    Espalha a rama pelo chão

```
print("Batatinha quando nasce")
print("Espalha a rama pelo chão")
```

VEJA BEM...

  - Camões bebeu muitos copos d'água escrevendo "Os Lusíadas"

```
print("Camões bebeu muitos copos d'água",
      'escrevendo "Os Lusíadas"')
```

Brincando com *strings*

- **Como imprimir estas frases?**
  - Batatinha quando nasce
    Espalha a rama pelo chão

```
print("Batatinha quando nasce")
print("Espalha a rama pelo chão")
```

VEJA BEM...

  - Camões bebeu muitos copos d'água escrevendo "Os Lusíadas"

```
print("Camões bebeu muitos copos d'água",
      'escrevendo "Os Lusíadas"')
```

VEJA BEM...

Brincando com *strings* — escapes

- O caracter “\” indica o início de uma *sequência de escape*: uma sequência de caracteres que é usada para representar algum outro em uma *string*

Brincando com *strings* — escapes

- **O caracter “\” indica o início de uma *sequência de escape*: uma sequência de caracteres que é usada para representar algum outro em uma *string***

```
print("Batatinha quando nasce\nEspalha a rama pelo chão")
```

Brincando com *strings* — escapes

- **O caracter “\” indica o início de uma *sequência de escape*: uma sequência de caracteres que é usada para representar algum outro em uma *string***

```
print("Batatinha quando nasce\nEspalha a rama pelo chão")
print("Camões bebeu muitos copos d'água escrevendo \"Os Lusíadas\"")
```

Brincando com *strings* — escapes

- **O caracter “\” indica o início de uma *sequência de escape*: uma sequência de caracteres que é usada para representar algum outro em uma *string***

```
print("Batatinha quando nasce\nEspalha a rama pelo chão")
print("Camões bebeu muitos copos d'água escrevendo \"Os Lusíadas\"")
print('Camões bebeu muitos copos d\'água escrevendo "Os Lusíadas"')
```

Brincando com *strings* — escapes

- **O caracter "\" indica o início de uma *sequência de escape*: uma sequência de caracteres que é usada para representar algum outro em uma *string***

```
print("Batatinha quando nasce\nEspalha a rama pelo chão")
print("Camões bebeu muitos copos d'água escrevendo \"Os Lusíadas\"")
print('Camões bebeu muitos copos d\'água escrevendo "Os Lusíadas"')
```

  - Espaços em branco "especiais" (\n, \t ...)
  - Caracteres "problemáticos" (\", \' ...)

Brincando com *strings* — escapes

- **Como imprimir esta frase:**

Brincando com *strings* — escapes

- **Como imprimir esta frase:**
  - O caracter '\' resolve "todos" os problemas do mundo

Brincando com *strings* — escapes

- **Como imprimir esta frase:**
  - O caracter '\' resolve "todos" os problemas do mundo

```
print('O caracter \'\\\' resolve "todos" os problemas do mundo')
print("O caracter '\\' resolve \"todos\" os problemas do mundo")
```

Brincando com *strings* — formatação

"Você digitou um total de 12 números: 7 números pares e 5 números ímpares"

```
n = 12
p = 7
i = 5
print("Você digitou um total de", n, "números:", p, "números pares e", i, "números ímpares")
```

Essa linha tem 92 caracteres; o recomendado é não passar de 79 ☹

Brincando com *strings* — formatação

"Você digitou um total de 12 números: 7 números pares e 5 números ímpares"

```
n = 12
p = 7
i = 5
print("Você digitou um total de", n, "números:",
      p, "números pares e", i, "números ímpares")
```

Brincando com *strings* — formatação

“Você digitou um total de 12 números: 7 números pares e 5 números ímpares”

```
n = 12
p = 7
i = 5
print("Você digitou um total de", n, "números:", end=" ")
print(p, "números pares e", i, "números ímpares")
```

Truques com `print()` — end e sep

- **`print()` não termina mudando para a próxima linha, mas sim com o que é definido por `end`**
  - (se você não definir end, python utiliza \n)

Truques com `print()` — end e sep

- **`print()` não termina mudando para a próxima linha, mas sim com o que é definido por `end`**
  - (se você não definir `end`, python utiliza `\n`)
- **`print()` não separa os itens com um espaço, mas sim com o que é definido por `sep`**
  - (se você não definir `sep`, python utiliza um espaço)

