Programação I / Introdução à Programação

Capítulo 5, "Data types" (cadeias de caracteres, "strings")

João Pedro Pedroso

2024/2025

- Últimas aulas:
  - métodos de *strings*: `lower`, `upper`, `find`, `replace` ...
  - funções `ord()`, `chr()`
  - codificação ASCII, unicode
- Hoje:
  - Formatação de *strings*
    - por exemplo, para produzir output visualmente apelativo

Formatação de *strings*: `str.format(...)`

- Em Python as *strings* têm um método para formatação
- Exemplo:

```
>>> name = "John"
>>> a = "Hello, {}"
>>> a
'Hello, {}'
>>> a.format(name)
'Hello, John'
```

Formatação de *strings*: `str.format(...)`

Método `str.format(...)`

- a *string* na qual é chamado pode ter
  - texto literal (`"Hello, "`)
  - *campos de substituição* (`"{}"`)
- cada *campo de substituição*
  - é delimitado por chavetas `{}`
  - na formatação é substituido por argumentos passados a `format` (no exemplo, a *string* `"John"`)
- `format` devolve uma cópia da *string* com os campos substituídos por argumentos (no exemplo `"Hello, John"`)

Formatação de *strings*: múltiplos argumentos

- Podemos passar múltiplos argumentos a `format`:

```
>>> a = "{}, my dear {}!"
>>> a.format("Hello", "John")
'Hello, my dear John!'
```

- Podemos especificar qual a posição do argumento que queremos utilizar em cada campo:

```
>>> a = "{0}, my dear {1}, {0} {0} and {0}!"
>>> a.format("Hello", "John")
'Hello, my dear John, Hello Hello and Hello!'
```

Formatação de *strings*: argumentos com nome

```
>>> a = "{greeting}, my dear {name}!"
>>> a.format(name="John", greeting="Good morning")
'Good morning, my dear John!'
```

Formatação de *strings*: espaçamento, alinhamento

- Podemos especificar espaçamento e alinhamento na apresentação usando `{:}`

```
>>> "-----{:10}-----".format("X")
'-----X          -----'

>>> "-----{:>10}-----".format("X")
'-----          X-----'

>>> "-----{:^10}-----".format("X")
'-----     X     -----'

>>> "-----{:<10}-----".format("X")
'-----X          -----'
```

Formatação de *strings*: preenchimento de espaços

- Podemos especificar o carater usado para preencher espaços

```
>>> "-----{:=>10}-----".format("X")
'-----=========X-----'

>>> "-----{:=<10}-----".format("X")
'-----X=========-----'

>>> "-----{:=^10}-----".format("X")
'-----====X=====-----'
```

Checkpoint `format`

`"Hello {}, how are you?".format("John")` → resulta em
`"Hello John, how are you?"`

`"{:<5}".format(i)` *string* com valor `i` alinhado à esquerda em 5 "espaços"

`"{:^5}".format(i)` ... alinhado ao centro

`"{:>5}".format(i)` ... alinhado à direita

Formatação de *strings*: campos numéricos: inteiros

- Podemos especificar tipos de apresentação usando `{:}`
  - o tipo aparece a seguir a `:`

```
>>> x = 17
>>> "value of x is {}".format(x)
'value of x is 17'

>>> "value of x is {:d}".format(x)    # decimal (base 10)
'value of x is 17'

>>> "value of x is {:b}".format(x)    # binary (base 2)
'value of x is 10001'

>>> "value of x is {:c}".format(x)    # as a character (Unicode)
'value of x is !'
```

- Também se pode imprimir valores na base 8 (`{:o}`) e 16 (`{:x}` ou `{:X}`)

Formatação de *strings*: campos numéricos: vírgula flutuante

- `e` notação científica (com expoente)
- `f` notação fixa (*fixed point*)
- `g` notação mista (*general format*)

```
>>> w = 2.5
>>> x = 1.2345678901234567
>>> y = 0.00000000000001234567890123456 7
>>> z = 1234567890.1234567
>>> "values are: w={}, x={}, y={}, z={}".format(w,x,y,z)
'values are: w=2.5, x=1.2345678901234567, y=1.2345678901234567e-14, z=1234567890.12

>>> "values are: w={:e}, x={:e}, y={:e}, z={:e}".format(w,x,y,z)
'values are: w=2.500000e+00, x=1.234568e+00, y=1.234568e-14, z=1.234568e+09'

>>> "values are: w={:f}, x={:f}, y={:f}, z={:f}".format(w,x,y,z)
'values are: w=2.500000, x=1.234568, y=0.000000, z=1234567890.123457'

>>> "values are: w={:g}, x={:g}, y={:g}, z={:g}".format(w,x,y,z)
'values are: w=2.5, x=1.23457, y=1.23457e-14, z=1.23457e+09'
```

