



Funções de dicionário (parte 1)



```
# Criando um dicionário vazio
dicionario_vazio = dict()
print(dicionario_vazio) # Output: {}

# Criando um dicionário com pares chave-valor
dicionario_pessoa = dict(nome="Alice", idade=30, cidade="São Paulo")
print(dicionario_pessoa) # Output: {'nome': 'Alice', 'idade': 30, 'cidade': 'São Paulo'}
```

• `get()`: Retorna o valor associado a uma chave, mas evita um erro se a chave não existir, retornando um valor padrão (ou None).

python

Copiar

```
idade = pessoa.get("idade")
print(idade) # Output: 30

# Tentando acessar uma chave inexistente sem causar um erro
estado = pessoa.get("estado", "Desconhecido")
print(estado) # Output: Desconhecido
```



Funções de dicionário (parte 1)

keys(): Retorna uma lista (ou view) das chaves no dicionário.

python

```
chaves = pessoa.keys()
print(chaves) # Output: dict_keys(['nome', 'idade', 'cidade'])
```

values(): Retorna uma lista (ou view) dos valores no dicionário.

python

```
valores = pessoa.values()
print(valores) # Output: dict_values(['Alice', 30, 'São Paulo'])
```

items(): Retorna uma lista (ou view) de pares chave-valor (tuplas).

python

```
itens = pessoa.items()
print(itens) # Output: dict_items([('nome', 'Alice'), ('idade', 30), ('cidade', 'São Paulo')])
```



Iteração (loop) em um dicionário

```
main.py +
1 variaveis = {
2     "numero": 10,
3     "palavra": "oi",
4     "dinheiro": 31.14,
5     "booleano": True,
6     "frutas": ["maçã", "banana", "uva", "pera", "abacate"]
7 }
8
9 for nome_chave, nome_item in variaveis.items():
10     print("o tipo da variável", nome_chave, "=", type(nome_item))
11
12 |
13 print(type(variaveis))
```

Ln: 12, Col: 5



Run



Share



\$

Command Line Arguments



```
o tipo da variável numero = <class 'int'>
o tipo da variável palavra = <class 'str'>
o tipo da variável dinheiro = <class 'float'>
o tipo da variável booleano = <class 'bool'>
o tipo da variável frutas = <class 'list'>
<class 'dict'>
```

Esse loop passa por todos os elementos do dicionário (`_dict_`) de `teml`:

`_dict_.keys()` = pega cada chave somente

`_dict_.values()` = pega cada valor somente

`_dict_.items()` = pega a chave e o valor



@fpftech.educacional

Funções de dicionário (parte2)



- **update()**: Atualiza o dicionário com pares chave-valor de outro dicionário ou de um iterável de pares.

python

Copiar

```
pessoa.update({"idade": 31, "estado": "SP"})  
print(pessoa) # Output: {'nome': 'Alice', 'idade': 31, 'cidade': 'São Paulo', 'estado': 'SP'}
```

- **pop()**: Remove a chave especificada e retorna o valor correspondente.

python

Copiar

```
idade_removida = pessoa.pop("idade")  
print(idade_removida) # Output: 31  
print(pessoa) # Output: {'nome': 'Alice', 'cidade': 'São Paulo', 'estado': 'SP'}
```



@fpftech.educacional

Para definir uma função em Python, utiliza-se a palavra-chave `def`, seguida pelo nome da função, parênteses `()`

Podem conter parâmetros, e dois pontos `:`.

O código da função é então escrito em um bloco indentado.

Primeiro vc cria a função, depois vc chama ela

```
def saudacao():  
    print("Olá, mundo!")
```

```
saudacao() # Output: Olá, mundo!
```



Parametros vs. argumentos

Exemplo com Parâmetros:

python

```
def saudacao(nome):  
    print(f"Olá, {nome}!")
```

Parametro =
quando cria a
função,
vc cria parâmetros
pra ela.

Chamando a Função com um Argumento:

python

```
saudacao("Alice") # Output: Olá, Alice!
```

Argumento =
quando vc chama a
função,
vc passa
argumentos pra ela

Parametros nomeados

```
def criar_usuario(nome, sobrenome, idade, cidade):  
    print(f"Nome: {nome}")  
    print(f"Sobrenome: {sobrenome}")  
    print(f"Idade: {idade}")  
    print(f"Cidade: {cidade}")
```

```
# Chamando a função usando parâmetros nomeados  
criar_usuario(nome="Maria", sobrenome="Silva", idade=30, cidade="São Paulo")
```



Exemplo com Parâmetros Opcionais:

python

```
def saudacao(nome="mundo"):  
    print(f"Olá, {nome}!")
```

Chamando a Função com ou sem Argumentos:

python

```
saudacao()  
saudacao("Carlos")
```

Output: Olá, mundo!
Output: Olá, Carlos!