Brincando com *strings* — formatação

“São 14:16:02h”

```
horas = "14"
minutos = "16"
segundos = "02"
```

Brincando com *strings* — formatação

"São 14:16:02h"

```
horas = "14"
minutos = "16"
segundos = "02"
print("São", end=" ")
print(horas, minutos, segundos, sep=":", end="h\n")
```

Brincando com *strings* — formatação

“São 14:16:02h”

```
horas = "14"
minutos = "16"
segundos = "02"
```

Brincando com *strings* — formatação

"São 14:16:02h"

```
horas = "14"
minutos = "16"
segundos = "02"
print("São {0}:{1}:{2}h".format(horas, minutos, segundos))
```

Brincando com *strings* — formatação

"São 14:16:02h"

```
horas = "14"
minutos = "16"
segundos = "02"
print("São {0}:{1}:{2}h".format(horas, minutos, segundos))
print("São {}:{}:{}h".format(horas, minutos, segundos))
```

Brincando com *strings* — formatação

"São 14:16:02h"

```
horas = "14"
minutos = "16"
segundos = "02"
print("São {0}:{1}:{2}h".format(horas, minutos, segundos))
print("São {}:{}:{}h".format(horas, minutos, segundos))
print("São {h}:{m}:{s}h".format(h=horas, m=minutos, s=segundos))
```

Brincando com *strings* — formatação

```
import math
print("Pi pode ser aproximado para {0:.7f} ou {1:.4f}".format(math.pi, math.pi))
```

Brincando com *strings* — formatação

```
import math
print("Pi pode ser aproximado para {0:.7f} ou {1:.4f}".format(math.pi, math.pi))
```

Pi pode ser aproximado para 3.1415927 ou 3.1416

Brincando com *strings* — formatação

```
import math
print("Pi pode ser aproximado para {0:.7f} ou {1:.4f}".format(math.pi, math.pi))
print("Pi pode ser aproximado para {:.7f} ou {:.4f}".format(math.pi, math.pi))
```

Pi pode ser aproximado para 3.1415927 ou 3.1416

Brincando com *strings* — formatação

```
import math
print("Pi pode ser aproximado para {0:.7f} ou {1:.4f}".format(math.pi, math.pi))
print("Pi pode ser aproximado para {:.7f} ou {:.4f}".format(math.pi, math.pi))
print("Pi pode ser aproximado para {0:.7f} ou {0:.4f}".format(math.pi))
```

Pi pode ser aproximado para 3.1415927 ou 3.1416

Brincando com *strings* — formatação

```
import math
print("Pi pode ser aproximado para {0:.7f} ou {1:.4f}".format(math.pi, math.pi))
print("Pi pode ser aproximado para {:.7f} ou {:.4f}".format(math.pi, math.pi))
print("Pi pode ser aproximado para {0:.7f} ou {0:.4f}".format(math.pi))
print("Pi pode ser aproximado para {pi:.7f} ou {pi:.4f}".format(pi=math.pi))
```

```
Pi pode ser aproximado para 3.1415927 ou 3.1416
```

Brincando com *strings* — formatação

```
x = 1234.5678
print("x vale {:.7e}".format(x))
```

Brincando com *strings* — formatação

```
x = 1234.5678
print("x vale {:.7e}".format(x))
```

```
x vale 1.2345678e+03
```

Brincando com *strings* — formatação

Em geral: `{que:como}`

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: `A.BC`

A: largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)

Brincando com *strings* — preenchimento

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
```

```
Total: 5
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
```

```
Total: 5
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
```

```
Total: 5
Total:  5
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
print("Total: {:02}".format(5))
```

```
Total: 5
Total:  5
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
print("Total: {:02}".format(5))
```

```
Total: 5
Total:  5
Total: 05
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
print("Total: {:02}".format(5))
print("Total: {:2}".format(10))
```

```
Total: 5
Total:  5
Total: 05
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
print("Total: {:02}".format(5))
print("Total: {:2}".format(10))
```

```
Total: 5
Total:  5
Total: 05
Total: 10
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
print("Total: {:02}".format(5))
print("Total: {:2}".format(10))
print("Total: {:02}".format(10))
```

```
Total: 5
Total:  5
Total: 05
Total: 10
```

Brincando com *strings* — preenchimento

```
print("Total: {}".format(5))
print("Total: {:2}".format(5))
print("Total: {:02}".format(5))
print("Total: {:2}".format(10))
print("Total: {:02}".format(10))
```

