Lógica de Programação

Acessando Índices com range( )

O loop for pode ser combinado com a função range( ) para iterar sobre os índices da lista, como um ninja visitando casas numeradas.

```
frutas = ["banana", "maçã", "laranja", "uva"]

for i in range(len(frutas)):
    print(f"Fruta {i + 1}: {frutas[i]}")  # Exibe
```

Copiando Listas

Você não pode copiar uma lista simplesmente digitando lista2 = lista1, porque: lista2 será apenas uma referência a lista1, e as alterações feitas lista1 também serão feitas automaticamente em lista2.

Existem maneiras de fazer uma cópia, uma maneira é usar o método List integrado copy( ).

Copiando Listas

Faça uma cópia de uma lista com o método copy( ).

```
lista = ["maçã", "banana", "laranja"]
minhalista = lista.copy()
print(minhalista)
```

Dicionários

Dicionários em Python são estruturas de dados poderosas e versáteis que permitem armazenar coleções de dados chave-valor. Eles são perfeitos para organizar e acessar informações de maneira eficiente, sendo utilizados em diversos cenários, como bancos de dados, processamento de linguagem natural e desenvolvimento web.

Criando e Acessando Dicionários

Criando um Dicionário

```
meu_dicionario = {}  # Dicionário vazio
meu_dicionario = {"chave1": "valor1", "chave2": 2}
```

Criando e Acessando Dicionários

Acessando Valores

```
valor = meu_dicionario["chave1"]  # Acessando valor pela chave
print(valor)  # Imprime o valor associado à chave "chave1"
```

Criando e Acessando Dicionários

Adicionando Elementos

```
meu_dicionario["nova_chave"] = "novo_valor"
```

Criando e Acessando Dicionários

Removendo Elementos

```
del meu_dicionario["chave_removida"]
```

Métodos Úteis de Dicionários

- len(dicionario): Retorna o número de pares chave-valor no dicionário.
- keys(dicionario): Retorna uma lista com as chaves do dicionário.
- values(dicionario): Retorna uma lista com os valores do dicionário.

Métodos Úteis de Dicionários

- items(dicionario): Retorna uma lista de tuplas contendo (chave, valor) para cada par do dicionário.
- get(dicionario, chave, valor_padrao): Retorna o valor associado à chave especificada. Se a chave não existir, retorna o valor_padrao.
- in(dicionario, chave): Verifica se a chave existe no dicionário.

Iterando sobre Dicionários

O loop for também funciona com dicionários, permitindo que você acesse chaves ou valores, como um ninja explorando um mapa.

```
pessoa = {"nome": "Maria", "idade": 31, "cidade": "Teresina"}

for chave in pessoa:
    print(f"{chave}: {pessoa[chave]}")  # Exibe chave e valor
```

Usando enumerate( ) para Acessar Índices e Elementos

A função enumerate( ) retorna uma tupla contendo o índice e o elemento em cada iteração, como um ninja recebendo um mapa e a localização de cada ponto de interesse.

```
frutas = ["banana", "maçã", "laranja", "uva"]

for indice, fruta in enumerate(frutas):
    print(f"Posição {indice}: {fruta}")
```

Modificando Elementos Durante a Iteração

O loop for permite modificar elementos da lista enquanto você itera, como um ninja ajustando sua rota conforme explora o caminho.

```
cidades = ["São Paulo", "Rio de Janeiro", "Salvador", "Brasília"]

for indice, cidade in enumerate(cidades):
    if cidade == "Brasília":
        cidades[indice] = "Brasília (Capital)"  # Modificando o nome

print(cidades)  # Exibe: ["São Paulo", "Rio de Janeiro", "Salvador",
```

Indo Além com Variáveis

Já vimos que as variáveis são nossas aliadas na hora de criarmos nossos algoritmos, mas tem algumas coisas sobre ela que talvez você ainda não saiba. E é isso que veremos agora!