Uma função pode retornar um valor usando a palavra-chave return.

Isso permite que a função envie um resultado de volta para o ponto onde foi chamada.

Exemplo com Retorno:

python

```
def soma(a, b):  
    return a + b
```

Chamando a Função e Usando o Valor Retornado:

python

```
resultado = soma(3, 4)  
print(resultado) # Output: 7
```



Return múltiplos valores (tupla)

Uma função pode retornar múltiplos valores usando tupla.

```
def dividir(a, b):  
    quociente = a // b  
    resto = a % b  
    return quociente, resto
```

IMPORTANTE = se preparar pra receber os múltiplos valores (tupla) usando varias variáveis pra receber (no exemplo q, r)

```
q, r = dividir(10, 3)  
print(f"Quociente: {q}, Resto: {r}") # Output: Quociente: 3, Resto: 1
```

Exemplo enumerate() / dicionario.items()



Parametro dinâmico

Imagine precisar criar uma função que some valores, porém, você pode passar dois parâmetros ou 3, ou 4 e assim por diante.

*args permite que você passe um número variável de argumentos posicionais para uma função.

Dentro da função, args será tratado como uma tupla.

```
def soma(*args):  
    resultado = 0  
    for num in args:  
        resultado += num  
    return resultado
```

```
# Exemplo de uso  
print(soma(1, 2, 3, 4))  
print(soma(10, 20))
```



Parametro dinâmico

E se forem dicionários?

`**kwargs` permite que você passe um número variável de argumentos nomeados (ou seja, pares chave-valor) para uma função.

Dentro da função, `kwargs` será tratado como um dicionário.

```
def exibir_informacoes(**kwargs):  
    for chave, valor in kwargs.items():  
        print(f"{chave}: {valor}")
```

```
exibir_informacoes(nome="João", idade=30, cidade="São Paulo")
```



1. len()

- **Descrição:** Retorna o comprimento da string
- **Exemplo:**

```
python
```

```
texto = "Python"  
print(len(texto))    # Saída: 6
```



2. str.upper()

- **Descrição:** Converte todos os caracteres da string para maiúsculas
- **Exemplo:**

```
python
```

```
texto = "Python"  
print(texto.upper()) # Saída: "PYTHON"
```



3. str.lower()

- **Descrição:** Converte todos os caracteres da string para minúsculas
- **Exemplo:**

```
python
```

```
texto = "Python"  
print(texto.lower()) # Saída: "python"
```



4. str.capitalize()

- **Descrição:** Converte o primeiro caractere da string para maiúsculo e o resto para minúsculo.
- **Exemplo:**

python

```
texto = "python programming"  
print(texto.capitalize()) # Saída: "Python programming"
```



5. str.strip()

- **Descrição:** Remove espaços em branco do início e do fim da string.
- **Exemplo:**

```
python
```

```
texto = " Python "  
print(texto.strip()) # Saída: "Python"
```



6. str.replace()

- **Descrição:** Substitui uma substring por outra.
- **Exemplo:**

```
python
```

```
texto = "Hello, World!"  
print(texto.replace("World", "Python")) # Saída: "Hello, Python!"
```



7. str.find()

- **Descrição:** Retorna o índice da primeira ocorrência de uma substring
- **Exemplo:**

python

```
texto = "Hello, World!"  
print(texto.find("World")) # Saída: 7
```

. Retorna -1 se a substring não for encontrada.



8. str.split()

- **Descrição:** Divide a string em uma lista de substrings com base em um delimitador.
- **Exemplo:**

```
python
```

```
texto = "Python is fun"  
print(texto.split()) # Saída: ['Python', 'is', 'fun']
```



9. str.join()

- **Descrição:** Junta uma sequência de strings em uma única string,
- **Exemplo:**

```
python
```

```
palavras = ['Python', 'is', 'fun']  
print(" ".join(palavras)) # Saída: "Python is fun"
```

usando uma string como delimitador.



