Tipos de Dados

Professora : Mariane Joaquim Melo

Variavéis

Variáveis são contêineres para armazenar valores de dados.

Uma variável é criada no momento em que você atribui um valor a ela.

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

Variáveis

```
x = 5
y = "John"
```

```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

Função Print()

A função print() é uma das funções mais importantes e usadas na linguagem Python.

Sua função é, basicamente, exibir mensagens na tela ou enviá-las para outro dispositivo, como imprimir dentro de arquivos de texto.

Função Print()

Sintaxe da função print():

print(objeto(s), argumentos_de_palavra-chave)

Função Print()

Onde:

objeto(s): Qualquer objeto, em qualquer quantidade. Os mais comuns são strings de texto e variáveis. Independente do tipo, sempre são convertidos em strings antes da impressão.

argumentos_de_palavra-chave: Argumentos opcionais. Permitem controlar como os objetos são separados, o que é impresso no final da linha, se a impressão ocorre em um arquivo, etc.

Função Print()

A forma mais simples de usar a função print() é simplesmente chamá-la sem passar nenhum argumento, como segue:

print()

Desta forma, uma linha em branco será retornada (uma linha com o caractere newline – \n). Esta é uma forma de adicionar espaços verticais ou pular linhas ao exibir a saída de um script.

Função Print()

O uso mais comum da função print() é a exibição de mensagens na tela para orientar o usuário. Para tal, basta passar a mensagem desejada como argumento da função.

```
print('Olá mundo!!')
```

Função Print()

Note que o texto, por se tratar de uma string (cadeia) de caracteres, deve estar envolto em aspas, mas nem sempre será necessário usar aspas. por exemplo, a mensagem pode estar contida em uma variável e ser exibida com print():

```
txt = "Olá mundo!!"
print(txt)
```

Não colocamos aspas em volta da palavra texto, pois se trata do nome da variável que contém a string a ser exibida.

Função Print()

A função print() pode aceitar qualquer número de argumentos posicionais, o que é realmente muito útil para criar mensagens complexas e formatadas. Desta forma, podemos também imprimir mais de um objeto separando-os com vírgulas na função:

```
nome = "mari"
print("Olá " , nome , ", seja bem vindo(a)!")
```

Função Print()

Podemos concatenar strings dentro da função print(), de modo a formar mensagens mais complexas e completas a serem exibidas na saída, usando o operador de concatenação + (não confundir com o operador aritmético de adição!):

```
nome = input("Digite seu nome: ")
print("Olá " + nome + "!. Bem-vindo ao curso de Python!\n")
```

Função Print()

Por padrão, a função print() exibe a mensagem desejada na tela e imprime um caractere de nova linha automaticamente, de modo que o cursor sempre vai para a linha seguinte (quebra de linha).

Porém, às vezes desejamos que o cursor permaneça na mesma linha da mensagem impressa, para continuarmos a mostrar dados sem mudar de linha. Podemos conseguir isso usando o argumento de palavra-chave end, atribuindo uma string vazia a ele.

```
print('Imprime a mensagem e muda de linha')
print('Imprime a mensagem e permanece na linha. ',end='')
print('Continuei na mesma linha!')
```

Função Print()

Print formatado:

```
nome = input("Digite seu nome por favor: ")
idade = input("Digite sua idade: ")
peso = input("Digite seu peso: ")
print("Olá {}, você tem {} anos e você pesa {} kg" .format(nome,
idade, peso))
print("")
```

Tipos de Dados

Python é uma linguagem dinamicamente tipada, o que significa que não é necessário declarar o tipo de variável pois o Interpretador se encarrega disso.

Isso significa também que o tipo da variável poder variar durante a execução do programa.

Tipos de Dados

Os tipos de dados padrão do Python são:

- Inteiro (int)
- Ponto Flutuante ou Decimal (float)
- Tipo Complexo (complex)
- String (str)
- Boolean (bool)
- List (list)
- Tuple
- Dictionary (dic)

Inteiro (int)

O tipo inteiro é um tipo composto por caracteres numéricos (algarismos) inteiros.

É um tipo usado para um número que pode ser escrito sem um componente decimal, podendo ter ou não sinal, isto é: ser positivo ou negativo.

Por exemplo, 21, 4, 0, e −2048 são números inteiros, enquanto 9.75, 1/2, 1.5 não são.

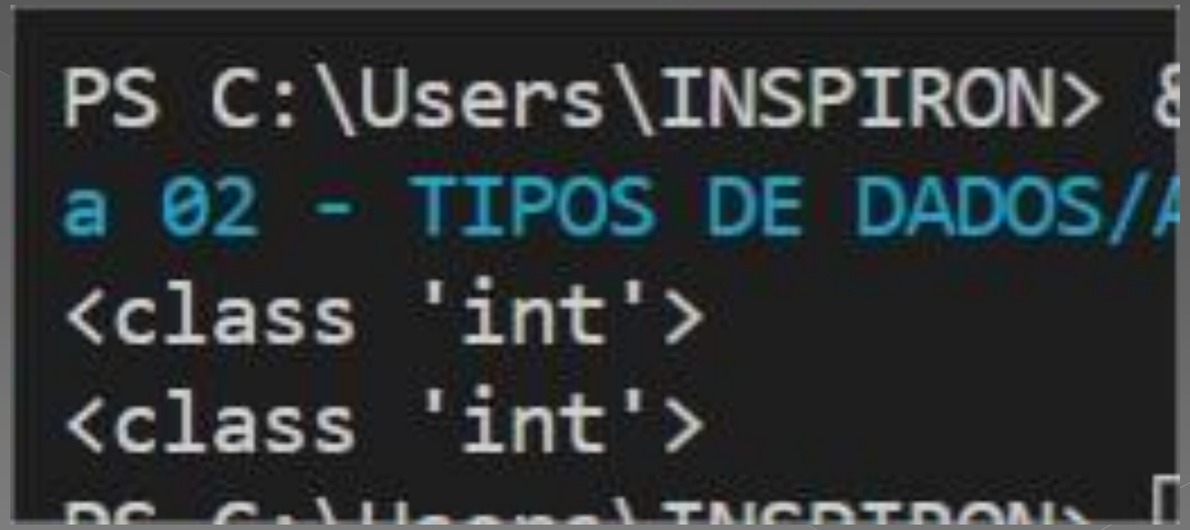
Inteiro (int)

ENTRADA

```
idade = 24
valor = -12

print(type(idade))
print(type(valor))
```

SAÍDA



Ponto Flutuante ou Decimal (float)

É um tipo composto por caracteres numéricos (algarismo) decimais.

O famoso ponto flutuante é um tipo usado para números racionais (números que podem ser representados por uma fração).

Informalmente conhecido como “número quebrado”.

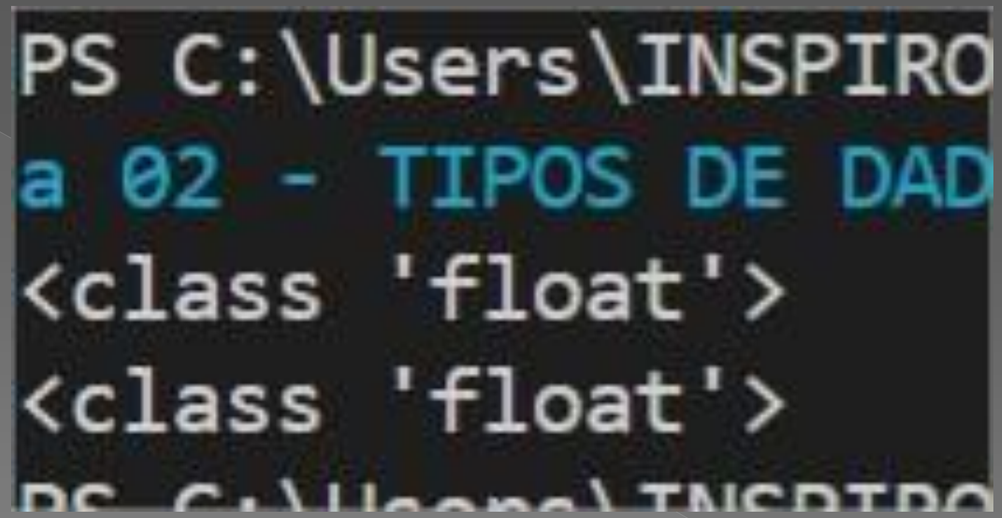
Ponto Flutuante ou Decimal (float)

ENTRADA

```
# FLOAT
altura = 1.56
fracao = 65/6

print(type(altura))
print(type(fracao))
```

SAÍDA



Complexo (complex)

Tipo de dado usado para representar números complexos (isso mesmo, aquilo que provavelmente estudou no terceiro ano do ensino médio).

Esse tipo normalmente é usado em cálculos geométricos e científicos.

Um tipo complexo contem duas partes: a parte real e a parte imaginária, sendo que a parte imaginária contem um j no sufixo.

A função **complex(real[, imag])** do Python possibilita a criação de números imaginários passando como argumento: real, que é a parte Real do número complexo e o argumento opcional imag, representando a parte imaginária do número complexo.

Complexo (complex)

ENTRADA

