# Python Básico II

**Programação Estruturada de Computadores (PEC)**

Professor: Ritomar Torquato

1

# Python Básico II

Objetivo: Reconhecer os tipos de dados e operadores do Python; Criação de Identificadores; Princípio Básico da Computação; Erros e exceções.

2

# Tipos de dados

- São categorias de valores processados de forma semelhante:
  - Por exemplo, números inteiros são processados de forma diferente dos números de ponto flutuante (com casa decimais) e dos números complexos.

3

# Tipos de dados

- Tipos primitivos: são aqueles já embutidos no núcleo da linguagem
  - **Simples**: numéricos (int, float, bool e complex) e cadeias de caracteres (strings);
  - **Compostos** : São tipos (classes) que o Python já traz prontos para uso, por exemplo: listas, dicionários, tuplas, conjuntos, etc.;
- Tipos definidos pelo usuário: são classes criados pelo programador (orientação objeto).

4

# Tipos Numéricos

- Tipos numéricos Inteiros:
  - **Inteiros (int)**: número sem fração de tamanho ilimitado, sujeito apenas a disponibilidade de memória virtual.
    - -30, -5, 0, 1 , 2 , 15 , 19
  - **Boleanos (bool)**: False e True.
    - igual a zero: False;
    - diferente de zero: True.

5

# Tipos Numéricos

- Tipo Real:
  - **Ponto Flutuante(float)**: número real de precisão dupla.
    - -30.6, -5.0, 0.0, 1.262 , 2.6 , 15.0 , 19.298376

6

## Strings

- São cadeias de caracteres
- Constantes string são escritas usando aspas simples ou duplas
  - Exemplo: "a" ou 'a'
- O operador "+" pode ser usado para concatenar strings
  - Exemplo: "a"+"b" é o mesmo que "ab"
- O operador "*" pode ser usado para repetir strings
  - Exemplo: "ab"*3 é o mesmo que "abababab"

7

## Strings

- Podemos entender uma string como uma sequencia de blocos, onde cada letra ocupa uma posição identificada por um número, iniciando do zero:

| L | A | R | A | N | J | A |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

8

## Strings

- Uma String é uma sequencia de letras endereçadas de tal forma que você possa requisitar qualquer uma delas.

```
>>> palavra = "laranja"
>>> palavra[2]
'r'
>>> palavra[2]*3
'rrr'
```

9

## Strings

- Você também pode solicitar um intervalo de uma sequencia, por exemplo, para solicitar o intervalo de 3 a 7:

```
>>> palavra = "laranja"
>>> palavra[3:7]
'anja'
```

**Importante**: O intervalo selecionado é ABERTO no final, sendo assim, o último caractere não é retornado.

10

## Strings

- Uma String é uma sequencia imutável, não pode ter parte do seu valor alterado, como mostrado abaixo:

```
>>> TesteString = "Sistemas"
>>> print (TesteString)
Sistemas
>>> TesteString[3] = "T"
Traceback (most recent call last):
  File "<pyshell#46>", line 1, in <module>
    TesteString[3] = "T"
TypeError: 'str' object does not support item
assignment
```

11

## Strings

- Python 3 usa a tabela de caracteres default do S.O. Exemplo: ASCII, UTF8
- Caracteres não imprimíveis podem ser expressos usando notação escape ou "barra invertida" (\)
  - \n é o mesmo que nova linha
  - \r é o mesmo que retorno de carro
  - \t é o mesmo que tabulação
  - \b é o mesmo que backspace
  - \\ é o mesmo que \
  - \x41 é o mesmo que o caractere cujo código hexadecimal é 41 ("A" maiúsculo)

12

## O tipo de uma Variável

- Usar a função type()

```
>>> a = 10
>>> type (a)
<class 'int'>
>>> s = "IFPI"
>>> type (s)
<class 'str'>
>>> b = False
>>> type (b)
<class 'bool'>
```

13

## Tipagem Dinâmica

- Em Python, toda variável tem um tipo. Com isso, o computador sabe quais operações são permitidas
- Tipos são definidos dinamicamente, pelo próprio Python
  - Não é preciso dizer de que tipo é cada variável.
  - O tipo pode mudar durante a execução.