10. str.startswith()

- **Descrição:** Verifica se a string começa com uma determinada substring.
- **Exemplo:**

```
python
```

```
texto = "Python"  
print(texto.startswith("Py")) # Saída: True
```



11. str.endswith()

- **Descrição:** Verifica se a string termina com uma determinada substring
- **Exemplo:**

python

```
texto = "Python"  
print(texto.endswith("on"))    # Saída: True
```



13. str.isalpha()

- **Descrição:** Verifica se todos os caracteres na string são letras.
- **Exemplo:**

```
python
```

```
texto = "Python"  
print(texto.isalpha()) # Saída: True
```



14. str.count()

- **Descrição:** Conta o número de ocorrências de uma substring na string.
- **Exemplo:**

python

```
texto = "banana"  
print(texto.count("a"))    # Saída: 3
```



15. str.format()

- **Descrição:** Formata a string utilizando placeholders {}.
- **Exemplo:**

python

```
nome = "João"  
idade = 30  
print("Meu nome é {} e eu tenho {} anos.".format(nome, idade))  
# Saída: "Meu nome é João e eu tenho 30 anos."
```



1. print()

- **Descrição:** Exibe informações no console.
- **Exemplo:**

```
python
```

```
print("Olá, Mundo!") # Saída: Olá, Mundo!
```



2. len()

- **Descrição:** Retorna o número de itens em um objeto,
- **Exemplo:**

python

```
lista = [1, 2, 3, 4]  
print(len(lista))    # Saída: 4
```

3. type()

- **Descrição:** Retorna o tipo do objeto passado como argumento.
- **Exemplo:**

python

```
print(type(10))           # Saída: <class 'int'>
print(type("Python"))    # Saída: <class 'str'>
```



4. input()

- **Descrição:** Lê uma entrada do usuário e a retorna como uma string.
- **Exemplo:**

python

```
nome = input("Digite seu nome: ")  
print("Olá,", nome)
```



5. int(), float(), str()

- **Descrição:** Convertem valores para inteiros (`int`), ponto flutuante (`float`), e string (`str`).
- **Exemplo:**

python

```
numero_str = "123"  
numero_int = int(numero_str)  
numero_float = float(numero_str)  
print(numero_int, numero_float)  # Saída: 123 123.0
```



6. sum()

- **Descrição:** Retorna a soma dos elementos de um iterável, como uma lista ou tupla.
- **Exemplo:**

python

```
numeros = [1, 2, 3, 4]  
print(sum(numeros)) # Saída: 10
```



7. max() e min()

- **Descrição:** Retornam o maior e o menor valor de um iterável, respectivamente.
- **Exemplo:**

python

```
numeros = [1, 2, 3, 4]
print(max(numeros)) # Saída: 4
print(min(numeros)) # Saída: 1
```



Tabela ASCII



tech
nológica

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr	Dec	Hex	Oct	HTML	Chr
0	0	000	NULL	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	Start of Header	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	Start of Text	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	End of Text	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	End of Transmission	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	Enquiry	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	Acknowledgment	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	Bell	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	Backspace	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	Horizontal Tab	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	Line feed	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	Vertical Tab	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	Form feed	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	Carriage return	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	Shift Out	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	Shift In	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	Data Link Escape	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	Device Control 1	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	Device Control 2	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	Device Control 3	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	Device Control 4	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	Negative Ack.	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	Synchronous idle	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	End of Trans. Block	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	Cancel	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	End of Medium	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	Substitute	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	Escape	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	File Separator	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	Group Separator	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	Record Separator	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	Unit Separator	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		Del



8. sorted()

- **Descrição:** Retorna uma nova lista ordenada a partir dos elementos de um iterável.
- **Exemplo:**

```
python
```

```
numeros = [3, 1, 4, 2]  
print(sorted(numeros)) # Saída: [1, 2, 3, 4]
```



9. range()

- **Descrição:** Gera uma sequência de números, geralmente usada em loops.
- **Exemplo:**

python

 Copiar

```
for i in range(5):  
    print(i) # Saída: 0 1 2 3 4
```

10. enumerate()

- **Descrição:** Retorna um objeto enumerado, que contém pares de índice e valor.
- **Exemplo:**

python

 Copiar

```
lista = ["a", "b", "c"]  
for indice, valor in enumerate(lista):  
    print(indice, valor)  
# Saída:  
# 0 a  
# 1 b  
# 2 c
```



Obrigado!



    @fpftech.educacional