```
# COMPLEXO
a = 5+2j
b = 20+6j

print(type(a))
print(type(b))
print(complex(2, 5))
```

SAÍDA

```
a 02 - TIPOS DE DADO
<class 'complex'>
<class 'complex'>
(2+5j)
PS C:\Users\INSPIRON
```

String (str)

É um conjunto de caracteres dispostos numa determinada ordem, geralmente utilizada para representar palavras, frases ou textos.

String (str)

ENTRADA

```
# STRING

nome = 'Mariane'
profissao = 'Engenheira da Computação'

print(type(profissao))
print(type(nome))
```

SAÍDA

```
PS C:\Users\INSPIRON>
a 02 - TIPOS DE DADOS/
<class 'str'>
<class 'str'>
PS C:\Users\INSPIRON>
```

Boolean (bool)

Tipo de dado lógico que pode assumir apenas dois valores: falso ou verdadeiro (False ou True em Python).

Na lógica computacional, podem ser considerados como 0 ou 1.

Boolean (bool)

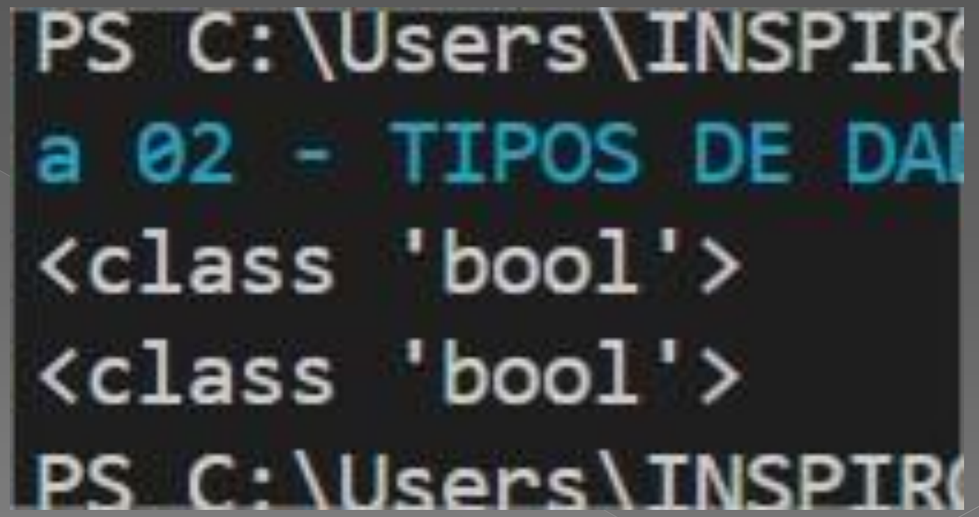
ENTRADA

```
#BOLEANO

fim_de_semana = True
feriado = False

print(type(fim_de_semana))
print(type(feriado))
```

SAÍDA



Listas (list)

Listas agrupam um conjunto de elementos variados, podendo conter: inteiros, floats, strings, outras listas e outros tipos.

Elas são definidas utilizando-se colchetes para delimitar a lista e vírgulas para separar os elementos.

Listas (list)

ENTRADA

```
#LISTAS

alunos = ['Amanda', 'Ana', 'Bruno', 'João']
notas = [10, 8.5, 7.8, 8.0]

print(type(alunos))
print(type(notas))
```

SAÍDA

```
PS C:\Users\INSPIR
a 02 - TIPOS DE DA
<class 'list'>
<class 'list'>
PS C:\Users\INSPIR
```

Tupla (tuple)

Assim como Lista, Tupla é um tipo que agrupa um conjunto de elementos.

Porém sua forma de definição é diferente: utilizamos parênteses e também separado por vírgula.

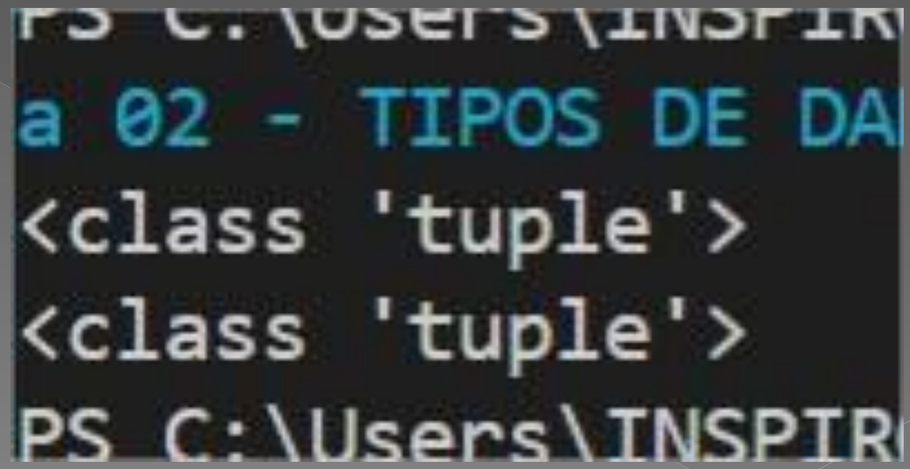
A diferença para Lista é que Tuplas são imutáveis, ou seja, após sua definição, Tuplas não podem ser modificadas.

Tupla (tuple)

ENTRADA

