Pensamento Computacional

2022/23

Aula 4

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Objetivos

- Conceitos Básicos de Programação:
  - Funções.
  - Strings e funções matemáticas.
  - Expressões lógicas e instruções condicionais.

Funções

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- **Definição:**
  - Sequência nomeada de declarações que executam uma tarefa ou procedimento;
  - Aceita zero ou mais argumentos;
  - Retorna um resultado e/ou produz algum efeito (tal como apresentar ou guardar o resultado de um cálculo).

- É possível definir uma função (próximas aulas).

Funções

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- Chamar (invocar) uma função pelo nome:

```
>>> rect_area(2, 4, "m")
8 m
```

  - Nome da função: `rect_area`
  - argumentos (3): `(2, 4, "m")`
  - resultado: `8 m`

Strings – relembrar!

- *Strings* são sequências de carateres.
- Literais string são delimitados por aspas simples ou duplas..

```
fruit = 'orange'
```

- Função que retorna o número de carateres:

```
len(fruit)                    #-> 6
```

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Strings – relembrar!

- Podemos aceder a um conjunto de carateres através da sua posição (**a primeira posição tem indíce 0**)

```
fruit[1]                    #-> 'r'
fruit[3:5]                  #-> 'ng'
```

- Podemos concatenar e repetir *strings*

```
name = 'tom' + 'cat'
#-> 'tomcat'
gps = 2 * 'tom'
#-> 'tomtom'
```

Strings

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- As *strings* são **imutáveis**. Os métodos que implicam alteração apenas retornam um novo objeto *string*.

```
fruit.upper()             #-> 'ORANGE'
fruit.replace('a', 'A') #-> 'orAnge'
fruit                     #-> 'orange' (not changed)
```

- A classe `str` tem vários métodos já definidos: para ver todos, usar `help(str)`

String methods

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

- As *strings* têm muitos métodos úteis:

Play ▶

`str.isalpha()`
`str.isdigit()`
`str.is...`
	True se todos os carateres são alfabéticos.
	True se todos os carateres são digitos.
	...

`str.upper()`
`str.lower()`
`...`
	Converte para maiúsculas.
	Converte para minúsculas.
	...

`str.strip()`
`str.lstrip()`
`str.rstrip()`
	Remove o espaço branco à esquerda e à direita.
	Remove o espaço branco à esquerda.
	Remove o espaço branco à direita.

`s1.find(s2)`	Encontra a string s2 na string s1
`str.split()` `str.split(sep)`	Divide str pelos carateres espaço branco. Divide a str usando sep como delimitador.

Carateres Especiais

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- A barra invertida (\) é usada para introduzir um carater especial:

```
# Introduces new line (\n)
>>> print ("Liliana Ferreira,\nFEUP")
Liliana Ferreira,
FEUP

# Introduces tab (\t)
>>> print("Liliana Ferreira,\tlsferreira@fe.up.pt");
Liliana Ferreira,   lsferreira@fe.up.pt
```

Escape sequence	Meaning
\\	Backslash
\'	Single quote
\"	Double quote
\n	Newline
\t	Tab

Verificar o conteúdo de uma *string*

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- **in** keyword
  - Verifica se uma dada frase ou carater está presente na string.
  - Devolve True ou False
  - Pode ser usado numa declaração *if* (próximas aulas)

```
>>> print ('city' in S)
True
# Check if NOT
>>> print ('city' not in S)
False
```

Método str.format()

- Permite substituições de variáveis e formatação de valores:

```
<template>.format(<positional_argument(s)>, <keyword_argument(s)>)
```

