Programação I
Aula 3 — Primeiros programas

Pedro Vasconcelos
DCC/FCUP

Nesta aula...

Tipos de números

Em *Python* distinguimos números inteiros e fracionários (*vírgula-flutuante*) associando-lhes tipos distintos.

	tipo	exemplos
inteiros	`int`	`42 -7`
vírgula-flutuante	`float`	`42.0 -7.0 -0.0254`

Tipos de números (cont.)

As operações aritméticas funcionam com ambos os tipos:

```
>>> 1+2
3
```
int + int => int

```
>>> 1.0+2.0
3.0
```
float + float => float

Também podemos misturar números inteiros e *floats* numa operação; o resultado será *float*:

```
>>> 1 + 2.5
3.5
```
int + float => float

Tipos de números (cont.)

Divisão entre inteiros dá um *float*:

```
>>> 17/5
3.4
```

Podemos obter o *quociente* e o *resto* da divisão inteira com os operadores `//` e `%`:

```
>>> 17//5          quociente da divisão inteira
3
>>> 17%5           resto da divisão inteira
2
```


Erros de arredondamento

Números inteiros podem ser representados de forma exata no computador.[1]

Números em vírgula-flutuante são aproximações finitas dos números reais:

```
>>> 8/3
2.6666666666666665
```

As operações sucessivas sobre estes números podem fazer acumular erros de arredondamento.

O controlo destes erros na computação é estudado em Análise Numérica.

[1] Apenas limitados pela memória disponível.

Erros de arredondamento

Um exemplo

Usando álgebra exacta:

$$\left(\frac{100}{3} - 33\right) \times 3 = 100 - 33 \times 3 = 1$$

Contudo, usando operações vírgula-flutuante obtemos resultados diferentes:

```
>>> (100.0/3.0 - 33.0) * 3.0
1.0000000000000007
>>> 100.0 - 33.0*3.0
1.0
```

O erro de arrendondamento foi

$$1.0000000000000007 - 1 \approx 7 \times 10^{-15}$$

Conversão automática entre tipos numéricos

$$\texttt{int} + \texttt{int} \Rightarrow \texttt{int}$$

$$\texttt{float} + \texttt{float} \Rightarrow \texttt{float}$$

$$\texttt{int} + \texttt{float} \Rightarrow \texttt{float}$$

$$\texttt{float} + \texttt{int} \Rightarrow \texttt{float}$$

Também com os operadores aritméticos $-$, $*$ e $**$.

A divisão (em *Python* 3) é um caso especial:

$$\texttt{int}/\texttt{int} \Rightarrow \texttt{float}$$

$$\texttt{int}//\texttt{int} \Rightarrow \texttt{int}$$

$$\texttt{int}\%\texttt{int} \Rightarrow \texttt{int}$$

Conversão explícita entre tipos

```
>>> int(2.71)
2

>>> round(2.71)
3

>>> float(-33)
-33.0
```

```
>>> str(-3.134)
'-3.134'

>>> float("3.14")
3.14

>>> float("trinta e três")
ValueError
```

Nota:

`round(...)` arrendonda ao inteiro mais próximo;

`int(...)` trunca a parte fracionária;

Cadeias de carateres

As cadeias de carateres são valores de tipo `str` (*string*).

Escrevemos o texto entre aspas simples ou duplas:

```
>>> "Olá mundo!"
'Olá mundo!'

>>> 'abracadabra'
'abracadabra'

>>> type('abracadabra')
<class 'str'>
```

Cadeias de carateres (cont.)

Podemos usar três aspas para introduzir cadeias de carateres com várias linhas.

```
>>> '''Bom dia!
--- Ola, mundo!'''
'Bom dia!\n--- Ola, mundo!'
```


Operações sobre cadeias de carateres

Concatenação `str + str ⇒ str`

Repetição `int * str ⇒ str`

```
>>> "Olá" + " " + "Mundo"
'Olá Mundo'

>>> 3*"Olá" + " Mundo!"
'OláOláOlá Mundo!

>>> 3*"Olá " + "Mundo!"
'Olá Olá Olá Mundo!
```

Variáveis

- Nomes para representar *quantidades* ou *propriedades* dum problema
- Começam com uma letra, seguido de letras, números ou sublinhado
- Podem ter letras com acentos
- Não podem ter espaços ou tabulações
- Não podem ser *palavras reservadas*:

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	yield
continue	except	global	lambda	raise	

Variáveis (cont.)

Exemplos de nomes válidos para variáveis:

```
nome  idade  Preço_Max  área2
```

Exemplos de nomes que não podemos usar:

```
76trombones  more$  lambda
```

Atribuições

Associa o valor de uma expressão a uma variável:

$$nome = expressão$$

```
>>> raio = 1
```

raio ⟶ 1

Atribuições (cont.)

Depois de definir uma variável, podemos usá-la em expressões seguintes:

```
>>> import math
>>> perimetro = 2*math.pi*raio
>>> perimetro
6.283185307179586
```

raio ⟶ 1
perimetro ⟶ 6.283185307179586

Atribuições (cont.)

Note que a atribuição é um comando, não é uma equação.

Exemplo:

```
>>> raio = 2
>>> perimetro
6.2831853071795862
```

`raio`	$\longrightarrow$	2
`perimetro`	$\longrightarrow$	6.2831853071795862

O `perimetro` não altera automaticamente por alterarmos o `raio`...

Atribuições (cont.)

Se quiseremos re-calcular o perímetro, voltamos a executar a atribuição:

```
>>> perimetro = 2*math.pi*raio
>>> perimetro
12.566370614359172
```

`raio`	$\longrightarrow$	2
`perimetro`	$\longrightarrow$	12.566370614359172

Ordem de atribuições

A ordem das atribuições é importante!

Exemplo: vamos anotar os valores de p e n após cada instrução.

Programa 1

`p = 1`	$p \rightarrow 1$
`n = 2`	$p \rightarrow 1 \quad n \rightarrow 2$
`p = p*n`	$p \rightarrow 2 \quad n \rightarrow 2$
`n = n+1`	$p \rightarrow 2 \quad n \rightarrow 3$

Final: $p \rightarrow 2 \quad n \rightarrow 3$

Programa 2

`p = 1`	$p \rightarrow 1$
`n = 2`	$p \rightarrow 1 \quad n \rightarrow 2$
`n = n+1`	$p \rightarrow 1 \quad n \rightarrow 3$
`p = p*n`	$p \rightarrow 3 \quad n \rightarrow 3$

Final: $p \rightarrow 3 \quad n \rightarrow 3$

Programas completos

perimetro.py

```
import math

raio = 2.5
perimetro = 2*math.pi*raio
```

Executa correctamente, mas não mostra quaisquer resultados.

Comandos para entrada e saída

`input()` lê uma cadeia de carateres do terminal (teclado);
`print(expr1, expr2, ...)` escreve resultados no terminal

Programa revisto

```
import math

print("Indique o valor do raio:")
raio = float(input())
perimetro = 2*math.pi*raio
print("Perimetro da circunferência:", perimetro)
```

Comentários

- Começam com o símbolo `#` e extendem até ao fim da linha
- Permitem incluir documentação para outros programadores
- Também úteis para o autor (para lembrar como funciona o programa)

Comentários (cont.)

```
# Calcular o perimetro de uma circunferência
# Pedro Vasconcelos, 2013

print("Indique o valor do raio:")
# ler uma cadeia de carateres e converte num número
raio = float(input())
perimetro = 2*math.pi*raio   # calcular o perímetro
# imprime o resultado
print("Perimetro da circunferência:", perimetro)
```