```
Total: 5
Total:  5
Total: 05
Total: 10
Total: 10
```

Brincando com *strings* — formatação

Em geral: **`{que:como}`**

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: **`A.BC`**

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)
- **C:** força formato: `f` (float) ou `e` (notação científica)

Brincando com *strings* — floats vs ints

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
```

```
Total: 5
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
```

```
Total: 5
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
```

```
Total: 5
Total: 5.0
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
print("Total: {:f}".format(5))
```

```
Total: 5
Total: 5.0
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
print("Total: {:f}".format(5))
```

```
Total: 5
Total: 5.0
Total: 5.0
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
print("Total: {:f}".format(5))
print("Total: {:.3f}".format(5))
```

```
Total: 5
Total: 5.0
Total: 5.0
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
print("Total: {:f}".format(5))
print("Total: {:.3f}".format(5))
```

```
Total: 5
Total: 5.0
Total: 5.0
Total: 5.000
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
print("Total: {:f}".format(5))
print("Total: {:.3f}".format(5))
print("Total: {:.3e}".format(5))
```

```
Total: 5
Total: 5.0
Total: 5.0
Total: 5.000
```

Brincando com *strings* — floats vs ints

```
print("Total: {}".format(5))
print("Total: {}".format(5.0))
print("Total: {:f}".format(5))
print("Total: {:.3f}".format(5))
print("Total: {:.3e}".format(5))
```

```
Total: 5
Total: 5.0
Total: 5.0
Total: 5.000
Total: 5.000e+00
```

Brincando com *strings* — formatação

Em geral: `{que:como}`

que: nome, número (índice) ou vazio (fica subentendido o índice)

como: `A.BC`

Brincando com *strings* — formatação

Em geral: `{que:como}`

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: `A.BC`

A: largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)

Brincando com *strings* — formatação

Em geral: **`{que:como}`**

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: **`A.BC`**

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)

Brincando com *strings* — formatação

Em geral: `{que:como}`

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: `A.BC`

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)
- **C:** força formato: `f` (float) ou `e` (notação científica)

Brincando com *strings* — formatação

Em geral: **`{que:como}`**

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: **`A.BC`**

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)
- **C:** força formato: `f` (float) ou `e` (notação científica)

como pode ser omitido (aí não precisa de "`:`") → `{0}`, `{blah}`

Brincando com *strings* — formatação

Em geral: **`{que:como}`**

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: **`A.BC`**

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)
- **C:** força formato: `f` (float) ou `e` (notação científica)

como pode ser omitido (aí não precisa de "`:`") → `{0}`, `{blah}`

que pode ser omitido → `{:05}`, `{:7.2f}`

Brincando com *strings* — formatação

Em geral: **`{que:como}`**

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: **`A.BC`**

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)
- **C:** força formato: `f` (float) ou `e` (notação científica)

como pode ser omitido (aí não precisa de "`:`") → `{0}`, `{blah}`

que pode ser omitido → `{:05}`, `{:7.2f}`

ambos podem ser omitidos (aí não precisa de "`:`") → `{}`

Brincando com *strings* — formatação

Em geral: **`{que:como}`**

que: **nome, número (índice) ou vazio (fica subentendido o índice)**

como: **`A.BC`**

- **A:** largura mínima; o que falta é preenchido por espaços (à esquerda para números, à direita para strings). Para números, pode ser precedido por zero → preenchimento com zeros (à esquerda)
- **B:** casas depois da vírgula (só para floats)
- **C:** força formato: `f` (float) ou `e` (notação científica)

como pode ser omitido (aí não precisa de "`:`") → `{0}`, `{blah}`

que pode ser omitido → `{:05}`, `{:7.2f}`

ambos podem ser omitidos (aí não precisa de "`:`") → `{}`

https://docs.python.org/3/library/string.html#formatspec

Brincando com *strings* — formatação

“São 14:16:02h”

```
horas = 14
minutos = 16
segundos = 2
```

Brincando com *strings* — formatação

“São 14:16:02h”

```
horas = 14
minutos = 16
segundos = 2
print("São {:02}:{:02}:{:02}h".format(horas, minutos, segundos))
```

Brincando com *strings* — formatação

```
a = 13.22784
b = 1200.20004
c = 13227.84
d = 37.6
e = 0.02
f = 127
print("|{:9.3f}| -- |{:9.3f}|".format(a, b))
print("|{:9.3f}| -- |{:9.3f}|".format(c, d))
print("|{:9.3f}| -- |{:9.3f}|".format(e, f))
```