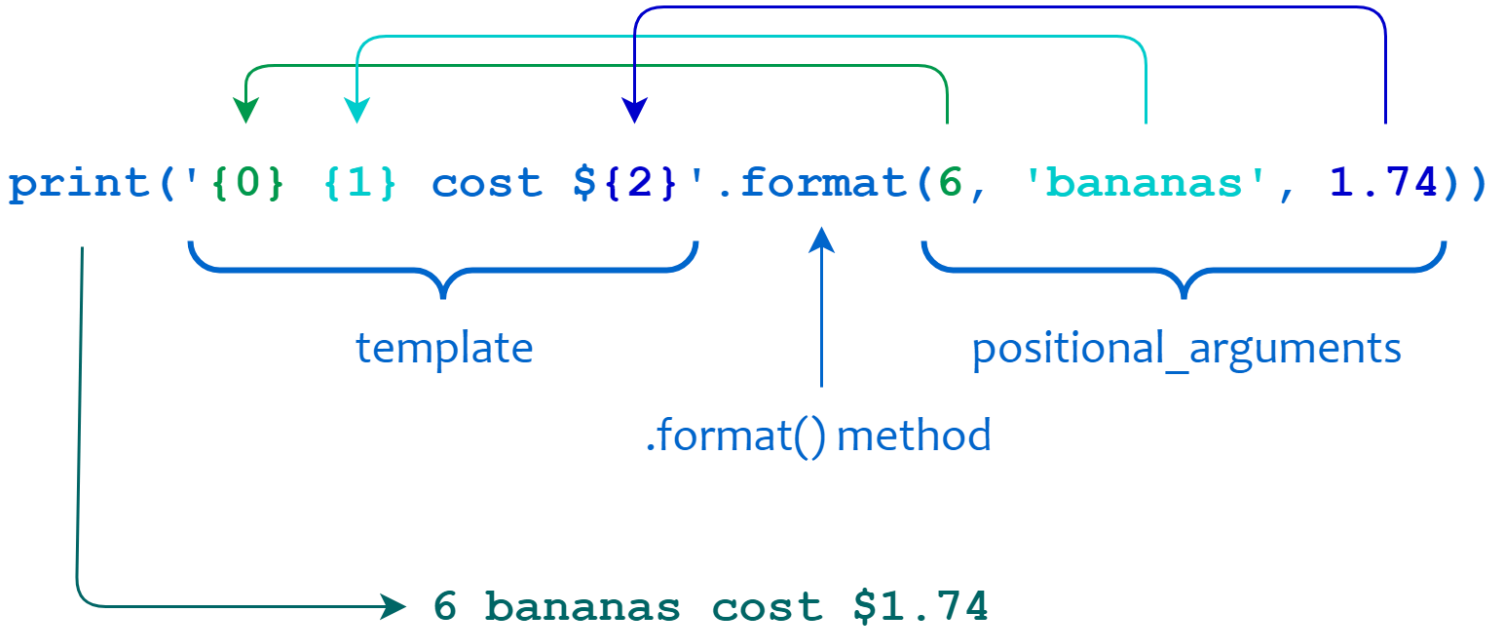
  - As strings de formatação contêm “campos de substituição” delimitados por {}.
  - Tudo o que não estiver dentro das chavetas é considerado texto literal.

Método str.format()

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- Exemplo

```
>>> print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))
6 bananas cost $1.74
```