Formatação de *strings*: campos numéricos: vírgula flutuante

- Funcionamento da precisão (por omissão, 6 dígitos)
- Sintaxe: e.g., `"... {:X.Yg} ..."`
  - `X` → número mínimo de carateres usado (*field width*)
  - `Y` → precisão
    - `'{.5f}'` → 5 casas decimais

```
>>> x = 1.2345678901234567
>>> y = 0.00000000000001234567890123456 7
>>> z = 1234567890.1234567
>>> "values are: x={}, y={}, z={}".format(x,y,z)
'values are: x=1.2345678901234567, y=1.2345678901234567e-14, z=1234567890.1234567'
>>> "values are: x={:.5f}, y={:.5f}, z={:.5f}".format(x,y,z)
'values are: x=1.23457, y=0.00000, z=1234567890.12346'
>>> "values are: x={:.1f}, y={:.1f}, z={:.1f}".format(x,y,z)
'values are: x=1.2, y=0.0, z=1234567890.1'
>>> "values are: x={:15.1f}, y={:15.1f}, z={:15.1f}".format(x,y,z)
'values are: x=            1.2, y=            0.0, z=   1234567890.1'
>>> "values are: x={:.1e}, y={:.1e}, z={:.1e}".format(x,y,z)
'values are: x=1.2e+00, y=1.2e-14, z=1.2e+09'
```

Versões do Python recentes: `f-strings`

`f"...{i}..."` o mesmo que `"...{}...".format(i)`

```
1  alunos = [(123, "Jose", 14.5),
2            (13, "Maria", 17.5),
3            (32, "Manuela", 16.7)]
4
5  for (numero, nome, nota) in alunos:
6      print(f"{nome} teve {nota} valores no exame.")
```

```
Jose teve 14.5 valores no exame.
Maria teve 17.5 valores no exame.
Manuela teve 16.7 valores no exame.
```

Exemplo: Cálculo de juros

- Pedimos um empréstimo ao banco de 5000 euros
- Começamos a pagar no próximo ano, durante 12 anos
- Em cada ano pagamos capital mais juros
- Se a taxa de juro for de 2% ao ano, qual o valor a pagar em cada ano?

```
1  K = 5000
2  r = .02
3  N = 12
4  amort = K/N
5  print("Ano Divida Amort Juros")
6  for i in range(1,N+1):
7      K -= amort
8      print(i, K, amort, K*r)
```

Exemplo: Cálculo de juros

- Pedimos um empréstimo ao banco de 5000 euros
- Começamos a pagar no próximo ano, durante 12 anos
- Em cada ano pagamos capital mais juros
- Se a taxa de juro for de 2% ao ano, qual o valor a pagar em cada ano?

```
1  K = 5000
2  r = .02
3  N = 12
4  amort = K/N
5  print("Ano Divida Amort Juros")
6  for i in range(1,N+1):
7      K -= amort
8      print(i, K, amort, K*r)
```

- Output:

```
Ano Divida Amort Juros
1 4583.333333333333 416.6666666666667 91.66666666666666
2 4166.666666666666 416.6666666666667 83.33333333333333
3 3749.9999999999995 416.6666666666667 74.99999999999999
```

Cálculo de juros: output formatado

- Experimente: `f"{i:3}{K:10.3f}{amort:12.5e}{K*r:14.7g}"`
- Equivalente a:
  `"{:3}{:10.3f}{:12.5e}{:14.7g}".format(1,K,amort,K*r)`
- Como obter o seguinte output?

```
  Ano     Divida      Amort      Juros
    1    4583.33    416.67       91.67
    2    4166.67    416.67       83.33
...
   12      -0.00    416.67       -0.00
```

Comparação de *strings* para ordenação

- A comparação entre duas *strings* não é alfabética:

```
>>> "joao" == "Joao"
False
```

- Convertendo ambas as *strings* para maiúscula/minúscula:

```
>>> "joao".lower() == "Joao".lower()
True
```

- Mas ainda temos problemas com carateres acentuados:

```
>>> "joão" < "José"
False
>>> "joão".lower() < "José".lower()
False
```

Comparação de *strings* para ordenação

- Podemos resolver o problema usando o módulo `unidecode`

```
>>> from unidecode import unidecode
>>> unidecode("joão")
'joao'
>>> unidecode("José")
'Jose'
>>> unidecode("joão".lower()) < unidecode("José".lower())
True
```

Noções estudadas

- formatação de strings
- campos de substituição
- output de *strings*, inteiros e *floats*
- `f-strings`
- ordenação

Próximas aulas: listas e tuplos