Objetivo Geral

Conhecer os tipos de dados em Python.

Por que usamos tipos?

Os tipos servem para definir as caracteristicas e comportamentos de um valor (objeto) para o interpretador. Por exemplo:

Com esse tipo eu sou capaz de realizar operações matemáticas.

Esse tipo para ser armazenado em memória irá consumir 24 bytes.

Tipos em Python

Os tipos built-in são:

Texto	str
Númerico	int, float, complex
Sequência	list, tuple, range
Mapa	dict
Coleção	set, fronzenset
Booleano	bool
Binário	bytes, bytearray, memoryview

[3]

Números inteiros

Números inteiros são representados pela classe ***int*** e possuem precisão ilimitada. São exemplos válidos de números inteiros:

1, 10, 100, -1, -10, -100…99001823

Números de ponto flutuante

Os números de ponto flutuante são usados para representar os números racionais e sua implementação é feita pela classe ***float***. São exemplos válidos de números de ponto flutuante:

1.5, -10.543, 0.76...999278.002

Booleano

É usado para representar verdadeiro ou falso, e é implementado pela classe ***bool***. Em Python o tipo booleano é uma subclasse de ***int***, uma vez que qualquer número diferente de 0 representa verdadeiro e 0 representa falso. São exemplos válidos de booleanos:

True e False

Strings

Strings ou cadeia de caracteres são usadas para representar valores alfanúmericos, em Python as strings são definidas utilizando a classe ***str***. São exemplos válidos de string:

"Python", 'Python', """Python""", '''Python''', "p"

Objetivo Geral

Entender o que são e como utilizar variáveis e constantes.

Variáveis

Em linguagens de programação podemos definir valores que podem sofrer alterações no decorrer da execução do programa. Esses valores recebem o nome de variáveis, pois eles nascem com um valor e não necessariamente devem permanecer com o mesmo durante a execução do programa.

[2]

```
age = 23
name = 'Guilherme'
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Guilherme e eu tenho 23 ano(s) de idade.


age, name = (23, 'Guilherme')
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Guilherme e eu tenho 23 ano(s) de idade.
```

Alterando os valores

Perceba que não precisamos definir o tipo de dados da variável, o Python faz isso automaticamente para nós. Por isso não podemos simplesmente criar uma variável sem atribuir um valor. Para alterar o valor da variável basta fazer uma atribuição de um novo valor:

```
age = 28
name = 'Guilherme'
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Guilherme e eu tenho 28 ano(s) de idade.

age = 27
name = 'Giovanna'
print(f'Meu nome é {name} e eu tenho {age} ano(s) de idade.')
>>> Meu nome é Giovanna e eu tenho 27 ano(s) de idade.
```

Constantes

Assim como as variáveis, constantes são utilizadas para armazenar valores. Uma constante nasce com um valor e permanece com ele até o final da execução do programa, ou seja, o valor é imutável.

Python não tem constantes

Não existe uma palavra reservada para informar ao interpretador que o valor é constante. Em algumas linguagens por exemplo: Java e C utilizamos *final* e *const*, respectivamente para declarar uma constante.

Em Python usamos a convenção que diz ao programador que a variável é uma constante. Para fazer isso, você deve criar a variável com o nome todo em letras maíusculas:

```
ABS_PATH = '/home/guilherme/Documents/python_course/'
DEBUG = True
STATES = [
    'SP',
    'RJ',
    'MG',
]
AMOUNT = 30.2
```

Boas práticas

- O padrão de nomes deve ser *snake case*.
- Escolher nomes sugestivos.
- Nome de constantes todo em maiúsculo.

Objetivo Geral

Aprender a converter os tipos das variáveis.

Convertendo tipos

Em alguns momentos é necessário será necessário converter o tipo de uma variável para manipular de forma diferente. Por exemplo:

Variáveis do tipo *string*, que armazenam números e precisamos fazer alguma operação matemática com esse valor.