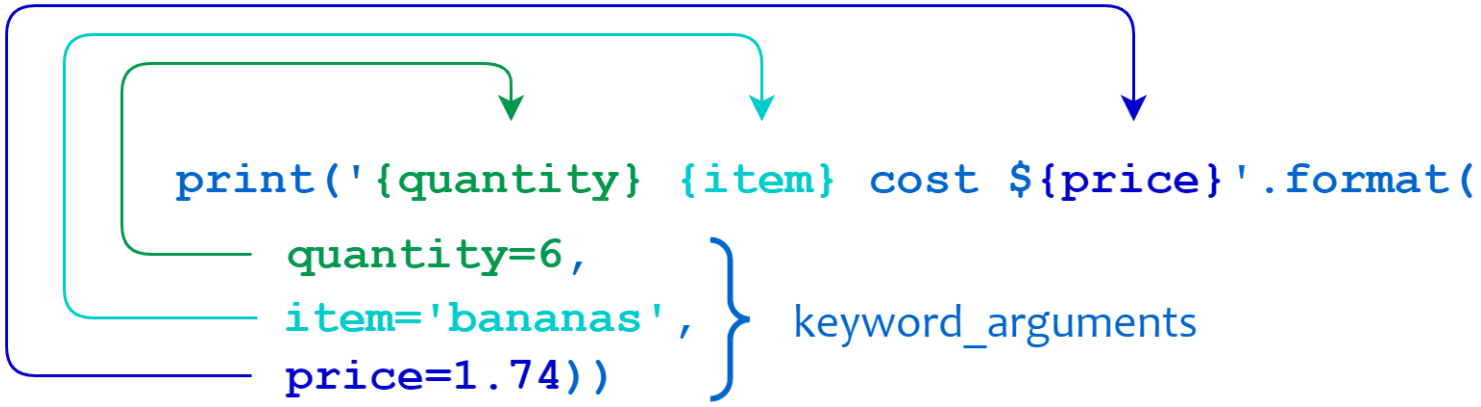
Método str.format() (cont.)

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- Usando keywords em vez de parâmetros de posicionamento

- Exemplo

```
>>> print('{quantity} {item} cost ${price}'.format(quantity=6,
item='bananas', price=1.74))
6 bananas cost $1.74
```



Formatação de *strings*: str.format()

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

```
  '{} {}'.format('one', 'two’)          #-> 'one two'
  '{:*>10}'.format('test')              #-> '******test'
  '{:04d}'.format(42)                   #-> '0042'
  '{:5.2f}'.format(3.2451)              #-> ' 3.25'

print(1,2,3)                            #-> 1 2 3
print(10,20,30)                         #-> 10 20 30

print('{:4d}{:4d}{:4d}'.format(1,2,3))        #->    1   2   3
print('{:4d}{:4d}{:4d}'.format(10,20,30))     #->   10  20  30
```

f-String

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- Nova e melhorada forma de formatar *strings*.

- Sintaxe é semelhante à de str.format() mas menos verbosa

  f”...”, F”...”, f’...’, F’...’

- Exemplo

```
>>> name = “Maria"
>>> age = 39
>>> f"Hello, {name}. You are {age}."
'Hello, Maria. You are 39.'
```

String Formatting: f-strings

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

```
f"{'one'} {'two'}"                  #-> 'one two'
f"{'test':*>10}"                    #-> '******test'
f'{42:04d}'                         #-> '0042'
f'{3.2451:5.2f}'                    #-> ' 3.25'


print(1,2,3)                        #-> 1 2 3
print(10,20,30)                     #-> 10 20 30

print(f'{1:4d}{2:4d}{3:4d}')        #->    1   2   3
print(f'{10:4d}{20:4d}{30:4d}')     #->   10  20  30
```

Funções Matemáticas

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- *Python* tem um módulo **math** que implementa as funções matemáticas mais conhecidas.
- Existe também um modulo **cmath** equivalente para trabalhar com números complexos.

- Um modulo – *module* – é um ficheiro Python que definie a coleção de funções e objetos relacionados.

- Antes de usar um dado módulo, devemos importá-lo:

```
>>> import math
>>> import cmath
```

Algumas funções matemáticas

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

math.	**Description**
pi	An approximation of pi.
e	An approximation of e.
sqrt(x)	The square root of x.
sin(x)	The sine of x.
cos(x)	The cosine of x.
tan(x)	The tangent of x.
asin(x)	The inverse of sine x.
acos(x)	The inverse of cosine x.
atan(x)	The inverse of tangent x.
log(x)	The natural (base e) logarithm of x.
log10(x)	The common (base 10) logarithm of x.
exp(x)	The exponential of x.
ceil(x)	The smallest whole number>= x.
floor(x)	The largest whole number <= x.
degrees(x)	Convert angle x from radians to degrees.
radians(x)	Convert angle x from degrees to radians.
help(math)	**Shows all math functions in the console**

Funções matemáticas - exemplos

```
>>> degrees = 45

>>> radians = degrees * math.pi / 180

# Angle arguments are always in radians!
>>> math.sin(radians)
0.707106781187

>>> cmath.sqrt(-1)
1j
```

Para usar as funções, especificar o nome do módulo e o nome da função, separados com um ponto.

