INSTITUTO DE COMPUTAÇÃO
UFRJ

COMPUTAÇÃO 1 – AULA 9

Dicionários

Prof. Cesar Raitz

1. Introdução

Já encontramos variáveis associadas, como corredor e tempo de volta em:

```
corredores = ['Bowser', 'Mario', 'Luigi', 'Peach']
tempos = [42.7, 36.2, 57.2, 29.5]
```

- `corredores[i]` está associado a `tempos[i]`
- Só tomar cuidado de inserir/apagar elementos nas duas listas
- Para encontrar o tempo de um corredor, precisamos escrever:

```
i = list.index(corredores, 'Mario')
tempo_Mario = tempos[i]
```

E se usássemos `'Mario'` como índice, ao invés de um número `i`?

- Ou melhor, se `tempo_Mario = tempos['Mario']` ?
- Já sabemos que não é possível com listas e tuplas 😕
- Mas há um tipo de variável que permite, chamada **dicionário**!
- Se assemelha muito a um dicionário ou a uma agenda telefônica

2. Definição e sintaxe

Um dicionário é uma estrutura que guarda pares de chave – conteúdo.

- O nome é **chave** porque é usada para acessar (abrir) o **conteúdo.**
- Podemos nos referir a um par chave – conteúdo como uma **entrada**.

Criando um dicionário

- Vamos criar um dicionário de capitais:

```
1. capitais = {
2.   "Argentina": "Buenos Aires",
3.   "Brasil":    "Brasília",
4.   "Colômbia":  "Bogotá",
5.   "Perú":      "Lima"
6. }
```

- Portanto a sintaxe é

```
1. <variável dict> = {
2.   <chave 1>: <conteúdo 1>,
3.   <chave 2>: <conteúdo 2>,
4.   ...
5.   <chave N>: <conteúdo N>
6. }
```

- *Parênteses*, *colchetes* e *chaves* são usados em diferentes contextos:

Nome	Estrutura vazia	Nome do tipo	Pode confundir pois também são usados para
colchetes	`[]`	**`list`**	Indexar e Fatiar
parênteses	`()`	**`tuple`**	Chamar funções e Precedência matemática
chaves	`{}`	**`dict`**	Substituir variáveis em f-strings

Adicionando pares

- Outra forma de criar um dicionário é começar com um dicionário vazio e ir adicionando as entradas.
- Por exemplo, vamos criar uma agenda telefônica (apenas dos meus amigos mais próximos):

```
1. agenda = {}
2. agenda['Pedro Pascal'] = '99323-2233'
3. agenda['Margot Robbie'] = '98233-5522'
4. agenda['Ben Affleck'] = '99555-8999'
```

Note que:

- Em cada linha, associamos um **conteúdo** (no. de telefone) à uma **chave** (nome do contato).
- Tanto a chave como o conteúdo são do tipo **`str`**
- A chave funciona como se fosse um índice.

Modificando o conteúdo

- *Se a chave já existe*, podemos mudar seu conteúdo a qualquer momento:

```
5. agenda['Ben Affleck'] = '97000-1234'
```

Acessando o conteúdo

- Agora podemos imprimir o conteúdo pra ver se está atualizado:

```
6. n = agenda['Ben Affleck']
7. print('O número do Ben é', n)
```

- A variável `n` é apenas *uma referência para o mesmo conteúdo* da chave `'Ben Affleck'`. Então não precisamos criar uma variável:

```
6. print('O número do Ben é', agenda['Ben Affleck'])
7.
```

Apagando entradas

- Se um amigo(a) bloquear seu número, não adianta mais ter seu número na agenda. No meu caso, foi o Pedro:

```
8. del agenda['Pedro Pascal']
```

- Fique atento! *Tentar acessar uma chave inexistente* gera um erro `KeyError`.

```
print(agenda['Pedro Pascal'])
```

Testando a presença

- Lembra do operador **in** (dentro)? Funciona para strings, listas e tuplas. Será que funciona também com dicionários?

```
 9. print('Pedro está na agenda?',
10.       'Pedro Pascal' in agenda)
11. print('Robbie está na agenda?',
12.       'Margot Robbie' in agenda)
```

```
Pedro está na agenda? False
Robbie está na agenda? True
```

Exercício 1. Dicionário de kart.