```
3    # TUPLAS
4
5    valores = (90, 79, 54, 32, 21)
6    pontos = (100, 94.05, 86.8, 62)
7
8    print(type(valores))
9    print(type(pontos))
```

SAÍDA



Tupla (tuple)

Caso haja uma tentativa de alterar os itens de uma tupla após sua definição, como no código a seguir:

ENTRADA

```
#tentando mudar os itens de uma tupla
tuple = (0, 1, 2, 3)
tuple[0] = 4
#error
```

SAÍDA

```
Traceback (most recent call last):
  File "c:\Users\INSPIRON\Documents\TÉCNICO INFO 2022-2\LING
 line 53, in <module>
    tuple[0] = 4
TypeError: 'tuple' object does not support item assignment
PS C:\Users\INSPIRON>
```

Dicionários (dict)

Dict é um tipo de dado muito flexível do Python.

Eles são utilizados para agrupar elementos através da estrutura de chave e valor, onde a chave é o primeiro elemento seguido por dois pontos e pelo valor.

Dicionários (dict)

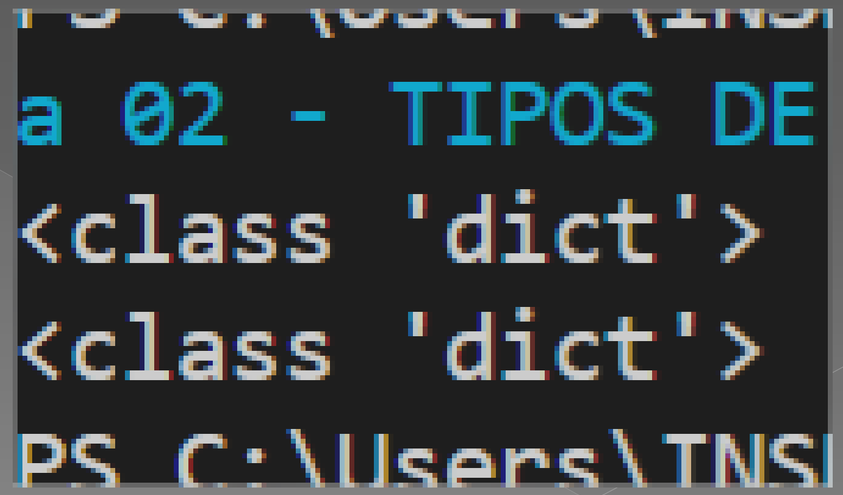
ENTRADA

```
# DICIONÁRIO

altura = {'Amanda': 1.65, 'Ana': 1.60, 'João': 1.70}
peso = {'Amanda': 60, 'Ana': 58, 'João': 68}

print(type(altura))
print(type(peso))
```

SAÍDA



Como mudar o tipo de uma variável

Em determinados cenários pode ser necessário mudar o tipo de uma variável e no Python isso é muito fácil, uma das vantagens de uma linguagem dinamicamente tipada.

Como mudar o tipo de uma variável

ENTRADA

```
# MUDANDO TIPO DE VARIÁVEL

# Antes da conversão
altura = 1.80
print(type(altura))

# Conversão do tipo
altura = str(altura)

# Depois da conversão
print(type(altura))
print(altura)
```

SAÍDA

```
PS C:\Users\INSPI
a 02 - TIPOS DE D
<class 'float'>
<class 'str'>
1.8
PS C:\Users\INSPI
```

Alguns métodos de formatação/verificação de string

```
n = input("Digite algo: ")

print(n.isnumeric()) #verifica se é numérico
print(n.isalpha()) #verifica se é letra
print(n.isspace()) #se é só espaço
print(n.isalnum()) # se é alpha numerico por exemplo 3a
print(n.isupper()) # se esta em letras maiusculas
print(n.islower()) # se esta em letras minuscula
print(n.istitle()) # palavra capitalizada por exemplo, Python
```

Como remover caracteres não alfanuméricos

Para ignorar caracteres não alfanuméricos (como símbolos de pontuação e espaços), você pode usar a função isalpha() para verificar se cada caractere da string é uma letra. Se não for uma letra, você pode ignorá-lo.

