Fundamentos de Programação de Computadores

Python- condições, ciclos, strings

Docente: Fátima Leal

DCT DEPARTAMENTO CIÊNCIA E TECNOLOGIA

UPT UNIVERSIDADE PORTUCALENSE

Python

- Na última aulas vimos:
- **Variáveis**
- **Expressões**
- **Funções**

Python – Operadores de comparação

- Como experimentamos no pseudocódigo, nas condições necessitamos de operadores de comparação.

==	igual
!=	diferente
>	maior
<	menor
>=	maior ou igual
<=	menor ou igual

```
>>> 4+2 == 6
True
>>>
>>> 3!=4
True
>>>
```

- No Python, estes operadores são utilizados de forma semelhante.

Python – Operadores de comparação

- Adicionalmente, por vezes necessitamos de agregar condições utilizando operadores lógicos.

operador	o que se verifica
`and`	se ambos os operandos são verdadeiros
`or`	se pelo menos um dos operandos é verdadeiro
`not`	se o operando é falso

```
>>> 4+2==6 and 3!=4
True
```

```
>>> 4+3==6 or 3!=4
True
```

Tabela da Verdade

P	Q	P and Q
F	F	F
F	V	F
V	F	F
V	V	V

P	Q	P or Q
F	F	F
F	V	V
V	F	V
V	V	V

P	not P
F	V
V	F

Python – Comparação Strings

- Os operadores de comparação também podem ser utilizados com strings

```
>>> word = "banana"
>>> word == "apple"
False
>>> word == "banana"
True
>>> word == "Banana"
False
```

Python – condições (if and else)

A condição é seguida de :

```
if <BOOLEAN EXPRESSION>:
    <STATEMENTS_1> # Executed if condition evaluates to True
else:
    <STATEMENTS_2> # Executed if condition evaluates to False
```

```
if x < y:
    <STATEMENTS_A>
elif x > y:
    <STATEMENTS_B>
else:          # x == y
    <STATEMENTS_C
```

Abreviação do else if

=

```
if x < y:
    <STATEMENTS_A>
else:
    if x > y:
        <STATEMENTS_B>
    else:
      <STATEMENTS_C>
```

Python – condições (if and else)

```
if x > 0:
    if x < 10:
        print("x é um algarismo entre 0 e 9!")



if x > 0 and x < 10:
    print("x é um algarismo entre 0 e 9!")



if 0 < x < 10:            #Possivel no Python mas não em muitas linguagens
    print("x é um algarismo entre 0 e 9!")
```

Nota que temos diferentes formas de construir condições. Por vezes, a leitura do código fica mais simples sem os ses encadeados.

Python – condições (if and else)

- É possível negar condições utilizando a palavra reservado do Python ***not***

```
if not (age >= 18):
    print ("Ainda nao entrou na idade adulta!")


if age < 18:
    print ("Ainda nao entrou na idade adulta!")
```

$$\texttt{not (A == B)} \iff \texttt{A != B}$$
$$\texttt{not (A < B)} \iff \texttt{A >= B}$$
$$\texttt{not (A <= B)} \iff \texttt{A > B}$$

Python – ciclos

- Computadores são frequentemente utilizados para automatizar tarefas repetitivas.
- Repetição de uma sequência de instruções: iteração.
- Iteração em Python: ciclos ***for***, ciclos ***while***
- O ciclo **for** permite percorrer uma lista e é utilizado quando se conhece o número máximo de vezes que se irá iterar, ou seja, **iteração definida.**
- O ciclo **while** repete uma sequência de instruções de acordo com uma condição imposta à cabeça. É utilizado quando se vai iterar até que determinada condição se verifique**, ou seja, iteração indefinida**.