Brincando com *strings* — formatação

```
a = 13.22784
b = 1200.20004
c = 13227.84
d = 37.6
e = 0.02
f = 127
print("|{:9.3f}| -- |{:9.3f}|".format(a, b))
print("|{:9.3f}| -- |{:9.3f}|".format(c, d))
print("|{:9.3f}| -- |{:9.3f}|".format(e, f))
```

```
|   13.228| -- | 1200.200|
|13227.840| -- |   37.600|
|    0.020| -- |  127.000|
```

strings vs `print()`

- escapes são recursos das *strings*
- `sep` e `end` são recursos de `print()`
- comandos de formatação (`.format()`) são recursos das *strings*

and now for something
(else) completely different

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")

if delta == 0:
    …
    print("A raiz dupla é", raiz)

if delta > 0:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")

if delta == 0:
    …
    print("A raiz dupla é", raiz)

if delta > 0:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

- **Tratamos todos os casos corretamente, mas não deixamos claro que os três casos são mutuamente excludentes (um e apenas um dos casos é executado)**

Condições mutuamente excludentes

É só usar **else**!

```
if delta < 0:
    print("não há raízes reais")

if delta == 0:
    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

Condições mutuamente excludentes

É só usar **else**!

```
if delta < 0:
    print("não há raízes reais")

if delta == 0:
    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

- **Oops!!!!!!**

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")
else:
    if delta == 0:
        …
        print("A raiz dupla é", raiz)
    else:
        …
        print("As raízes são {} e {}".format(raiz1, raiz2))
```

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")
else:
    if delta == 0:
        …
        print("A raiz dupla é", raiz)
    else:
        …
        print("As raízes são {} e {}".format(raiz1, raiz2))
```

- **Tratamos todos os casos corretamente, mas não deixamos claro que os três casos são mutuamente excludentes**
  - É um tanto estranho que `delta == 0` seja um "sub-caso" do **`else`** anterior

Condições mutuamente excludentes

```
if delta < 0:
    print("não há raízes reais")
else:
    if delta == 0:
        …
        print("A raiz dupla é", raiz)
    else:
        …
        print("As raízes são {} e {}".format(raiz1, raiz2))
```

- **Tratamos todos os casos corretamente, mas não deixamos claro que os três casos são mutuamente excludentes**
  - É um tanto estranho que `delta == 0` seja um "sub-caso" do **`else`** anterior
  - Parece que há algo de especial no caso `delta < 0`

Condições mutuamente excludentes — elif

```
if delta < 0:
    print("não há raízes reais")
else:
    if delta == 0:
        …
        print("A raiz dupla é", raiz)
    else:
        …
        print("As raízes são {} e {}".format(raiz1, raiz2))
```

Condições mutuamente excludentes — elif

```
if delta < 0:
    print("não há raízes reais")
elif delta == 0:

    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

Condições mutuamente excludentes — elif

```
if delta < 0:
    print("não há raízes reais")
elif delta == 0:


    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

- A indentação deixa mais claro que, na verdade, são casos do mesmo "nível" e mutuamente excludentes (**um** e **apenas um** dos casos é executado)

Condições mutuamente excludentes — elif

```
if delta < 0:
    print("não há raízes reais")
elif delta == 0:

    …
    print("A raiz dupla é", raiz)
else:
    …
    print("As raízes são {} e {}".format(raiz1, raiz2))
```

- **A indentação deixa mais claro que, na verdade, são casos do mesmo "nível" e mutuamente excludentes (um e apenas um dos casos é executado)**
  - Mas, na verdade, esse código equivale exatamente ao anterior!

Exercício

- **Dados x e y, diga se eles são iguais ou qual deles é maior**

Exercício

- **Dados x e y, diga se eles são iguais ou qual deles é maior**
  - Quantos casos possíveis?

Exercício

- **Dados x e y, diga se eles são iguais ou qual deles é maior**
  - Quantos casos possíveis?

```
if x < y:
    print("x é menor do que y.")
if x > y:
    print("x é maior do que y.")
if x == y:
    print("x e y são iguais.")
```

Exercício

- **Dados x e y, diga se eles são iguais ou qual deles é maior**
  - Quantos casos possíveis?
  - Os casos são mutuamente excludentes?

```
if x < y:
    print("x é menor do que y.")
if x > y:
    print("x é maior do que y.")
if x == y:
    print("x e y são iguais.")
```