Tópicos

- Expressões booleanas
  - Tipo bool;
  - Operadores relacionais;
  - Operadores lógicos;
  - Propriedades.
- Execução condicional
  - If statement
  - If-else
  - If-elif-else
- Expressões condicionais

Expressões booleanas

- Uma **expressão booleana** é uma expressão que ou é Verdadeira (True) ou é Falsa (False).

```
>>> n = 5       # this IS NOT a boolean expression!
>>> n == 5      # this IS a boolean expression!
True
>>> 6 == n      # this is another boolean expression.
False
```

- `True` e `False` são valores especiais que pertencem ao tipo `bool`.

Expressões booleanas

- Valores booleanos podem ser guardados em variáveis:

```
>>> isEven = n%2==0
```

- Podem ser convertidos para string:

```
>>> str(isEven)
'False'
```

- Ou inteiros:

```
>>> int(False)   # 0
>>> int(True)    # 1
```

Expressões booleanas

Valores nulos ou vazios convertem para `False`:

```
>>> bool(0)          # False
>>> bool(0.0)        # False
>>> bool('')         # False
>>> bool([])         # False
```

Outros valores convertem para `True`:

```
>>> bool(1)          # True
>>> bool('False')    # True (surprise!)
>>> bool([False])    # True (surprise?)
```

Operadores relacionais e lógicos

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- **Operadores relacionais** produzem resultados booleanos:

```
x == y       # x is equal to y
x != y       # x is not equal to y
x > y        # x is greater than y
x < y        # x is less than y
x >= y       # x is greater than or equal to y
x <= y       # x is less than or equal to y
x < y < z    # x is less than y and y is less than z (cool!)
```

Operadores relacionais e lógicos

- Há três **operadores lógicos**: `and`, `or`, `not`.

```
x>=0 and x<10     # x is between 0 and 10 (exclusive)

0<=x and x<10     # same thing

x==0 or not isEven and y/x>1
```

Propriedades

- Lembrem-se destas propriedades:

```
x == y   <=>   not (x != y)   <=>   y == x
x != y   <=>   not (x == y)   <=>   y != x
x > y    <=>   not (x <= y)   <=>   y < x
x <= y   <=>   not (x > y)    <=>   y >= x
```

- E destas (onde A, B, C são booleanos):

```
not (not A)        <=>   A
not (A and B)      <=>   (not A) or (not B)
not (A or B)       <=>   (not A) and (not B)
A or B             <=>   B or A
A and B            <=>   B and A
A or (B and C)     <=>   (A or B) and (A or C)
A and (B or C)     <=>   (A and B) or (A and C)
```

Regras de precedência

- Aritméticas> relacionais > `not` > `and` > `or`.

```
x<=1+2*y**3 or n!=0 and not 1/n<=y
(x<=1+2*y**3) or (n!=0 and not 1/n<=y)
(x<=(1+2*y**3)) or ((n!=0) and (not 1/n<=y))
(x<=(1+(2*y**3))) or ((n!=0) and (not (1/n<=y)))
(x<=(1+(2*(y**3)))) or ((n!=0) and (not ((1/n)<=y)))
```

Exemplos

```
23 < 4                                  #->  False
5 < 10 < 20                             #->  True
5 == 3 + 2                              #->  True
15 > 5*2 and 10 - 2 == 9                #->  False
```

- Os operadores relacionais também funcionam com strings.

```
'sky' > 'clouds'                        #-> True
'Sky' > 'clouds'                        #-> False
'8' > '5'                               #-> True
'123' > '4'                             #-> False
```

Avaliação curto-circuito (short-circuit)

- Os operadores `and` e `or` só avaliam o 2º operando se necessário!

```
A and B    # if A is false then A, otherwise B
A or B     # if A is true then A, otherwise B
```

