Computação Distribuída
2019 / 2020

Licenciatura em Engenharia Informática

Lab. 02 – Ciclos e dicionários em Python

1. Ciclo for

Em Python, um ciclo *for* serve para iterar sobre os elementos de uma lista de modo a executar determinadas instruções tendo como base cada elemento da lista. No exemplo seguinte, o programa faz o *print* de cada elemento da lista.

```
for value in [0, 1, 2, 3, 4, 5]:
    print(value)
```

Resolva os seguintes exercícios:

1. Faça uma função que retorne a soma de todos os elementos de uma lista. Use o ciclo *for* para iterar sobre os elementos da lista.

2. Faça uma função que indique o valor máximo de uma lista e a sua posição na lista.

3. Faça uma função que retorne o reverso de uma lista.

4. Implemente a função *is_sorted* que verifique se uma lista está ordenada por ordem crescente. Por exemplo, a lista [1, 2, 2, 3] está ordenada por ordem crescente, enquanto a lista [1, 2, 3, 2] não está.

5. Implemente a função *is_sorted_desc* que verifique se uma lista está ordenada por ordem decrescente.

6. Implemente a função *has_duplicates* que verifique se uma lista contém elementos duplicados. Sugestão: use dois ciclos *for*, onde para cada número verifique o resto da lista.

2. Ciclo while

Um ciclo *while* é, semelhantemente ao ciclo *for*, uma instrução que permite repetir um determinado número de instruções. No entanto, tem com base de partida uma condição que permite manter a repetição apenas enquanto essa condição se mantiver verdadeira.

```
n = 10
while n > 0:
    print(n)
    n = n-1

> 10, 9, 8, ..., 1
```

O exemplo anterior pode ser lido literalmente: “enquanto *n* for maior que zero, mostra o valor de *n* e depois reduz o valor de *n* por 1”. Quando o valor de *n* chegar a zero o ciclo termina.

1. Faça uma função que receba um número *n* e imprima no ecrã, por ordem decrescente, que números são pares e que números são ímpares.

2. Faça uma nova função que receba um número *n* e imprima todos os múltiplos de 3 entre 0 e *n*, por ordem crescente.

3. Dicionários

Em Python, um dicionário é uma estrutura de dados que permite indexar um valor por uma *chave*. O seguinte exemplo mostra-nos como podemos indexar as idades dos alunos por nome de aluno:

```
ages = {
    “Pedro”: 10,
    “Isabel”: 11,
    “Ana”: 9,
    “Tomás”: 10,
    “Manuel”: 10,
    “José”: 11,
    “Maria”: 12,
    “Gabriel”: 10,
}

>>> print(ages[“Pedro”])
10
```

Usando o exemplo das idades dado acima, e recorrendo à documentação do Python sobre dicionários (https://docs.python.org/3.5/library/stdtypes.html#typesmapping), resolva os seguintes exercícios:

1. Obtenha o número de alunos no dicionário.

2. Faça uma função que receba o dicionário como parâmetro e devolva a média de idades dos alunos. Sugestão: utilize o método *items()* como neste exemplo: https://docs.python.org/3.5/tutorial/datastructures.html#looping-techniques

3. Faça uma função que receba o dicionário e retorne o nome do aluno mais velho.

4. Faça uma função que receba o dicionário e um número *n* e retorne um novo dicionário em que os alunos estão *n* anos mais velhos. Ex: *nova_idade(ages, 10)* retorna os alunos 10 anos mais velhos.

4. Sub-dicionários

Nos dicionários Python, uma chave pode indexar um valor simples, como um número, ou então um valor mais complexo, como uma lista ou mesmo um outro dicionário.

```
students = {
    "Pedro": {"idade": 10, "morada": "Setúbal"},
    "Isabel": {"idade": 11, "morada": "Azeitão"},
    "Ana": {"idade": 9, "morada": "Setúbal"},
}

>>> students[“Pedro”][“morada”]
Setúbal
```

Usando o exemplo anterior, resolva os seguintes exercícios:

1. Quantos estudantes estão no dicionário *students*? Utilize a função apropriada.

2. Faça uma função que receba como argumento o dicionário e calcule a média de idades dos alunos.

3. Faça uma função que receba o dicionário, o nome de uma morada e retorne o nome de todos os alunos que cuja morada seja a morada passada por argumento. Por exemplo, a invocação da função *find_students(students, “Setúbal”)* deverá retornar o Pedro e a Ana.

(fim de enunciado)