[2]

Inteiro para float

```
preco = 10
print(preco)
>>> 10

preco = float(preco)
print(preco)
>>> 10.0

preco = 10 / 2
print(preco)
>>> 5.0
```

Float para inteiro

```
preco = 10.30
print(preco)
>>> 10.3

preco = int(preco)
print(preco)
>>> 10
```

Conversão por divisão

```
preco = 10
print(preco)
>>> 10

print(preco / 2)
>>> 5.0

print(preco / 2)
>>> 5
```

Numérico para string

```
preco = 10.50
idade = 28

print(str(preco))
>>> 10.5

print(str(idade))
>>> 28

texto = f"idade {idade} preco {preco}"
print(texto)
>>> idade 28 preco 10.5
```

String para número

```
preco = "10.50"
idade = "28"

print(float(preco))
>>> 10.50

print(int(idade))
>>> 28
```

Erro de conversão

```
preco = "python"
print(float(preco))

>>>
Traceback (most recent call last):
  File "main.py", line 3, in <module>
        print(float(preco))
ValueError: could not convert string to float: 'python'
```

Objetivo Geral

Aprender como receber e exibir informações para o usuário.

Função input

A função builtin *input* é utilizada quando queremos ler dados da entrada padrão (teclado). Ela recebe um argumento do tipo string, que é exibido para o usuário na saída padrão (tela). A função lê a entrada, converte para string e retorna o valor.

[2]

Exemplo

```
nome = input("Informe o seu nome: ")
>>> Informe o seu nome: |
```

Função print

A função builtin *print* é utilizada quando queremos exibir dados na saída padrão (tela). Ela recebe um argumento obrigatório do tipo varargs de objetos e 4 argumentos opcionais (sep, end, file e flush). Todos os objetos são convertidos para string, separados por *sep* e terminados por *end*. A string final é exibida para o usuário.

Exemplo

```
nome = "Guilherme"
sobrenome = "Carvalho"

print(nome, sobrenome)
print(nome, sobrenome, end="...\n")
print(nome, sobrenome, sep="#")

>>> Guilherme Carvalho
>>> Guilherme Carvalho...
>>> Guilherme#Carvalho
```

Objetivo Geral

O que são operadores aritméticos e como utilizá-los.

O que são?

Os operadores aritméticos executam operações matemáticas, como adição, subtração com operandos.

[2]

Adição, subtração e multiplicação

```
# Adição
print(1 + 1)
>>> 2

# Subtração
print(10 - 2)
>>> 8

# Multiplicação
print(4 * 3)
>>> 12
```

Divisão e divisão inteira

```
# Divisão
print(12 / 3)
>>> 4.0

# Divisão inteira
print(12 // 2)
>>> 6
```

Módulo e exponenciação

```
# Módulo
print(10 % 3)
>>> 1

# Exponenciação
print(2 ** 3)
>>> 8
```

Na matemática

Na matemática existe uma regra que indica quais operações devem ser executadas primeiro. Isso é útil pois ao analisar uma expressão, a depender da ordem das operações o valor pode ser diferente:

x = 10 - 5 * 2

x é igual a 10 ou 0?

Na matemática

A definição indica a seguinte ordem como a correta:

- Parêntesis
- Expoêntes
- Multiplicações e divisões (da esquerda para a direita)
- Somas e subtrações (da esquerda para a direita)

Exemplo

```
print(10 - 5 * 2)
>>> 0

print((10 - 5) * 2)
>>> 10

print(10 ** 2 * 2)
>>> 200

print(10 ** (2 * 2))
>>> 10000

print(10 / 2 * 4)
>>> 20.0
```

Objetivo Geral

O que são operadores de comparação e como utilizá-los.

O que são?

São operadores utilizados para comparar dois valores.