Python – ciclo for

- Vamos começar a trabalhar com o ciclo for utilizando a função **range** do Python

```
for x in range (10):
    print(x)
```

- Neste exemplo, a variável x vai tomar valores de 0 a 9. Um ciclo for **começa sempre em 0.**

Python – ciclo for

```
for x in range(5):   # 0, 1, 2, 3, 4
    print(x)

for x in range(10): # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    print(x)

for x in range(3,10): # 3, 4, 5, 6, 7, 8, 9
    print(x)

for x in range(3,10,2): # 3, 5, 7, 9
    print(x)
```

- **range (n)** → valores inteiros de 0 até n-1 inclusive
- **range(i,n)** → valores inteiros de i até n-1 inclusive
- **range(i,n,d)** → valores inteiros i, i + d, i + 2d, . . ., inferiores a n

Python – ciclo while

- Permite iterar sem saber a priori quantas vezes se vai executar o ciclo. Utiliza-se uma condição.

```
while <CONDITION>:
    <STATEMENTS>
```

Exemplo:

```
n=6
current_sum = 0
i=0
while i <= n:
    current_sum += i
    i += 1
print(current_sum)
```

Python – break e continue

`break`: interrompe a execução de um ciclo

```
for i in range(10):
    print(i)
    if i == 3:
        break

print("completed")
```

```
0
1
2
3
completed
```

`continue`: passa diretamente para a próxima iteração de um ciclo

```
i = 0
while i < 10:
    i += 1
    if i < 7:
        continue
    print(i)
print("completed")
```

```
7
8
9
10
completed
```

Python – Incrementar Variáveis

- Em programação, é necessário frequentemente fazermos o incremento ou o decremento de variáveis. Analisemos os seguinte excertos de código.

```
i = i + 1
i += 1      # forma mais curta (e mais habitual)
```

← Incrementar

```
i -= 1
```

← Decrementar

```
v = v + k    ↔   v += k
v = v - k    ↔   v -= k
v = v * k    ↔   v *= k
v = v / k    ↔   v /= k
v = v // k   ↔   v //= k
v = v ** k   ↔   v **= k
```

← Outros operadores

Python - Strings

- **Strings** são sequências de carateres
- No Python, identificamos como strings tudo o que estiver entre “” ou ‘’
- Utilizadas para representar frases ou palavras

```
string1 = "Bem-vindos"
string2 = string1 - 1
string3 = "Bem-vindos" / 123
string4 = string1 * "Bem-vindos"
string5 = "15" + 2
```

Quais dos comandos que envolvem strings estão corretos?

Python - Strings

- A maior parte dos operadores aritméticos não funciona com strings
- Exceções: **concatenação** de strings (+), **repetição** (*)
- **Precedência**: tal como com **operações numéricas**.
- **Strings** são tipos de **dados compostos**. Ao contrário do **int, float ou bool** que são tipos de **dados simples ou primitivos**.
- Às strings chamamos também **coleções de dados**.

Python – Strings concatenação e repetição

- A concatenação de strings é realizada através do comando +

```
a = "Fundamentos"
b = "de"
c = "Programacao"
espaco = " "

print (a + espaco + b + espaco + c)
```

- A repetição de uma string é realizada através do comando *

```
>>> string1 = "Ola"
>>> string1*3
'OlaOlaOla'
>>>
```

- Como vimos a entrada e saída de dados deve ser feita usando strings como tipos de dados.

Python – Strings indexação

- **Operador de indexação**: parêntesis retos ([])
- **Valor entre parêntesis**: índice
- **Primeiro carater**: índice 0
- **Último carater**: índice -1 (ou n-1)

```
>>> disciplina = "Fundamentos de PRG"
>>> letra = disciplina[2]
>>> print(letra)
```

- Qual o resultado do print?

Python – Strings indexação

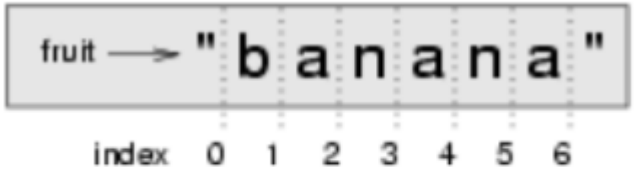
- Podemos obter o tamanho de uma string utilizando a função `len(string)` do Python
- Podemos ter acesso a índices a contar do fim utilizando índices negativos

```
>>> disciplina[-2]
'R'
```

- Acesso a índices maiores que o tamanho da string origina um erro
- Não se pode alterar valores de carateres de strings