14

## Tipagem Dinâmica

```
>>> nome = 'Maria'
>>> type(nome)
<class 'str'>
>>> nome = 123
>>> type(nome)
<class 'int'>
>>> nome = 1.25
>>> type(nome)
<class 'float'>
>>> nome = True
>>> type(nome)
<class 'bool'>
```

**O identificador "nome" foi usado para os diferentes tipos durante a execução.**

15

## Tipagem Forte

- Uma vez que uma variável tenha um valor de um tipo, ele não pode ser usado como se fosse de outro tipo:

```
>>> a = 1
>>> b = '2'
>>> a + b
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    a + b
TypeError: unsupported operand type(s) for +:
'int' and 'str'
```

16

## Identificadores

- Usamos um **IDENTIFICADOR** para dar nome a variáveis, funções, módulos, etc.,
- Devemos observar 3 (três) regras básicas:
  1. O primeiro caractere deve ser uma **letra** ou um **sublinhado** ( _ )
  2. Os demais caracteres devem ser uma **letra**, um **número** ou um **sublinhado** ( _ )
  3. Não é possível usar **palavras reservadas** da linguagem.

17

## Identificadores

- **Considere ainda**:
  - Letra são os caracteres de A-Z maiúsculos ou a-z minúsculos.
  - Números são os caracteres de 0-9.
  - Caracteres acentuados e cedilha são permitidos mas é melhor evitar.
  - Diferencia minúsculas/MAIÚSCULAS

18

## Identificadores

- Identificadores **válidos**:
  - a
  - _ab
  - _1
  - minha_variavel
  - print
  - For
  - umGrande
  - dois
  - Quatro
  - true
  - false

19

## Identificadores

- Identificadores **inválidos**:
  - 1a
  - a b
  - 1#
  - def
  - minha variavel
  - for
  - if
  - and
  - 4mor
  - True
  - False

20

## Convenção de Nomes

| Tipo | Convenção | Exemplo |
|---|---|---|
| Variável | Única letra minúscula, palavra ou palavras minúsculas separadas por sublinhados. | n, nome, nome_cliente |
| Constante | Única letra maiúscula, palavra ou palavras maiúsculas separadas por sublinhados. | CONSTANTE, MINHA_CONSTANTE, ID_MUITO_LONGO |
| Função | Palavra ou palavras minúsculas separadas por sublinhados. | funcao, minha_funcao |

https://www.python.org/dev/peps/pep-0008/#naming-conventions

21

## Operadores

- Atuam sobre operandos e produzem um resultado. Exemplo: 3 + 2
  - Os números (3 e 2) são os operandos
  - O operador (+) que representa adição

22

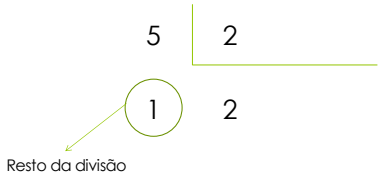
## Operadores Aritméticos

| Operador | Função |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão (Real) |
| // | Divisão (Inteira) |
| ** | Exponenciação |
| % | Módulo (Resto da Divisão) |

23

## Operadores Aritméticos

- Operador Módulo




24

## Operadores Aritméticos

- Operador Módulo: retorna o resto da divisão inteira. Por exemplo:
  - `num_int = 11 % 2`
  - `num_float = 8.5 % 3.2`
- Resulta em:
  - `1 # num_int`
  - `2.0999999999999996 # num_float`

25

## Operadores Aritméticos

- Quando usar a divisão inteira e o módulo?
  - R. Sempre que desejar separar partes inteiras de frações.