No papel, crie um dicionário para o exemplo de corredores de kart da primeira página. O nome do corredor será a chave, e o tempo de volta será conteúdo. Depois apague a entrada de Bowser e imprima os tempos de Mario e Peach. Utilize o operador **in** para testar se Luigi está no dicionário.

3. Repetindo com dicionários

- Podemos imprimir um dicionário com **print**, e o resultado é...

```
{'Margot Robbie': '98233-5522', 'Ben Affleck':
'99555-8999'}
```

- Podemos imprimir cada entrada separadamente, mas como saber quais são as chaves do dicionário?
- A função `.keys()` fornece as chaves, ela é usada *como sufixo*: `agenda.keys()`

Caminhando nas chaves

- Mesmo assim, está muito trabalhoso! A boa notícia é que *dicionários são iteráveis!* Podemos usá-los com o laço FOR:

```
14. for k in agenda:
15.   print(f"{k:>20}: {agenda[k]}")
```

```
Margot Robbie: 98233-5522
  Ben Affleck: 99555-8999
```

- k é uma variável caminhante: em cada iteração, ela terá o valor de uma chave do dicionário (Margot Robbie, Ben Affleck etc.)
- `agenda[k]` é o conteúdo associado à chave k.
- No código acima, `agenda.keys()` tem o mesmo efeito.

Caminhando em chave e conteúdo

- A função sufixo `.items()` fornece um iterável em chave e conteúdo (como se fosse uma função `zip(chaves, conteúdos)`)

```
17. for k, numero in agenda.items():
18.   print(f"{k:>20}: {numero}")
```

- Percebeu que `numero` substitui `agenda[k]`?
- Quase sempre é melhor usar `.items()`

Caminhando nos conteúdos

A última forma de iterar num dicionário é olhando *apenas os conteúdos*, com a função sufixo `.values()`.

- Você pode obter uma lista de *todo o conteúdo*, transformando o iterável numa lista:
  `list( dicionario.values() )`.
- Se os conteúdos são números, podemos descobrir o mínimo com
  `min( list(dicionario.values()) )`.

Exercício 2. Tempos de volta.

Usando seu dicionário de kart, encontre os tempos de volta mínimo e máximo. É mais simples obter uma lista dos conteúdos primeiro, e depois usar as funções **`min`** e **`max`**.

Exercício 3. Múltiplos tempos de volta.

Imagine que cada corredor de kart deu 3 voltas. Assim, cada conteúdo será uma lista com 3 tempos, um para cada volta. Por exemplo:

```
kart['Daisy'] = [34.4, 37.3, 29.8]
kart['Kong'] = [39.5, 33.3, 35.8]
# por aí vai...
```

Escreva um código para mostrar o tempo médio de volta de cada corredor. *Dica:* Dentro de um laço `for`, você pode calcular a média do conteúdo (uma lista de valores) e imprimir o resultado junto com a chave (nome do corredor).

Tipos de chave e conteúdo

- Chaves devem ser imutáveis
- Conteúdo pode ser qualquer um

```
1. meu_dicionario = {}
2. meu_dicionario['idade'] = 25
3. meu_dicionario[42] = 'trigo'
4. meu_dicionario[True] = [4, 6, 10]
```

4. Dicionário Avançado

Outra forma de criar dicionários

```
1. harry = {'nome': 'Harry',
2.          'sobrenome': 'Potter',
3.          'casa': 'Grifinória',
4.          'amigos': ['Ron', 'Hermione'],
5.          'nascido': 1980}
```

- Chave e conteúdo são separados por dois pontos
- Cada entrada é separada por vírgula
- Os delimitadores são chaves { } ☹
- Também podemos imprimi-lo diretamente. **`print`**`(harry)` mostra

```
{'nome': 'Harry', 'sobrenome': 'Potter',
'casa': 'Grifinória', 'amigos': ['Ron',
'Hermione'], 'nascido': 1980}
```