[2]

Igualdade

```
saldo = 450
saque = 200

print(saldo == saque)
>>> False
```

Diferença

```
saldo = 450
saque = 200
print(saldo != saque)
>>> True
```

Maior que / maior ou igual

```
saldo = 450
saque = 200
print(saldo > saque)
>>> True

print(saldo >= saque)
>>> True
```

Menor que / menor ou igual

```
saldo = 450
saque = 200
print(saldo < saque)
>>> False

print(saldo <= saque)
>>> False
```

Objetivo Geral

O que são operadores de atribuição e como utilizá-los.

O que são?

São operadores utilizados para definir o valor inicial ou sobrescrever o valor de uma variável.

[2]

Atribuição simples

```
saldo = 500

print(saldo)
>>> 500
```

Atribuição com adição

```
saldo = 500
saldo += 200

print(saldo)
>>> 700
```

Atribuição com subtração

```
saldo = 500
saldo -= 100

print(saldo)
>>> 400
```

Atribuição com multiplicação

```
saldo = 500
saldo *= 2

print(saldo)
>>> 1000
```

Atribuição com divisão

```
saldo = 500
saldo /= 5

print(saldo)
>>> 100.0

saldo = 500
saldo //= 5

print(saldo)
>>> 100
```

Atribuição com módulo

```
saldo = 500
saldo %= 480

print(saldo)
>>> 20
```

Atribuição com exponenciação

```
saldo = 80
saldo **= 2

print(saldo)
>>> 6400
```

Objetivo Geral

O que são operadores lógicos e como utilizá-los.

O que são?

São operadores utilizados em conjunto com os operadores de comparação, para montar uma expressão lógica. Quando um operador de comparação é utilizado, o resultado retornado é um booleano, dessa forma podemos combinar operadores de comparação com os operadores lógicos, exemplo:

op_comparacao + op_logico + op_comparacao… N …

[2]

Exemplo

```
saldo = 1000
saque = 200
limite = 100

saldo >= saque
>>> True

saque <= limite
>>> False
```

Operador E

```
saldo = 1000
saque = 200
limite = 100

saldo >= saque and saque <= limite
>>> False
```

Operador OU

```
saldo = 1000
saque = 200
limite = 100

saldo >= saque or saque <= limite
>>> True
```

Operador Negação

```
contatos_emergencia = []

not 1000 > 1500
>>> True

not contatos_emergencia
>>> True

not "saque 1500;"
>>> False

not ""
>>> True
```

Parênteses

```
saldo = 1000
saque = 250
limite = 200
conta_especial = True

saldo >= saque and saque <= limite or conta_especial and saldo >= saque
>>> True

(saldo >= saque and saque <= limite) or (conta_especial and saldo >= saque)
>>> True
```

Objetivo Geral

Aprender como o interpretador Python utiliza a indentação do código para delimitar os blocos de comandos.

A estética

Identar código é uma forma de manter o código fonte mais legível e manutenível. Mas em Python ela exerce um segundo papel, através da indentação o interpretador consegue determinar onde um bloco de comando inicia e onde ele termina.

[2]

Bloco de comando

As linguagens de programação costumam utilizar caracteres ou palavras reservadas para terminar o início e fim do bloco. Em Java e C por exemplo, utilizamos chaves:

[3]

Bloco em Java

```
void sacar(double valor) {  // início do bloco do método

    if (this.saldo >= valor) { // início do bloco do if

        this.saldo -= valor;

    } // fim do bloco do if

}  // fim do bloco do método
```

Bloco em Java sem formatar

```
void sacar(double valor) {  // início do bloco do método
if (this.saldo >= valor) {  // início do bloco do if
this.saldo -= valor;
}  // fim do bloco do if
}  // fim do bloco do método
```

Utilizando espaços

Existe uma convenção em Python, que define as boas práticas para escrita de código na linguagem. Nesse documento é indicado utilizar 4 espaços em branco por nível de indentação, ou seja, a cada novo bloco adicionamos 4 novos espaços em branco.

