**PYTHON – GUIA RÁPIDO**

Claro! O comando `print` em Python é uma das funções mais usadas e serve para exibir informações no console. A função `print` é bastante flexível e oferece várias maneiras de formatar a saída. Vamos explorar as diferentes formas de usar o `print` e como fazer formatações diversas.

## 1. Sintaxe Básica

A forma mais simples de usar o `print` é passar os valores que você quer exibir como argumentos:

```python

print("Olá, Mundo!")

```

Você pode passar múltiplos argumentos, separados por vírgulas, e eles serão exibidos com um espaço entre eles:

```python

print("Olá", "Mundo", "!")

```

## 2. Parâmetros da Função `print`

A função `print` aceita alguns parâmetros opcionais que controlam seu comportamento:

- `sep`: Define o separador entre os argumentos. Por padrão, é um espaço.

- `end`: Define o que será impresso ao final. Por padrão, é uma nova linha (`\n`).

- `file`: Define o objeto onde a saída será enviada. Por padrão, é o `sys.stdout` (o console).

- `flush`: Se `True`, força a limpeza do buffer de saída. Por padrão, é `False`.

Exemplos:

```python

print("Olá", "Mundo", sep="-") # Saída: Olá-Mundo

print("Olá", end="!") # Saída: Olá!

```

## 3. Formatação de Strings

### 3.1. Concatenação Simples

Usar o operador `+` para concatenar strings:

```python

nome = "João"

print("Olá, " + nome + "!")

```

### 3.2. Substituição com `%`

Você pode usar o operador `%` para formatar strings:

```python

idade = 25

print("Eu tenho %d anos" % idade)

```

### 3.3. Método `str.format()`

O método `str.format()` permite substituir marcadores `{}` por valores:

```python

nome = "Ana"

idade = 30

print("Meu nome é {} e eu tenho {} anos".format(nome, idade))

```

Você também pode usar índices e nomes nos marcadores:

```python

print("Meu nome é {0} e eu tenho {1} anos".format(nome, idade))

print("Meu nome é {nome} e eu tenho {idade} anos".format(nome="Carlos", idade=22))

```

### 3.4. Formatação com `f-strings` (Python 3.6+)

As f-strings permitem inserir expressões dentro das chaves `{}` prefixando a string com `f`:

```python

nome = "Mariana"

idade = 27

print(f"Meu nome é {nome} e eu tenho {idade} anos")

```

Você pode fazer operações diretamente nas f-strings:

```python

print(f"Daqui a 5 anos, {nome} terá {idade + 5} anos")

```

## 4. Controle de Precisão e Largura

Você pode controlar a precisão e a largura de números em formatações:

```python

pi = 3.14159265

print("Valor de Pi: {:.2f}".format(pi)) # Saída: Valor de Pi: 3.14

print(f"Valor de Pi: {pi:.2f}") # Saída: Valor de Pi: 3.14

numero = 42

print("Número com largura 5: {:5d}".format(numero)) # Saída: Número com largura 5: 42

print(f"Número com largura 5: {numero:5d}") # Saída: Número com largura 5: 42

```

## 5. Exibição de Caracteres Especiais

Para exibir caracteres especiais, use barras invertidas (`\`):

- `\n`: Nova linha

- `\t`: Tabulação

Exemplo:

```python

print("Linha 1\nLinha 2")

print("Coluna 1\tColuna 2")

```

## 6. Exibição de Caracteres Unicode

Para exibir caracteres Unicode, use a sequência `\u` seguida do código hexadecimal do caractere:

```python

print("Símbolo de coração: \u2665")

```

Esses são os conceitos e formas principais de usar o `print` em Python, com diversas técnicas de formatação e controle da saída.

Claro! Vamos analisar o código:

python

Copiar código

cpf = ''.join(filter(str.isdigit, cpf))

Aqui está o que ele faz, passo a passo:

1. **filter(str.isdigit, cpf)**:
   * filter é uma função que filtra elementos de um iterável (no caso, a string cpf), retornando apenas os elementos que satisfazem a condição dada.
   * str.isdigit é um método de string que retorna True se o caractere é um dígito (ou seja, um número de 0 a 9) e False caso contrário.
   * Neste caso, filter(str.isdigit, cpf) cria um iterador que inclui apenas os caracteres da string cpf que são dígitos.
2. **''.join(...)**:
   * ''.join(...) é um método de string que junta todos os elementos de um iterável (como uma lista ou iterador) em uma única string. O que aparece antes de join (neste caso, a string vazia '') é usado como separador entre os elementos.
   * Como o separador é uma string vazia, ele apenas junta todos os dígitos sem inserir nada entre eles.
3. **Resultado**:
   * O resultado é uma nova string cpf, que contém apenas os números do cpf original, removendo qualquer caractere que não seja um dígito, como pontos (.), hífens (-), ou espaços.

**Exemplo**

Se cpf = "123.456.789-00", após executar o código, cpf se tornará "12345678900". Todos os caracteres que não são números foram removidos.

Isso é útil para normalizar o CPF antes de processá-lo, validá-lo ou armazená-lo.

Claro! Python oferece uma ampla variedade de funções e métodos para manipulação de strings. Vou listar algumas das mais comuns, com uma breve explicação de cada uma:

### 1. \*\*`len(string)`\*\*

- \*\*Descrição\*\*: Retorna o comprimento (número de caracteres) da string.