```
entrada = input("Digite algo: ")

# Convertendo a entrada para letras minúsculas e removendo caracteres nã
entrada_processada = ''.join(c for c in entrada.lower() if c.isalpha())
```

neste exemplo, a string de entrada é convertida para letras minúsculas e, em seguida, os caracteres não alfanuméricos são removidos usando lista.

1.for c in entrada.lower(): Isso percorre cada caractere (**c**) na versão em letras minúsculas da entrada.
2.if c.isalpha(): Esta é uma condição que verifica se o caractere **c** é uma letra alfabétic (ou seja, uma letra do alfabeto). A função **isalpha()** retorna **True** se o caractere for uma letra e **False** caso contrário.
3.c: Se a condição **c.isalpha()** for verdadeira (ou seja, **c** é uma letra), o caractere **c** é mantido.
4.join(...): A função **join()** é usada para combinar os caracteres mantidos (ou seja, as letras) de volta em uma string. Aqui, estamos usando **''** (uma string vazia) como o separador, o que significa que estamos concatenando os caracteres sem espaços entre eles.

Ordem de procedência

1º - () (parênteses)
2º - ** (potência)
3º - * , / , // , %
4º - + , -

Exercício:

1 Faça um programa que leia um valor em metros e o exiba convertido em centímetros e milímetros.
2 Faça um programa que leia quanto dinheiro uma pessoa tem na carteira e mostre quantos Dólares ela pode comprar.
Considere: US$ 1.00 = R$ 5.41
3 Faça um algoritmo que leia o preço de um produto e mostre seu novo preço, com 5% de desconto.

CONDICIONAIS

Professora : Mariane Joaquim Melo

Condicionais

Ao escrever o código em qualquer linguagem, você terá que controlar o fluxo do seu programa. Geralmente, esse é o caso quando há uma tomada de decisão envolvida. Você desejará executar uma determinada linha de códigos se uma condição for satisfeita, e um conjunto diferente de códigos, caso não seja.

Em Python, você tem as instruções if , elif e else para esse propósito.

Condicionais

A sintaxe é:

```
if (condição):
    Bloco de declaração recuado
```

O bloco de linhas recuadas com o mesmo valor após os dois pontos (:) será executado sempre que a condição for TRUE(verdade).

Condicionais

Operadores Relacionais que podemos usar com if:

a==b igual
a!=b diferente
a<b menor
a<=b menor igual
a>b maior
a>=b maior igual
and ... or

Condicionais

Sintaxe com else:

```
if(condição):
    Bloco de declaração recuado para quando a condição for TRUE
else:
    Bloco de declaração recuado para quando a condição for FALSE
```

Condicionais

```
if(condição):
    Bloco de declaração recuado para quando a condição for TRUE
elif(condição2):
    Bloco de declaração recuado para quando a condição2 for TRUE
else:
    Bloco de instrução alternativo se todas as verificações de condição acima falharem
```

Operador ternário

Em Python, um operador ternário é uma forma concisa de escrever uma expressão condicional em uma única linha. Ele permite que você avalie uma expressão e retorne um valor com base em uma condição. O operador ternário é frequentemente usado quando você deseja atribuir um valor a uma variável com base em uma condição.

sintaxe:
<valor_se_verdadeiro> if <condição> else <valor_se_falso>

Operador ternário

Ex:

```
idade = 18
mensagem = "Maior de idade" if idade >= 18 else "Menor de idade"
print(mensagem)
```

Exercício:

Faça um programa que receba a altura e o sexo de uma pessoa e calcule e mostre seu peso ideal, utilizando as seguintes formulas (onde h corresponde à altura):

- Homens: $(72.7 * h) - 58$
- Mulheres: $(62, 1 * h) - 44, 7$

Exercício:

imagine que você deseja manter um registro de vários cursos, por exemplo: 'Geografia' e 'Inglês'. A pontuação total para ambos ainda é a mesma, ou seja, 100. No entanto, para Geografia, a divisão para teoria e prática é de 50-50, mas para Inglês a divisão é de 60-40. Como você resolveria?

OBRIGADO!

Mariane Joaquim Melo
Linguagem de Programação - SATC