Vamos a um exemplo

Variáveis não precisam ser declaradas com nenhum tipo específico e podem até mudar de tipo depois de terem sido definidas.

```
x = 4
x = "Eu Amo Python"
print(x)
```

Dando um tipo a variável

Você pode especificar para o python qual é o tipo da variável que você está declarando.

```
x = str(3)    # x resultado '3'
y = int(3)    # y resultado 3
z = float(3)  # z resultado 3.0
```

Obtendo o tipo da variável

Você pode obter o tipo de dados de uma variável com a função type().

```
x = 5
y = "Lucas"
print(type(x))
print(type(y))
```

Maiúsculas e minúsculas

Os nomes das variáveis diferenciam maiúsculas de minúsculas.

```
a = 4
A = "Python"
```

O exemplo acima cria duas variáveis diferentes

Maiúsculas e minúsculas

Os nomes das variáveis diferenciam maiúsculas de minúsculas.

```
a = 4
A = "Python"
```

O exemplo acima cria duas variáveis diferentes

Nomeando variáveis

Uma variável pode ter um nome curto (como x e y) ou um nome mais descritivo (idade, nome do carro, volume_total). Regras para variáveis Python:

- O nome de uma variável deve começar com uma letra ou sublinhado
- Um nome de variável não pode começar com um número
- Um nome de variável só pode conter caracteres alfanuméricos e sublinhados (Az, 0-9 e _)

Nomeando variáveis

Uma variável pode ter um nome curto (como x e y) ou um nome mais descritivo (idade, nome do carro, volume_total). Regras para variáveis Python:

- Os nomes das variáveis diferenciam maiúsculas de minúsculas (idade, Idade e IDADE são três variáveis diferentes)
- Um nome de variável não pode ser nenhuma das palavras-chave do Python .

Nomeando variáveis

Exemplo de como nomear suas variáveis

```
myvar = "João"
my_var = "João"
_my_var = "João"
myVar = "João"
MYVAR = "João"
myvar2 = "João"
```

Nomeando variáveis

Exemplo de como não declarar suas variáveis

```
2myvar = "João"
my-var = "João"
my var = "João"
```

Vários valores para múltiplas variáveis

Python permite atribuir valores a múltiplas variáveis em uma única linha:

```
x, y, z = "Laranja", "Banana", "Morango"
print(x)
print(y)
print(z)
```

Um valor para múltiplas variáveis

E você pode atribuir o mesmo valor a múltiplas variáveis em uma linha:

```
x = y = z = "Laranja"
print(x)
print(y)
print(z)
```

Descompactando uma Coleção

Se você tiver uma coleção de valores em uma lista, tupla etc. Python permite extrair os valores em variáveis. Isso é chamado de descompactação.

Descompactando uma Coleção

Descompactando uma lista

```
frutas = ["Maçã", "Banana", "Laranja"]
x, y, z = frutas
print(x)
print(y)
print(z)
```

Variáveis globais

Variáveis criadas fora de uma são conhecidas como variáveis globais.

Variáveis globais podem ser usadas por qualquer pessoa, tanto dentro quanto fora das funções.

Convertendo tipo da variável

Você pode converter de um tipo para outro com os métodos int( ), float( ) e complex( ):

OBS: Você não pode converter números complexos em outro tipo de número.

```
x = 1     # int
y = 2.8   # float
z = 1j    # complexo

#converte de int para float:
a = float(x)
```

Números aleatórios

O Python não tem uma função para criar números aleatórios, mas possui um módulo interno chamado Random que pode ser usado para criar números aleatórios:

Números aleatórios

Importe o módulo aleatório e exiba um número aleatório entre 1 e 9:

```
import random

print(random.randrange(1, 10))
```

Manipulando as strings

Python possui um conjunto de métodos integrados que você pode usar em strings.

Maiúsculas

O upper( ) método retorna a string em maiúsculas:

```
a = "Hello, World!"
print(a.upper())
```

Minúsculas

O lower( ) método retorna a string em letras minúsculas:

```
a = "Hello, World!"
print(a.lower())
```

Remover espaço em branco

Espaço em branco é o espaço antes e/ou depois do texto real, e muitas vezes você deseja remover esse espaço.

Remover espaço em branco

O strip( ) método remove qualquer espaço em branco do início ou do fim:

```
a = " Hello, World! "
print(a.strip())
```

Substituir Letra

O replace( ) método substitui uma string por outra string:

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

Sequência dividida

O split( ) método retorna uma lista onde o texto entre o separador especificado se torna os itens da lista.

Sequência dividida

O split( ) método divide a string em substrings se encontrar instâncias do separador:

```
a = "Hello, World!"
print(a.split(","))
```

Por hoje é só...

Hoje vimos muita coisa, vamos descansar que amanhã tem mais!