Tipos de Dados em Python

Tipos de Dados Primitivos

Os tipos simples primitivos em programação são aqueles que representam valores únicos e indivisíveis. Eles são as unidades básicas de armazenamento e manipulação de dados.

Inteiro (int):

Explicação: O tipo de dado inteiro `int` em Python representa números inteiros, positivos ou negativos, sem parte decimal. Inteiros são usados para representar quantidades inteiras, como contadores, índices e números inteiros em geral.

Exemplos:

```
numero_inteiro = 42
outro_inteiro = -20
print(numero_inteiro)  # Saída: 42
print(outro_inteiro)  # Saída: -20
```

Situação Real: Contagem de itens em um estoque.

```
quantidade_produtos = 150
```

Justificativa: Utilizamos um número inteiro para representar a quantidade de produtos no estoque, pois não faz sentido ter uma parte decimal em unidades de produtos.

Ponto Flutuante (float):

Explicação: O tipo de dado ponto flutuante `float` é usado para representar números reais, incluindo a parte decimal. Pode ser utilizado para cálculos que envolvem números fracionários ou decimais.

Exemplos:

```
numero_ponto_flutuante = 3.14
outro_ponto_flutuante = 2.5
print(numero_ponto_flutuante)  # Saída: 3.14
print(outro_ponto_flutuante)  # Saída: 2.5
```

Situação Real: Cálculo de preço total de uma compra.

```
preco_produto = 9.99
quantidade_comprada = 5
```

```
total_compra = preco_produto * quantidade_comprada
```

Justificativa: Usamos um número de ponto flutuante para representar o preço do produto, pois ele pode ter valores decimais. O cálculo do total da compra pode resultar em um número real.

String (str):

Explicação: Strings `str` em Python são sequências de caracteres. Elas são utilizadas para representar texto e são definidas entre aspas simples (' ') ou duplas (" ").

Exemplos:

```
texto_simples = 'Olá, mundo!'
outra_string = "Python é incrível!"
print(texto_simples)  # Saída: Olá, mundo!
print(outra_string)  # Saída: Python é incrível!
```

Situação Real: Armazenamento do nome de um usuário.

```
nome_usuario = "John Doe"
```

Justificativa: Strings são usadas para representar texto, e aqui armazenamos o nome do usuário como uma sequência de caracteres.

Booleano (bool):

Explicação: O tipo de dado booleano `bool` representa valores lógicos: `True` (verdadeiro) ou `False` (falso). É comumente usado em expressões condicionais e operações lógicas.

Exemplos:

```
condicao_verdadeira = 10 > 5
condicao_falsa = 3 == 1
print(condicao_verdadeira)  # Saída: True
print(condicao_falsa)  # Saída: False
```

Situação Real: Verificação da disponibilidade de um item, se estiver disponível adiciona-o ao carrinho de compras.

```
if item.esta_disponivel = True:
    carrinho_compras.adicionar(item)
else:
    tela.notificao_alerta("Item não está disponível")
```

Justificativa: Utilizamos um valor booleano para indicar se um item está disponível (True) ou não (False). É uma representação lógica simples.

Tipos de Dados Compostos

Os tipos compostos são estruturas que permitem armazenar e organizar múltiplos valores. São construídos a partir de tipos simples primitivos. Em Python, alguns dos tipos compostos mais comuns incluem:

Lista (list):

Explicação: Listas `list` são coleções ordenadas e mutáveis de elementos. Podem armazenar diferentes tipos de dados e são acessadas por índices.

Exemplos:

```
lista_numeros = [1, 2, 3, 4, 5]
lista_strings = ['apple', 'banana', 'orange']
print(lista_numeros)  # Saída: [1, 2, 3, 4, 5]
print(lista_strings)  # Saída: ['apple', 'banana', 'orange']
```

Situação Real: Armazenamento de diferentes tipos de produtos em um carrinho de compras.

```
carrinho_compras = ["notebook", "fones", 29.99, True]
```

Justificativa: Listas permitem armazenar uma variedade de tipos de dados, útil para representar os itens diversificados presentes em um carrinho de compras.

Tupla (tuple):

Explicação: Tuplas `tuple` são semelhantes às listas, mas são imutáveis. São usadas para representar coleções ordenadas de elementos que não devem ser modificados.