Python – Fatias de Strings

- Em python podemos extrair partes da string utilizando os índices dos carateres

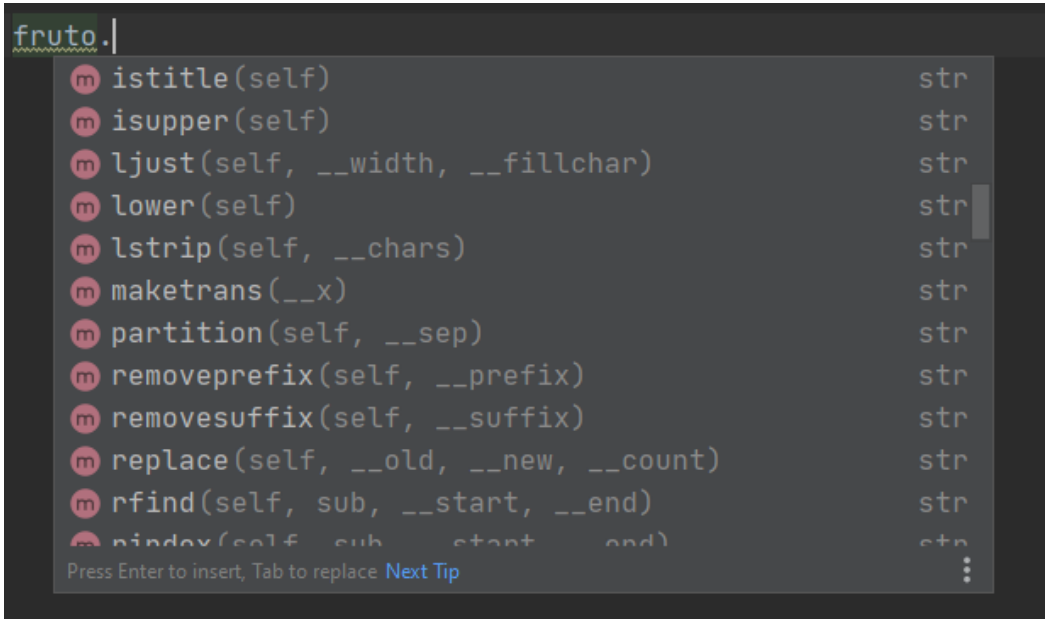



```
>>> fruit = "banana"
>>> fruit[:3]
'ban'
>>> fruit[3:]
'ana'
>>> fruit[2:5]
'nan'
```

Python – métodos para Strings

- O Python oferece diversos métodos que podemos utilizar sobre strings

```
>>> fruto = 'banana'
>>> fruto.find('ana')
1
```



Python – Substrings

```
>>> phrase = "Pirates of the Caribbean"
>>> print(phrase[0:5])
Pirat
```

- txt[i:j] substring entre indices i e j-1 inclusive
- txt[i:] substring desde o índice i até ao final
- txt[:j] substring desde o ínicio até ao índice j-1 inclusive

- Quando o valor de j for superior ao tamanho da string, no acesso a uma fatia isso não dá erro (resulta na string desde i até ao final)

Python – testar ocorrências em Strings

`txt1 in txt2` → testa ocorrência de `txt1` dentro de `txt2`

```
>>> 'ana' in 'Banana'
True
>>> 'mana' in 'Banana'
False
>>> 'mana' not in 'Banana'
True
>>> 'Banana' in 'Banana'
True
>>> '' in 'Banana'
True
```

Python – Percorrer strings

```
word="Banana"
for letter in word:
    print(letter)
```

```
for i in range(len(word)):
    print(i, word[i])
```

Se as strings são espécie de vetores, podemos percorrê-los

```
i = 0
while i < len(word):
    letter = word[i]
    print(i, letter)
    i += 1
```

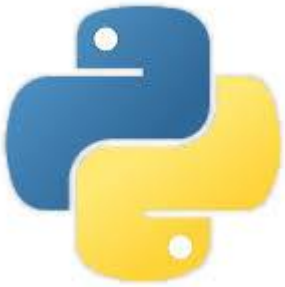
Python - Exemplo

```
def remove_vowels(phrase):
    vowels = "aeiou"
    string_sans_vowels = ""
    for letter in phrase:
        if letter.lower() not in vowels:
            string_sans_vowels += letter
    return string_sans_vowels
```

```
>>> remove_vowels("Programacao I")
'Prgrmc '
```

Python

- Depois de uma aplicação implementada funcionar:
  - Devemos simplificar o código
  - Melhorar a sua leitura
  - Fazer comentários usando o simbolo # que nos façam recordar as ideias implementadas



UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.