- \*\*Exemplo\*\*: `len("Python")` retorna `6`.

### 2. \*\*`string.lower()`\*\*

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

- \*\*Exemplo\*\*: `"PyThOn".lower()` retorna `"python"`.

### 3. \*\*`string.upper()`\*\*

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

- \*\*Exemplo\*\*: `"python".upper()` retorna `"PYTHON"`.

### 4. \*\*`string.capitalize()`\*\*

- \*\*Descrição\*\*: Converte o primeiro caractere da string para maiúscula e o restante para minúsculas.

- \*\*Exemplo\*\*: `"python".capitalize()` retorna `"Python"`.

### 5. \*\*`string.title()`\*\*

- \*\*Descrição\*\*: Converte o primeiro caractere de cada palavra para maiúscula.

- \*\*Exemplo\*\*: `"hello world".title()` retorna `"Hello World"`.

### 6. \*\*`string.strip()`\*\*

- \*\*Descrição\*\*: Remove espaços em branco no início e no fim da string.

- \*\*Exemplo\*\*: `" hello ".strip()` retorna `"hello"`.

### 7. \*\*`string.rstrip()`\*\*

- \*\*Descrição\*\*: Remove espaços em branco no final da string.

- \*\*Exemplo\*\*: `"hello ".rstrip()` retorna `"hello"`.

### 8. \*\*`string.lstrip()`\*\*

- \*\*Descrição\*\*: Remove espaços em branco no início da string.

- \*\*Exemplo\*\*: `" hello".lstrip()` retorna `"hello"`.

### 9. \*\*`string.replace(old, new)`\*\*

- \*\*Descrição\*\*: Substitui todas as ocorrências de `old` por `new` na string.

- \*\*Exemplo\*\*: `"hello world".replace("world", "Python")` retorna `"hello Python"`.

### 10. \*\*`string.split(separator)`\*\*

- \*\*Descrição\*\*: Divide a string em uma lista de substrings usando o `separator` como delimitador.

- \*\*Exemplo\*\*: `"hello world".split(" ")` retorna `["hello", "world"]`.

### 11. \*\*`string.join(iterable)`\*\*

- \*\*Descrição\*\*: Junta os elementos de um iterável (como uma lista) em uma única string, usando a string original como delimitador.

- \*\*Exemplo\*\*: `"-".join(["2023", "08", "20"])` retorna `"2023-08-20"`.

### 12. \*\*`string.find(substring)`\*\*

- \*\*Descrição\*\*: Retorna o índice da primeira ocorrência da `substring` na string. Retorna `-1` se não for encontrada.

- \*\*Exemplo\*\*: `"hello".find("l")` retorna `2`.

### 13. \*\*`string.index(substring)`\*\*

- \*\*Descrição\*\*: Similar ao `find`, mas lança um erro se a substring não for encontrada.

- \*\*Exemplo\*\*: `"hello".index("l")` retorna `2`.

### 14. \*\*`string.count(substring)`\*\*

- \*\*Descrição\*\*: Conta o número de vezes que a `substring` aparece na string.

- \*\*Exemplo\*\*: `"hello".count("l")` retorna `2`.

### 15. \*\*`string.startswith(prefix)`\*\*

- \*\*Descrição\*\*: Verifica se a string começa com a substring `prefix`. Retorna `True` ou `False`.

- \*\*Exemplo\*\*: `"hello".startswith("he")` retorna `True`.

### 16. \*\*`string.endswith(suffix)`\*\*

- \*\*Descrição\*\*: Verifica se a string termina com a substring `suffix`. Retorna `True` ou `False`.

- \*\*Exemplo\*\*: `"hello".endswith("lo")` retorna `True`.

### 17. \*\*`string.isdigit()`\*\*

- \*\*Descrição\*\*: Retorna `True` se todos os caracteres da string forem dígitos.

- \*\*Exemplo\*\*: `"12345".isdigit()` retorna `True`.

### 18. \*\*`string.isalpha()`\*\*

- \*\*Descrição\*\*: Retorna `True` se todos os caracteres da string forem letras.

- \*\*Exemplo\*\*: `"abc".isalpha()` retorna `True`.

### 19. \*\*`string.isalnum()`\*\*

- \*\*Descrição\*\*: Retorna `True` se todos os caracteres da string forem alfanuméricos (letras ou dígitos).

- \*\*Exemplo\*\*: `"abc123".isalnum()` retorna `True`.

### 20. \*\*`string.isspace()`\*\*

- \*\*Descrição\*\*: Retorna `True` se todos os caracteres da string forem espaços em branco.

- \*\*Exemplo\*\*: `" ".isspace()` retorna `True`.

### 21. \*\*`string.swapcase()`\*\*

- \*\*Descrição\*\*: Inverte o caso de todos os caracteres da string. Maiúsculas se tornam minúsculas e vice-versa.

- \*\*Exemplo\*\*: `"Python".swapcase()` retorna `"pYTHON"`.

### 22. \*\*`string.zfill(width)`\*\*

- \*\*Descrição\*\*: Retorna a string preenchida à esquerda com zeros até o comprimento especificado por `width`.

- \*\*Exemplo\*\*: `"42".zfill(5)` retorna `"00042"`.

Esses são alguns dos métodos e funções mais utilizados para manipulação de strings em Python. Eles podem ser combinados de várias maneiras para realizar operações mais complexas.