Mais uma forma de criar dicionários

- Não precisa usar, mas é bom saber que existe!

```
1. filme = dict(nome='O Poderoso Chefão',
2.              ano=1972,
3.              nota='*****')
4.
5. print(filme['nome'])
6. print(filme['ano'])
7. print(filme['nota'])
```

Conteúdo padrão com .get()

- Sempre é necessário verificar se a chave existe antes de modificar seu conteúdo:

```
1. if 'Romário'not in gols:
2.   gols['Romário'] = 0
3. # Agora já tem a chave Junior
4. gols['Romário'] += 1
```

- A função **dict.get()** fornece um valor padrão caso a chave não exista:

```
5. gols['Romário'] = gols.get('Romário', 0) + 1
```

Copiando o dicionário com .copy()

- Se você precisar associar novos valores a um dicionário, crie uma cópia!

```
1. friend_notes = agenda.copy()
2. friend_notes['Ben Affleck'] = 'Fica chato quando bebe'
3. friend_notes['Margot Robbie'] = 'Me deve R$ 20'
```

- Mas fique atento quando o conteúdo for uma lista: se você alterar a lista na cópia, estará alterando a lista original!

5. Alguns exemplos

Exemplo 1. Recomendação de Filmes

```
1. filme1 = dict(nome='O Planeta dos Macacos',
2.               nota='****', ano=1968)
3.
4. filme2 = dict(nome='Crepúsculo',
5.               nota='****', ano=2008)
6.
7. filme3 = dict(nome='O Poderoso Chefão',
8.               nota='*****', ano=1972)
9.
10. lista_filmes = [filme1, filme2, filme3]
11.
12. # Ops! Corrigindo a nota:
13. filme2['nota'] = '**'
14.
15. print("Recomendamos os Filmes")
16. print("----------------------")
```

```
17. for filme in lista_filmes:
18.   # filme é um dicionário!
19.   nome = filme['nome']
20.   nota = filme['nota']
21.   ano = filme['ano']
22.
23.   if len(nota) >= 4:
24.     # Só recomendamos com + de 4 estrelas!
25.     print(f"{nome} ({nota}) {ano}")
```

```
Recomendamos os Filmes
----------------------
O Planeta dos Macacos (****) 1968
O Poderoso Chefão (*****) 1972
```

- Reparou que `filmes` é uma lista de dicionários?

Exemplo 2. Rede Antissocial

- Precisamos cadastrar e-mail, idade e amigos de cada usuário
- Poderíamos usar uma lista, i.e. [e-mail, idade, lista de amigos]
- Quando a rede crescer, haverá mais atributos (número de pets, comidas preferidas, ...). Como vamos saber a ordem?
- Melhor seria ter *atributos com nome*:

```
1. user = {'email': 'walds_21@bcmt.br',
2.         'amigos': ['Rubiula', 'Ellisson'],
3.         'idade': 22}
```

```
4.
5. rede = {}
6. # Cadastra o primerio usuário!
7. rede['Waldisney'] = user
```

- O conteúdo para Waldisney é um dicionário, então podemos acessar seus atributos pelas respectivas chaves:

```
 8. dic = rede['Waldisney']
 9. print('email: ', dic['email'])
10. print('idade: ', dic['idade'])
11. print('amigos:', dic['amigos'])
```

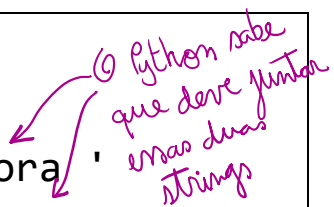
- A essas alturas, você já deve ter entendido, a variável é apenas um nome apontando para os dados. Podemos então pular esse intermediário:

```
 8.
 9. print('email: ', rede['Waldisney']['email'])
10. print('idade: ', rede['Waldisney']['idade'])
11. print('amigos:', rede['Waldisney']['amigos'])
```

- Vamos cadastrar só mais alguns usuários para testar:

```
13. rede['Rubiula'] = {
14.     'email': 'ruru@milizema.br',
15.     'amigos': ['Waldisney', 'Ellisson'],
16.     'idade': 25 }
17.
18. rede['Ellisson'] = {
19.     'email': 'ellison@smail.com',
20.     'amigos': ['Rubiula'],
```

```
21.     'idade': 18 }
22.
23. print('A rede antissocial agora ' 
24.       f'tem {len(rede)} usuários!')
```




```
A rede antissocial agora tem 3 usuários!
```

Exercício 4. Usuário mais antissocial.

