String

- Tipo composto por carateres;
- Coleção de dados;
- Imutável: aos seus elementos não se pode atribuir novos valores

Percorrer *strings*

- Com ciclo `for` (*iteração por item*)

```
for achar in "Go Spot Go":
    print(achar)
```

- Com ciclo `for` (*iteração por índice*)

```
fruit = "apple"
for idx in range(len(fruit)):
    print(fruit[idx])
for idx in range(len(fruit)-1, -1, -1):
    print(fruit[idx])
```

- Com ciclo `while`

```
fruit = "apple"

position = 0
while position < len(fruit):
    print(fruit[position])
    position = position + 1
```


Operações e métodos com *strings*

- Experimentar exemplos do livro!
- Comparação: baseada na função `ord(s)`, que dá número de ordem (ASCII) de um carater.

```
ss = "    Hello, World    "

els = ss.count("l")
print(els)

print("***"+ss.strip()+"***")
print("***"+ss.lstrip()+"***")
print("***"+ss.rstrip()+"***")

news = ss.replace("o", "***")
print(news)
```

```
print("apple" < "banana")
print("apple" == "Apple")
print("apple" < "Apple")
```


Nesta aula...

Tabelas

- Antes da existência de computadores: tabelas com logaritmos, senos, cossenos, ...
- Para algumas operações, os computadores também usam tabelas, para aproximar valores (bug do Pentium: valor mal tabelado).

```
print("n",'\t',"2**n")
print("---",'\t',"-----")

for x in range(13):
    print(x, '\t', 2**x)
```

```
n       2**n
---     -----
0       1
1       2
2       4
3       8
4       16
5       32
6       64
7       128
8       256
9       512
10      1024
11      2048
12      4096
```


Sequências de escape

- numa *string* a barra invertida (`\`) marca o início de uma *sequência de escape*
- utilizadas para representar carateres especiais:
  - tabulações: `\t`
  - mudanças de linha `\n`
  - testar: `print("\\")`
- depois de um print, o cursor normalmente vai para o princípio da linha seguinte;
- exceção: utilizando parâmetro `end=""`:

```
print("hello, world",end="")
print("hello, world",end="")
print("hello, world")
```


Carateres especiais

- Tabulação: `\t`
- Mudança de linha: `\n`
- Apóstrofo: `"'"`
- Aspas: `'"'`
- String com mais de uma linha:

```
"""start here

some " and ' inside

end here"""
```


Operador de formatação

- Sintaxe: `format % values`
  - `format` – *string* com `%`
  - operador `%` – módulo, ou formatação de *strings*
  - `values` – valores que serão substituídos nos `%` da *string* de formatação
- Exemplo:

```
>>> "mês %d é %s" % (2,"Fevereiro")
'mês 2 é Fevereiro'
>>>
```

Operador de formatação: exemplos

d, i inteiro decimal com sinal
```
"'%d'___'%3d'___'%-3d'" % (5, 5, 5)
'5'___'  5'___'5  '
```

o, x, X inteiro em formato octal/hexadecimal
```
"%o %x %X" % (15, 15, 15)
17 f F
```

e, E, f, F, g, G virgula (ou ponto) flutuante, formato exponencial ou decimal
```
"%8.1e %E %f %g %G" % (15, 15, 15, 15, 1500000)
 1.5e+01 1.500000E+01 15.000000 15 1.5E+06
```

s string
```
"___%s___%10s___%-10s___" % ("abc", "abc", "abc")
___abc___       abc___abc       ___
```

r representação interna do Python
```
"%r %r %r" % (15, "abc", math.sin)
15 'abc' <built-in function sin>
```

%% o caracter % `"%d%% da nota" % 12`
```
12% da nota
```


Operadores `in` e `not in`

- testam se uma *string* é *substring* de outra
- uma *string* é substring de si própria;
- a *string* vazia é substring de *qualquer string*;
- o operador `not in` resulta na negação lógica de `in`.

```
>>> 'p' in 'apple'
True
>>> 'i' in 'apple'
False
>>> 'ap' in 'apple'
True
>>> 'pa' in 'apple'
False
>>> 'apple' in 'apple'
True
>>> '' in 'a'
True
>>> 'apple' in 'appl'
False
>>> 'apple' not in 'appl'
True
```


Padrão *acumulador* com *strings*

```
def removeVowels(s):
    vowels = "aeiouAEIOU"
    sWithoutVowels = ""
    for eachChar in s:
        if eachChar not in vowels:
            sWithoutVowels = sWithoutVowels + eachChar
    return sWithoutVowels

print(removeVowels("compsci"))
print(removeVowels("aAbEefIijOopUus"))
```

Alternativa à linha 5:

```
if eachChar != 'a'  and eachChar != 'e'  and eachChar != 'i'  and
   eachChar != 'o'  and eachChar != 'u'  and eachChar != 'A'  and
   eachChar != 'E'  and eachChar != 'I'  and eachChar != 'O'  and
   eachChar != 'U':

    sWithoutVowels = sWithoutVowels + eachChar
```