Partes inteiras não necessariamente são números inteiros.

| 1/3 | 1/3 | 1/3 |
|---|---|---|

26

## Operadores Aritméticos

Uma partida de futebol é disputada em dois tempos de 45 minutos com um intervalo de 15 minutos entre os tempo. Quantas horas e quantos minutos se passam do início ao final de uma partida?

```
>>> minutos_total = 45 + 15 + 45
>>> h = minutos_total // 60
>>> m = minutos_total % 60
>>> f'A partida tem {h} horas e {m} minutos.'
'A partida tem 1 horas e 45 minutos.'
```

1 hora = 60 min

27

## Operadores Aritméticos

Pedro deseja pagar uma conta de R$9,75 com cédulas de R$2,00 e moedas de R$0,25. Qual a menor quantidade de cédulas e moedas ele deve usar?

```
>>> total = 9.75
>>> c = total // 2
>>> total = total % 2
>>> m = total // 0.25
>>> total = total % 0.25
>>> f'{c:.0f} de R$2,00 e {m:.0f} de R$0.25'
'4 de R$2,00 e 7 de R$0.25'
```

28

## Operadores Aritméticos


**Uma moeda de R$0,25 é um inteiro e não apenas uma parte.**

29

## Operador de Atribuição

- Usado para definir o conteúdo de uma variável
- Em Python, o operador de atribuição é representado pelo símbolo "=" (igual), a sintaxe básica é:
  - `variavel = valor ou expressão`

Leia: variável recebe valor ou expressão
O lado direito do ( = ) é processado
O valor gerado é atribuído à variável

30

## Operador de Atribuição

- Exemplos de atribuição em Python:

```
# a variável ch recebe o valor 'a'
ch = 'a'
# a variável b recebe o valor falso
b = False
# a variável s recebe o valor '155'
s = '155'
# a variável i recebe o valor 100
i = 100
```

31

## Operador de Atribuição

- Exemplos de atribuição em Python:

```
# atribui o mesmo valor em várias
# variáveis
a = b = c = 0
# atribui em várias variáveis
y, z, r = 9.2, -7.6, 10
# deleta uma variável
del x
```

32

## Atribuição Composta

| Operador | Função |
|---|---|
| += | Atribuição com adição |
| -= | Atribuição com subtração |
| *= | Atribuição com multiplicação |
| /= | Atribuição com divisão |
| //= | Atribuição com divisão inteira |
| %= | Atribuição com módulo ou resto |

33

## Atribuição Composta

- Em resumo, os operadores aritméticos possuem um operador de atribuição correspondente:
  - Exemplo: A = A + 2 ↔ A += 2
- Outros Exemplos

| Expressão | Forma compacta |
|---|---|
| $x = x + y$ | $x \mathrel{+}= y$ |
| $x = x - y$ | $x \mathrel{-}= y$ |
| $x = x * y$ | $x \mathrel{*}= y$ |
| $x = x / y$ | $x \mathrel{/}= y$ |
| $x = x \% y$ | $x \mathrel{\%}= y$ |

34

## Operadores Relacionais

Fazem uma comparação que resulta em verdadeiro (True) ou falso (False)

| Operador | Função |
|---|---|
| == | Igual |
| != | Diferente |
| > | Maior que |
| < | Menor que |
| >= | Maior ou igual a |
| <= | Menor ou igual a |

35

## Operadores Relacionais

```
a = 10
b = 12
c = 10

b1 = (a == b)
b2 = (a == c)
b3 = (a != b)

print("a =", a)          # a = 10
print("b =", b)          # b = 12
print("c =", c)          # c = 10

print("a == b :", b1)    # a == b : False
print("a == c :", b2)    # a == c : True
print("a != b :", b3)    # a != b : True
```

36

## Operadores Relacionais

```
a = 25
b = 25.0
c = -37

b1 = (a > b)
b2 = (a >= c)
b3 = (a <= b)

print("a =", a)          # a = 25
print("b =", b)          # b = 25.0
print("c =", c)          # c = -37

print("a > b :", b1)     # a > b : False
print("a >= c :", b2)    # a >= c : True
print("a <= b :", b3)    # a <= b : True
```