Agora vamos ao que interessa: encontrar o usuário mais antissocial da rede!

A maioria dos algoritmos utiliza variáveis auxiliares, criadas no início dos algoritmos. Você já encontrou variáveis *de valor fixo*, *contadores*, *acumuladores* e *caminhantes*. No template que segue, `mais_antis` e `menor_numero` auxiliam na busca do **melhor candidato**. Elas precisam começar com valores absurdos, para que possam ser atualizadas ao longo da estrutura de repetição.

```
26. mais_antis = 'X'
27. menor_numero = ________
28.
29. for nome, atributos in rede__________:
30.   lista_amigos = atributos[__________]
31.   num_amigos = len(__________)
32.   if __________ < menor_numero:
33.     menor_numero = __________
34.     mais_antis = __________
```

```
35.
36. print(f'O vencedor é {mais_antis} com '
37.       f'{menor_numero} amigos!')
```

6. Exercícios

Exercício 5. Dicionário de médias.

Crie uma função que receba um *dicionário de kart* como argumento (aquele com lista de tempos, como no Exercício 3) e retorne um dicionário com a média dos tempos de cada corredor. *Sugestão: Copie o dicionário e itere sobre as chaves, substituindo o conteúdo pela média da lista.*

Exercício 6. Melhor e pior voltas?

Crie uma função que receba um *dicionário de kart* como argumento (aquele com lista de tempos, como no Exercício 3) e imprima o nome dos corredores com melhor e pior tempo de volta, assim como os respectivos tempos.

Exercício 7. Converter para algarismos romanos.

Escreva um programa que converta números inteiros entre 0 e 999 para algarismos romanos. Use os três dicionários abaixo:

```
UNIDADES = {
  0:'',  1:'I',  2:'II',  3:'III',  4:'IV',
  5:'V', 6:'VI', 7:'VII', 8:'VIII', 9:'IX'}
DEZENAS = {
  0:'',  1:'X',  2:'XX',  3:'XXX',  4:'XL',
  5:'L', 6:'LX', 7:'LXX', 8:'LXXX', 9:'XC'}
CENTENAS = {
  0:'',  1:'C',  2:'CC',  3:'CCC',  4:'CD',
  5:'D', 6:'DC', 7:'DCC', 8:'DCCC', 9:'CM'}
```

Exercício 8. Contagem de palavras.

Contar palavras é um recurso importante para análise semântica de textos, sendo usada em conjunto com técnicas de *Inteligência Artificial*. Construa uma função que receba uma string e retorne um dicionário com o número de ocorrências de cada palavra. Por exemplo:

```
conta_palavras("Ai se eu te pego, ai ai se eu te pego")
# retorna: {'ai': 3, 'se': 2, 'eu': 2, 'te': 2, 'pego': 2}
```

Repare que não importa se a palavra está em maiúscula ou minúscula.

Dica: Comece com um dicionário vazio e faça um laço iterando sobre cada palavra (função `str.split()`*).*

Exercício 9. Média de idade dos amigos.

Usando a rede antissocial do Exemplo 2 (rede será uma variável global), construa uma função que retorne a média da idade dos amigos de uma pessoa. Inclua estes casos de teste:

```
# CASOS DE TESTE
x = media_idade_dos_amigos("Waldisney")  # x = 21.5
y = media_idade_dos_amigos("Rubia")      # y = 20.0
z = media_idade_dos_amigos("Ellison")    # z = 25.0
```

Exercício 10. Fatiando dicionários.

O problema de dicionários é a falta de fatiamento. É mesmo? Podemos definir a fatia de 'Ana' a 'Maria' como todas as entradas que estão alfabeticamente entre 'Ana' e 'Maria'. Então, 'João' estaria na fatia, pois 'Ana' < 'João' e 'João' < 'Maria'. Escreva uma função para fatiar um dicionário por ordem alfabética.