Exercício

- **Dados x e y, diga se eles são iguais ou qual deles é maior**
  - Quantos casos possíveis?
  - Os casos são mutuamente excludentes?

```
if x < y:
    print("x é menor do que y.")
elif x > y:
    print("x é maior do que y.")
else:
    print("x e y são iguais.")
```

Atalhos

Atalhos

- É muito comum fazer `x = x +1`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`

Atalhos

- **É muito comum fazer `x = x +1`**
- **Tão comum que python oferece um atalho:**
  - `x += 1`
- **Não precisa ser 1:**
  - `x += 5`
  - `x += 2.3`
  - `x += -7`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?
  - `x -= 1`

Atalhos

- É muito comum fazer `x = x +1`
- Tão comum que python oferece um atalho:
  - `x += 1`
- Não precisa ser 1:
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- E já que fizemos para o "+", por que não outros operadores?
  - `x -= 1`
  - `x *= 2`

Atalhos

- **É muito comum fazer `x = x +1`**
- **Tão comum que python oferece um atalho:**
  - `x += 1`
- **Não precisa ser 1:**
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- **E já que fizemos para o "+", por que não outros operadores?**
  - `x -= 1`
  - `x *= 2`
  - `x /= 2`

Atalhos

- **É muito comum fazer `x = x +1`**
- **Tão comum que python oferece um atalho:**
  - `x += 1`
- **Não precisa ser 1:**
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- **E já que fizemos para o "+", por que não outros operadores?**
  - `x -= 1`
  - `x *= 2`
  - `x /= 2`
  - `x //= 2`

Atalhos

- **É muito comum fazer `x = x +1`**
- **Tão comum que python oferece um atalho:**
  - `x += 1`
- **Não precisa ser 1:**
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- **E já que fizemos para o "+", por que não outros operadores?**
  - `x -= 1`
  - `x *= 2`
  - `x /= 2`
  - `x //= 2`
  - `x %= 2`

Atalhos

- **É muito comum fazer `x = x +1`**
- **Tão comum que python oferece um atalho:**
  - `x += 1`
- **Não precisa ser 1:**
  - `x += 5`
  - `x += 2.3`
  - `x += -7`
- **E já que fizemos para o "+", por que não outros operadores?**
  - `x -= 1`
  - `x *= 2`
  - `x /= 2`
  - `x //= 2`
  - `x %= 2`
  - `x **= 2`

Atalhos

- Também é muito comum fazer coisas como

```
a = 5
b = 10
nome = "Fulano"
```

Atalhos

- Também é muito comum fazer coisas como

```
a = 5
b = 10
nome = "Fulano"
```

- Python oferece um atalho para isso também:

Atalhos

- Também é muito comum fazer coisas como

```
a = 5
b = 10
nome = "Fulano"
```

- Python oferece um atalho para isso também:

```
a, b, c = 5, 10, "Fulano"
```

and now for (yet) something (else) completely different

Álgebra booleana

Propriedades Comutativas	A and B = B and A	A or B = B or A
Propriedades Distributivas	A and (B or C) = (A and B) or (A and C)	A or (B and C) = (A or B) and (A or C)
Propriedades Associativas	(A or B) or C = A or (B or C)	(A and B) and C = A and (B and C)
Propriedades Idempotentes	A and A = A	A or A = A
Dupla Negação	not not A = A	
Elementos Absorventes	A or True = True	A and False = False
Elementos Neutros	A or False = A	A and True = A
Leis de De Morgan	not (A or B) = (not A) and (not B)	not (A and B) = (not A) or (not B)

Álgebra booleana

- pizza
- sushi
- moqueca
- hambúrguer

Álgebra booleana

- **pizza**
- **sushi**
- **moqueca**
- **hambúrguer**

  - pizza **ou** hambúrguer

Álgebra booleana

- **pizza**
- **sushi**

- **moqueca**
- **hambúrguer**

- pizza **ou** hambúrguer

- **nem** sushi **nem** moqueca

Álgebra booleana

- **pizza**
- **sushi**

- pizza **ou** hambúrguer

- **moqueca**
- **hambúrguer**

- **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

Álgebra booleana

- **pizza**
- **sushi**

- pizza **ou** hambúrguer

- **moqueca**
- **hambúrguer**

- **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **pizza**
- **sushi**
- **moqueca**
- **hambúrguer**

- **Sou guloso**
  - pizza **ou** hambúrguer
  - **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **pizza**
- **sushi**
- **moqueca**
- **hambúrguer**

- **Sou guloso**
  - pizza **ou** hambúrguer
- **Sou alérgico a peixes**
  - **nem** sushi **nem** moqueca

Equivalentes! Qual usar?