Exemplos:

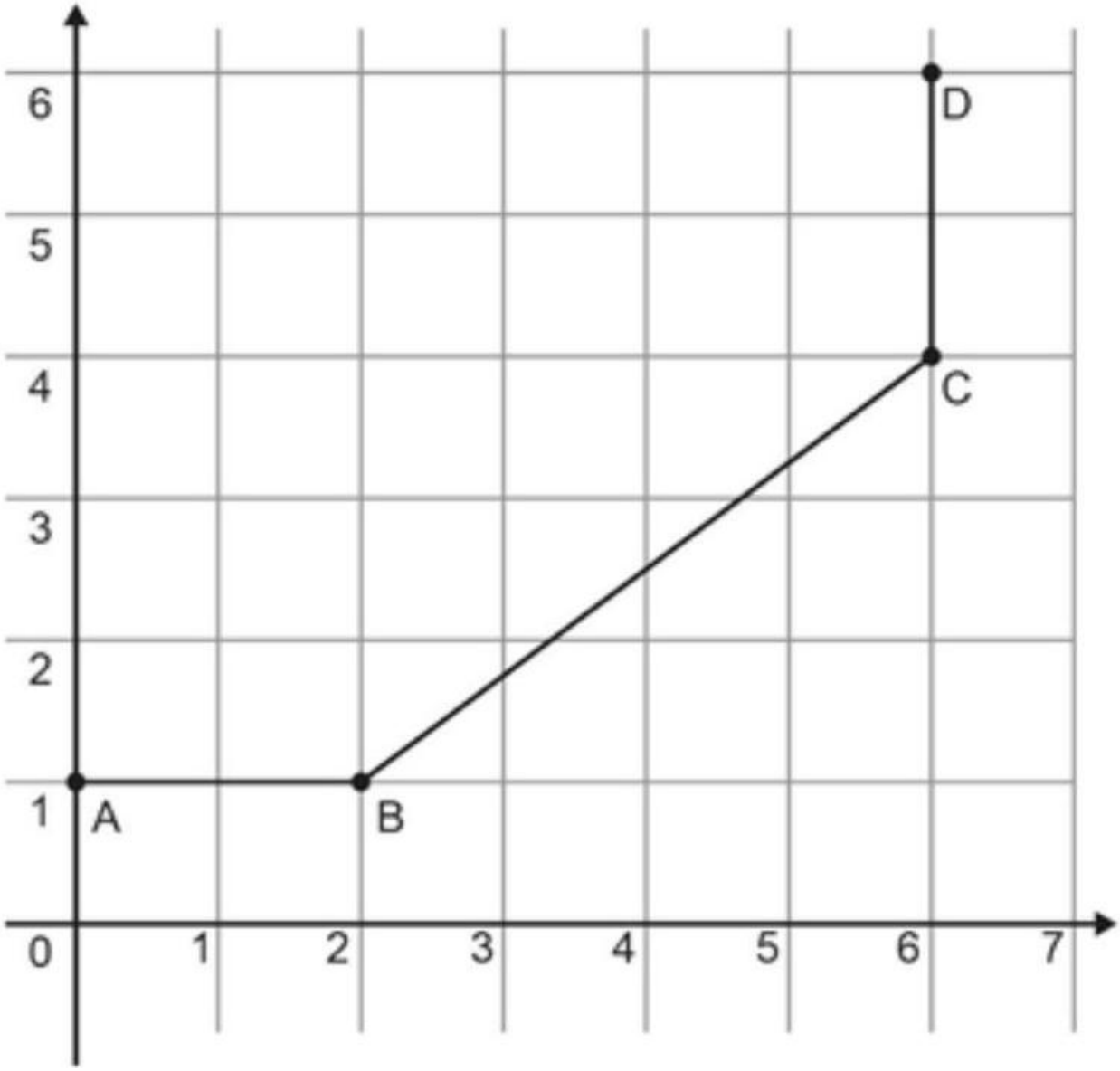
```
tupla_cores = ('vermelho', 'verde', 'azul')
coordenadas = (3.5, -2.8)
print(tupla_cores)  # Saída: ('vermelho', 'verde', 'azul')
print(coordenadas)  # Saída: (3.5, -2.8)
```

Situação Real: Coordenadas fixas de um ponto em um gráfico.

```
ponto_A = (0, 1)
ponto_B = (2, 1)
ponto_C = (6, 4)
```

```
ponto_D = (6, 6)

# Adicionar os segmentos no gráfico
grafico.plot(ponto_A, ponto_B)
grafico.plot(ponto_B, ponto_C)
grafico.plot(ponto_C, ponto_D)
```



Justificativa: Usamos uma tupla para representar as coordenadas de um ponto, pois essas informações não devem ser alteradas durante a execução do programa.