- A isto chama-se de **avaliação curto-circuito**.
- Pode ser muito útil:

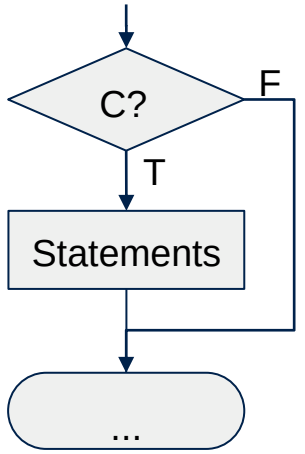
```
1/n>2 and n!=0    # ZeroDivisionError if n==0
n!=0 and 1/n>2    # False if n==0, 1/n not evaluated
n==0 or 3/n<4     # True if n==0, 3/n not evaluated
```

Note que a ordem dos operandos é importante!

Execução condicional

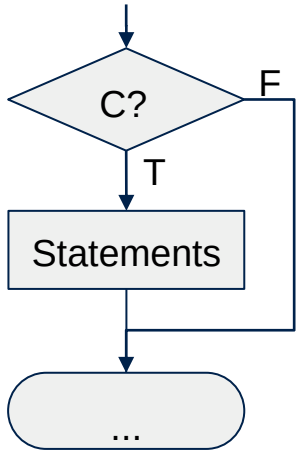
- **As expressões condicionais** permitem que o programa verifique certas condições e altere o seu comportamento de adequadamente.
- A sua forma mais simples é a expressão `if` :

```
if condition:
    suite_of_statements
...
```



Execução condicional

```
if condition:
    suite_of_statements
...
```



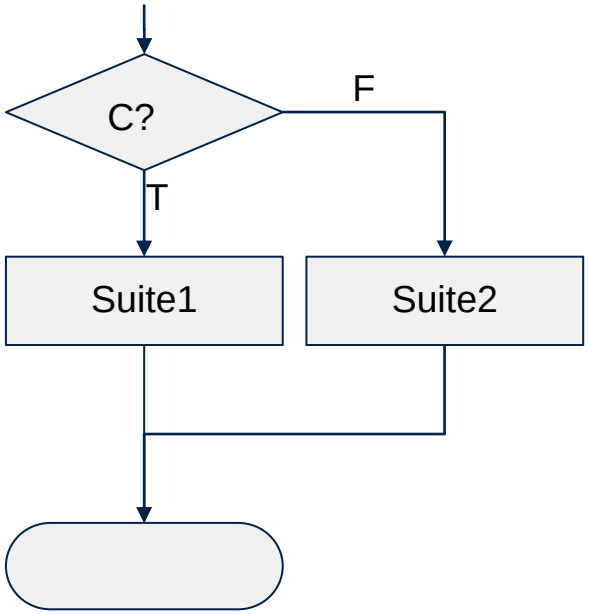
- A *condition* deve ser uma expressão booleana.
- A *suite of statements* é executada se a condição for verdadeira. Se não, a execução continua após estas estas declarações (*statements*).
- O conjunto de declarações deve ter uma ou mais declarações *indentadas*.

Execução condicional

- Um outro formato da declaração `if` é a de execução alternativa, na qual existem dois caminhos alternativos e a condição determina qual é que é executado.

```
if x%2 == 0:
    print('x is even')
else:
    print('x is odd')

#END
```