Padrão *acumulador* com *strings*

```
def count(text, aChar):
    lettercount = 0
    for c in text:
        if c == aChar:
            lettercount = lettercount + 1
    return lettercount

print(count("banana","a"))
```


Procurar em strings 1

```
def find(astring, achar):
    ix = 0
    found = False
    while ix < len(astring) and not found:
        if astring[ix] == achar:
            found = True
        else:
            ix = ix + 1
    if found:
        return ix
    else:
        return -1
```

- Utilização: `print(find("Compsci", "p"));`
  `print(find("Compsci", "x"));`
- `find` é o "oposto" da indexação;
- Contrução do ciclo `while`: utiliza variável lógica para indicar se já foi encontrado o carater ou não (há construções mais eficientes; esta foi usada para ilustração);
- Padrão *eureka*: quando encontra, termina.

Procurar em strings 2

```
def find2(astring, achar, start):
    ix = start
    found = False
    while ix < len(astring) and not found:
        if astring[ix] == achar:
            found = True
        else:
            ix = ix + 1
    if found:
        return ix
    else:
        return -1
```

- Fornece-se índice onde a procura deve começar;
- Utilização: `print(find2('banana', 'a', 2))`.

Procurar em strings 3

```
def find3(astring, achar, start=0):
    ix = start
    found = False
    while ix < len(astring) and not found:
        if astring[ix] == achar:
            found = True
        else:
            ix = ix + 1
    if found:
        return ix
    else:
        return -1
```

- Fornece-se índice **opcional**: onde a procura começa;
- Utilização: `print(find3('banana', 'a', 2))`.

Procurar em strings 4

```
def find4(astring, achar, start=0, end=None):
    ix = start
    if end == None:
        end = len(astring)
    found = False
    while ix < end and not found:
        if astring[ix] == achar:
            found = True
        else:
            ix = ix + 1
    if found:
        return ix
    else:
        return -1
```

- Fornece-se índices opcionais: onde a procura começa e acaba (contante `None`);
- Semântica semelhante à de `range`; utilização:

```
ss = "Python strings have some interesting methods."
print(find4(ss, 's'))
print(find4(ss, 's', 7))
print(find4(ss, 's', 8, 10))
```

Módulo `string`

```
>>> import string
>>> print(string.digits)
0123456789
>>> print(string.ascii_uppercase)
ABCDEFGHIJKLMNOPQRSTUVWXYZ
>>> print(string.ascii_lowercase)
abcdefghijklmnopqrstuvwxyz
>>> print(string.whitespace)
>>> for c in string.whitespace:
...     print(ord(c))
...
32
9
10
13
11
12
```

- Mais informação: módulo `string` em `www.python.org`.

Noções estudadas esta semana

indexação: `a[i]` acede ao carater na posição `i` da string `a` (primeiro carater: posição 0)

função `len` retorna o número de carateres numa `string`

fatias `[:]` (*slicing*) *substring* de uma dada *string*; mais geralmente, uma subsequência de uma sequência pode ser acedido com `sequence[start:stop]`

comparação de *strings*: `> < >= <= == !=` comparam as *strings* ASCII, de forma lexicográfica através da função `ord` aplicada a cada um dos carateres (ordenação com carateres portugueses: a ver mais tarde)

`in`, `not in` operadores que testam se uma *string* está contida noutra

coleção tipo de dados em que um valor é composto por elementos, que também são valores

Noções estudadas esta semana

valor por omissão (*default value*) valor atribuído a um parâmetro opcional, se o respetivo argumento não for fornecido na chamada à função

cursor marcador invisível, que posiciona o próximo carater a ser impresso

dot notation utilização do ponto (`.`) para aceder funções definidas num módulo, ou métodos e atributos de um objeto

imutável tipo de dados composto, a cujos elementos não se pode atribuir novos valores

índice valor utilizado para selecionar um membro de uma coleção ordenada (e.g., carater numa *string*, elemento de uma lista)

parâmetro opcional parâmetro ao qual se atribuí um valor por omissão no cabeçalho de uma função, que é utilizado caso o respetivo argumento não seja fornecido na chamada à função

sequência coleção ordenada de dados

percorrer iterar através de todos os elementos de uma coleção

whitespace carateres que não se vêm quando são impressos

Noções estudadas esta semana

reatribuição durante a execução de um programa, fazer mais do que uma atribuição de valor à mesma variável

sequência de escape o carater de escape `\` seguido de uma (ou mais) letras, para designar um carater especial, que não é uma letra

newline `\n` sequência de escape que significa uma mudança de linha

tab `\t` sequência de escape que significa uma tabulação

Próximas aulas

- Listas.
- Outras estruturas de dados em *Python*.