Bloco em Python

```
def sacar(self, valor: float) -> None:  # início do bloco do método

    if self.saldo >= valor:  # início do bloco do if

        self.saldo -= valor

    # fim do bloco do if

# fim do bloco do método
```

Isso não funciona em Python!

```
def sacar(self, valor: float) -> None:  # início do bloco do método
if self.saldo >= valor:  # início do bloco do if
self.saldo -= valor
# fim do bloco do if
# fim do bloco do método
```

Qual versão é mais fácil de ler?

```
void sacar(double valor) {
if (this.saldo >= valor) {
this.saldo -= valor;}}
```

```
def sacar(self, valor: float) -> None:
    if self.saldo >= valor:
        self.saldo -= valor
```

Objetivo Geral

O que são as estruturas condicionais e como utilizá-las.

O que são?

A estrutura condicional permite o desvio de fluxo de controle, quando determinadas expressões lógicas são atendidas.

[2]

If

Para criar uma estrutura condicional simples, composta por um único desvio, podemos utilizar a palavra reservada if. O comando irá testar a expressão lógica, e em caso de retorno verdadeiro as ações presentes no bloco de código do if serão executadas.

Exemplo

```
saldo = 2000.0
saque = float(input("Informe o valor do saque: "))

if saldo >= saque:
    print("Realizando saque!")

if saldo <= saque:
        print("Saldo insuficiente!")
```

If/else

Para criar uma estrutura condicional com dois desvios, podemos utilizar as palavras reservadas if e else. Como sabemos se a expressão lógica testada no if for verdadeira, então o bloco de código do if será executado. Caso contrário o bloco de código do else será executado.

Exemplo

```
saldo = 2000.0
saque = float(input("Informe o valor do saque: "))

if saldo >= saque:
    print("Realizando saque!")
else:
        print("Saldo insuficiente!")
```

If/elif/else

Em alguns cenários queremos mais de dois desvios, para isso podemos utilizar a palavra reservada elif. O elif é composto por uma nova expressão lógica, que será testada e caso retorne verdadeiro o bloco de código do elif será executado. Não existe um número máximo de elifs que podemos utilizar, porém evite criar grandes estruturas condicionais, pois elas aumentam a complexidade do código.

Exemplo

```
opcao = int(input("Informe uma opção: [1] Sacar \n[2] Extrato: "))

if opcao == 1:
        valor = float(input("Informe a quantia para o saque: "))
        # ...
elif opcao == 2:
        print("Exibindo o extrato...")
else:
        sys.exit("Opção inválida")
```

If aninhado

Podemos criar estruturas condicionais aninhadas, para isso basta adicionar estruturas if/elif/else dentro do bloco de código de estruturas if/elif/else.

Exemplo

```
if conta_normal:
        if saldo >= saque:
                print("Saque realizado com sucesso!")
        elif saque <= (saldo + cheque_especial):
                print("Saque realizado com uso do cheque especial!")
elif conta_universitaria:
        if saldo >= saque:
                print("Saque realizado com sucesso!")
        else:
                print("Saldo insuficiente!")
```

Objetivo Geral

Conhecer as estruturas de repetição for e while e quando utilizá-las.

O que são estruturas de repetição?

São estruturas utilizadas para repetir um trecho de código um determinado número de vezes. Esse número pode ser conhecido previamente ou determinado através de uma expressão lógica.

[2]

Exemplo sem repetição

```
# Receba um número do teclado e exiba os 2 números seguintes

a = int(input("Informe um número inteiro: "))
print(a)

a += 1
print(a)

a += 1
print(a)
```

Exemplo com repetição

```
# Receba um número do teclado e exiba os 2 números seguintes

a = int(input("Informe um número inteiro: "))
print(a)

repita 2 vezes:
    a += 1
    print(a)
```

Comando while

O comando while é usado para repetir um bloco de código várias vezes. Faz sentido usar while quando não sabemos o número exato de vezes que nosso bloco de código deve ser executado.

while

```
opcao = -1

while opcao != 0:
    opcao = int(input("[1] Sacar \n[2] Extrato \n[0] Sair \n: "))

    if opcao == 1:
        print("Sacando...")
    elif opcao == 2:
        print("Exibindo o extrato...")
```