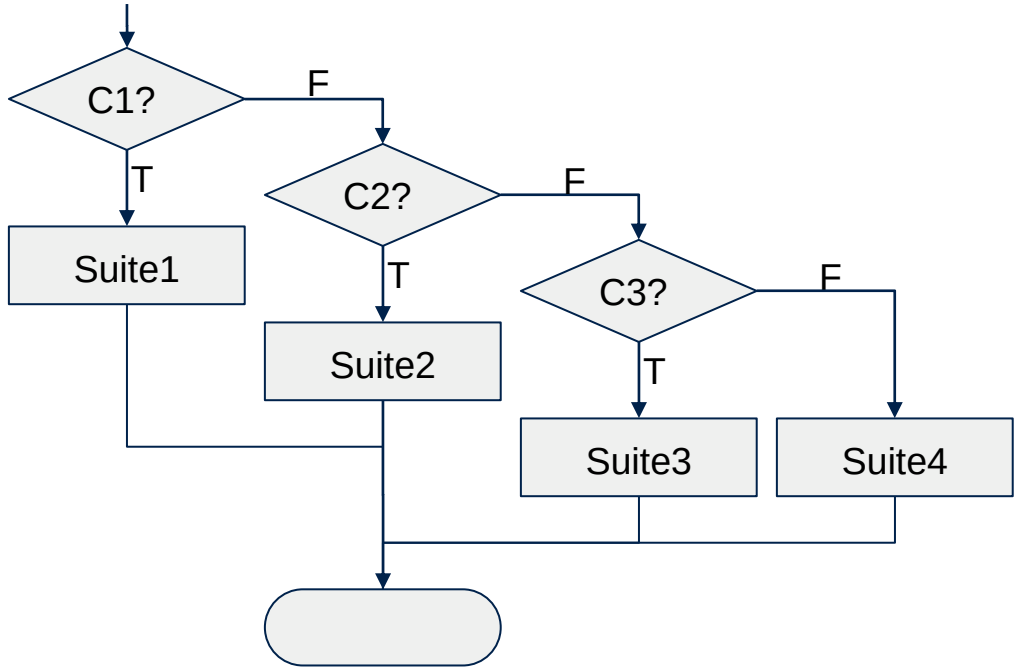


Execução condicional

- Por vezes, existem mais do que dois caminhos alternativos. Neste caso, precisamos de adicionar mais ramos à expressão condicional.

```
if x < 10:
    mark = 'Poor'
elif x < 13:
    mark = 'Reasonable'
elif x < 17:
    mark = 'Good'
else:
    mark = 'Excelent'

print(mark)
```



Semântica das Expressões Condicionais

- Quais as condições que ativam cada conjunto de declarações?

```
if C1:
    Suite1          ← Suite1 is executed iff C1
elif C2:
    Suite2          ← Suite2 is executed iff ¬C1∧C2
elif C3:
    Suite3          ← Suite3 is executed iff ¬C1∧¬C2∧C3
else:
    Suite4          ← Suite4 is executed iff ¬C1∧¬C2∧¬C3

Rest                ← is always executed
```

Exemplos: if-elif-else

```
num = int(input('num? '))

if num < 10:
    print(num, 'is less than 10.')
elif num > 10:
    print(num, 'is greater than 10.')
else:
    print(num, 'is equal to 10!')
```

```
word = input('Word: ')

if word < 'banana':
    print(word, 'comes before banana.')
elif word > 'banana':
    print(word, 'comes after banana.')
else:
    print('Word is "banana"!')
```

Declarações condicionais combinadas *(nested)*

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

- As declarações condicionais podem ser combinadas entre si.

```
if y > 0:
    if x > 0:
        quadrant = 1
    else:
        quadrant = 2
else:
    if x < 0:
        quadrant = 3
    else:
        quadrant = 4
```

- Embora a indentação permita facilitar a criação e compreensão da estrutura, expressões combinadas alagadas podem tornar-se difíceis de ler! É importante encontrar formas de simplificação.

Expressão Condicional

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

- Python também inclui uma expressão condicional, baseada num operador ternário:

```
expression1 if condition else expression2
```

- Usa as *keywords* if e else, mas é uma *expressão*!
- A condição é a primeira a ser avaliada.
- Se *true*, então expression1 é avaliada e é o resultado.
- Se *false*, então expression2 é avaliada e é o resultado.

"

```
n = int(input("number? "))
msg = "odd" if n%2!=0 else "even"
print(n, "is", msg)
```

Pensamento Computacional

2022/23

TPC: exercícios propostos aula 4