Dicionário (dict):

Explicação: Dicionários `dict` são coleções não ordenadas de pares chave-valor. Cada valor é associado a uma chave, proporcionando uma forma eficiente de acessar dados.

Exemplos:

```
pessoa = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
notas_alunos = {'Alice': 90, 'Bob': 85, 'Charlie': 92}
print(pessoa)  # Saída: {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
print(notas_alunos)  # Saída: {'Alice': 90, 'Bob': 85, 'Charlie': 92}
```

Situação Real: Armazenamento de informações de um cliente.

```
dados_cliente = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
```

Justificativa: Dicionários são ideais para armazenar informações associadas a chaves. Neste caso, temos um dicionário com informações do cliente usando chaves como 'nome', 'idade' e 'cidade'.

Conjunto (set):

Explicação: Conjuntos (set) são coleções não ordenadas e mutáveis de elementos únicos. Eles são úteis para operações de conjunto, como união, interseção e diferença.

Exemplos:

```
conjunto_numeros = {1, 2, 3, 4, 5}
print(conjunto_numeros)  # Saída: {1, 2, 3, 4, 5}

conjunto_cores = {'vermelho', 'verde', 'azul'}
print(conjunto_cores)  # Saída: {'vermelho', 'verde', 'azul'}
```

Situação Real: Representação de tags únicas em um sistema de blogs.

```
tags_artigo = {'python', 'programação', 'iniciantes'}
```

Justificativa: Conjuntos são apropriados quando precisamos de uma coleção de elementos únicos. Aqui, utilizamos um conjunto para armazenar as tags de um artigo, garantindo que não haja repetições.

Iteração e Manipulação de Dados em Tipos Compostos

Vamos explorar como realizar operações comuns de iteração e manipulação de dados em quatro tipos compostos em Python: `list`, `tuple`, `dict` e `set`.

1. Listas (list):

Adicionar Novo Elemento:

```

```

```
lista_frutas = ['maçã', 'banana', 'uva']
lista_frutas.append('morango')  # Adiciona 'morango' ao final da lista
```

Remover Elemento:

```
lista_frutas.remove('banana')  # Remove 'banana' da lista
```

Alterar Elemento Existente:

```
lista_frutas[0] = 'pêssego'  # Substitui 'maçã' por 'pêssego'
```

Unir Duas Listas:

```
lista2 = ['laranja', 'abacaxi']
lista_unida = lista_frutas + lista2  # Une as duas listas
```

2. Tuplas (tuple):

Adicionar Novo Elemento (Não é possível):

Tuplas são imutáveis, então não é possível adicionar novos elementos diretamente.

Remover Elemento (Não é possível):

Tuplas são imutáveis, então não é possível remover elementos após a criação.

Alterar Elemento Existente (Não é possível):

Tuplas são imutáveis, logo, os elementos não podem ser alterados após a criação.

Unir Duas Tuplas:

```
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)
tupla_unida = tupla1 + tupla2  # Concatenação de tuplas
```

3. Dicionários (dict):

Adicionar Novo Elemento:

```
dados_aluno = {'nome': 'Maria', 'idade': 25}
dados_aluno['nota'] = 9.5  # Adiciona 'nota' ao dicionário
```

Remover Elemento:

```
del dados_aluno['idade']  # Remove a chave 'idade' do dicionário
```

Alterar Elemento Existente:

```
dados_aluno['nota'] = 8.0  # Atualiza o valor associado à chave 'nota'
```

Unir Dois Dicionários:

```
dados_extra = {'disciplina': 'Matemática'}
dados_aluno.update(dados_extra)  # Adiciona dados_extra ao dicionário
```

4. Conjuntos (set):

Adicionar Novo Elemento:

```
conjunto_cores = {'vermelho', 'azul'}
conjunto_cores.add('verde')  # Adiciona 'verde' ao conjunto
```

Remover Elemento:

```
conjunto_cores.remove('azul')  # Remove 'azul' do conjunto
```

Alterar Elemento Existente (Não é possível):

Conjuntos são mutáveis, mas não possuem índices, então não é possível alterar elementos específicos.

Unir Dois Conjuntos:

```
conjunto2 = {'amarelo', 'roxo'}
conjunto_unido = conjunto_cores.union(conjunto2)  # União de conjuntos
```

Estas operações são fundamentais para manipular dados em tipos compostos em Python, proporcionando flexibilidade na construção e manipulação de estruturas de dados complexas.