O que facilita o entendimento

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `not sushi and not moqueca`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for sushi ou moqueca**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)
- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)
- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`
- **Topa lição de casa, jantar e depois cinema?**

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca — `(not sushi) and (not moqueca)`**
- **não quero se for (sushi ou moqueca) — `not (sushi or moqueca)`**

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - `not (jantar and cinema)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - `not (jantar and cinema)`
  - Tem que abrir mão de (pelo menos) um deles

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - » `not (jantar and cinema)`
  - Tem que abrir mão de (pelo menos) um deles
    - » `(not jantar) or (not cinema)`

Álgebra booleana

- **Leis de De Morgan:**
  - not (A or B) = (not A) and (not B)
  - not (A and B) = (not A) or (not B)

- **nem sushi nem moqueca** — `(not sushi) and (not moqueca)`
- **não quero se for (sushi ou moqueca)** — `not (sushi or moqueca)`

- **Topa lição de casa, jantar e depois cinema?**
  - Não tenho grana para jantar e cinema!
    - `not (jantar and cinema)`
  - Tem que abrir mão de (pelo menos) um deles
    - `(not jantar) or (not cinema)`
    - `lição and ((not jantar) or (not cinema))`

Exercício

Leia duas datas fornecidas pelo usuário (para cada data você deve ler três valores: dia, mês e ano) e informe qual dessas datas é a mais recente

Exercício

Leia duas datas fornecidas pelo usuário (para cada data você deve ler três valores: dia, mês e ano) e informe qual dessas datas é a mais recente

```
dia1 = int(input("Digite o primeiro dia: "))
mês1 = int(input("Digite o primeiro mês: "))
ano1 = int(input("Digite o primeiro ano: "))
dia2 = int(input("Digite o segundo dia: "))
mês2 = int(input("Digite o segundo mês: "))
ano2 = int(input("Digite o segundo ano: "))
…
```

Exercício

```
…
if ano1 == ano2:
    if mês1 == mês2:
        if dia1 == dia2:
            print("As datas são iguais")
        elif dia1 > dia2:
            print("A primeira data é mais recente")
        else:
            print("A segunda data é mais recente")
    elif mês1 > mês2:
        print("A primeira data é mais recente")
    else:
        print("A segunda data é mais recente")
elif ano1 > ano2:
    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2) or (ano1 == ano2 and mês1 > mês2) or (ano1 == ano2 and mês1 == mês2 and dia1 > dia2):
    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2) or (ano1 == ano2 and mês1 > mês2) or (ano1 == ano2 and mês1 == mês2 and dia1 > dia2):
    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Propriedade distributiva: (A and B) or (A and C) = A and (B or C)

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2) or (ano1 == ano2 and mês1 > mês2) or (ano1 == ano2 and mês1 == mês2 and dia1 > dia2):
    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Propriedade distributiva: (A and B) or (A and C) = A and (B or C)

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2) or (ano1 == ano2 and mês1 > mês2) or (ano1 == ano2 and mês1 == mês2 and dia1 > dia2):
    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Propriedade distributiva: (A and B) or (A and C) = A and (B or C)

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2) or (ano1 == ano2 and (mês1 > mês2 or (mês1 == mês2 and dia1 > dia2))):
    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2)
     or (ano1 == ano2 and mês1 > mês2)
     or (ano1 == ano2 and mês1 == mês2 and dia1 > dia2):

    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif (ano1 > ano2)
     or (ano1 == ano2 and (mês1 > mês2 or (mês1 == mês2 and dia1 > dia2))):

    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif  ano1 > ano2
     or  ano1 == ano2 and mês1 > mês2
     or  ano1 == ano2 and mês1 == mês2 and dia1 > dia2 :

    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif  ano1 > ano2
     or  ano1 == ano2 and (mês1 > mês2 or  mês1 == mês2 and dia1 > dia2)  :

    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

Exercício

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif  ano1 > ano2
     or  ano1 == ano2 and mês1 > mês2
     or  ano1 == ano2 and mês1 == mês2 and dia1 > dia2 :

    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```

```
…
if ano1 == ano2 and mês1 == mês2 and dia1 == dia2:
    print("As datas são iguais")
elif  ano1 > ano2
     or  ano1 == ano2 and (mês1 > mês2 or  mês1 == mês2 and dia1 > dia2)  :

    print("A primeira data é mais recente")
else:
    print("A segunda data é mais recente")
```