37

## Operadores Lógicos

Criam expressões com operandos lógicos que resultam em verdadeiro (True) ou falso (False)

| Operador | Função |
|---|---|
| and | E lógico |
| or | OU lógico |
| not | NÃO lógico |

38

## Operadores Lógicos

- Tabela Verdade

| E (and) | V | F |
|---|---|---|
| V | V | F |
| F | F | F |

| OU (or) | V | F |
|---|---|---|
| V | V | V |
| F | V | F |

| NÃO (not) | V | F |
|---|---|---|
| - | F | V |

39

## Operadores Lógicos

```
x = True
y = False
z = True


b1 = (x and y)
b2 = (x and z)
b3 = (x or z)

print("x =", x)          # x = True
print("y =", y)          # y = False
print("z =", z)          # z = True

print("x and y :", b1) # x and y : False
print("x and z :", b2) # x and z : True
print("x or z  :", b3) # x or z  : True
```

40

## Operadores Lógicos

- Criando expressões lógicas

```
>>> nome = "Maria"
>>> idade = 30
>>> nome == "Maria" and idade == 30
True
>>> nome == "Maria" and idade != 30
False
```

41

## Prioridade dos Operadores

- primeiro as operações aritméticas, depois as de comparação, por fim as lógicas

| Nível | Categoria | Operadores |
|---|---|---|
| 7 (alto) | exponenciação | ** |
| 6 | multiplicação | * / // % |
| 5 | adição | + - |
| 4 | relação | == != <= >= > < |
| 3 | negação | not |
| 2 | conjunção | and |
| 1 (baixo) | disjunção | or |

42

## Princípio da Computação

- Se quisermos que o computador resolva um problema qualquer:


Dados necessários | Execução do Algoritmo | Informação para o Usuário

43

## Princípio da Computação

- Se quisermos que o computador resolva um problema qualquer:

```
# Entrada de dados
a = int(input('Valor de A: '))
b = int(input('Valor de B: '))

# Processamento
soma = a + b

# Saída de Dados
print(f'A soma de {a} com {b} é {soma}')
```

44

## Erros e exceções

Temos basicamente 3 (três) tipos de erros:

- Erro de sintaxe
- Erro de exceção
- Erro de lógica ou semântica

45

## Erros e exceções

- Erro de sintaxe
  - Há uma falha na tradução do algoritmo para a linguagem Python
  - O interpretador vai detectar e dar dicas apontando o local do erro
  - São mais fáceis de corrigir

46

## Erros e exceções

- Erro de sintaxe

O parser apresenta uma pequena 'seta'ou 'destaque colorido' apontando para o ponto da linha em que o erro foi detectado.

```
>>> a =! b
SyntaxError: invalid syntax
```

Neste caso, não existe o operador =!
O correto seria usar !=

47

## Erros e exceções

- Erro de exceção
  - Comandos sintaticamente corretos, podem causar um erro na hora de sua execução
  - Erros detectados durante a execução são chamados exceções.
  - A maioria das exceções não são tratadas pelos programas e acabam resultando em mensagens de erro.

48

## Erros e exceções

- **TypeError**: Exceção gerada quando uma operação ou função é aplicada a um objeto do tipo inadequado.

```
>>> '2' + 2
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    '2' + 2
TypeError: can only concatenate str (not "int") to str
```

49

## Erros e exceções

- **ZeroDivisionError**: Exceção gerada quando o segundo argumento de uma operação de divisão ou módulo é zero.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    10 * (1/0)
ZeroDivisionError: division by zero
```

50

## Erros e exceções

- **NameError**: Exceção gerada quando um nome (identificador) local ou global não é encontrado.

```
>>> 4 + minha_variavel * 3
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    4 + minha_variavel * 3
NameError: name 'minha_variavel' is not defined
```

51

## Erros e exceções

- **ValueError**: Exceção gerada quando uma operação ou função recebe um argumento (parâmetro) que tem o tipo certo, mas um valor inapropriado.

```
>>> x = input()
d
>>> int(x)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    int(x)
ValueError: invalid literal for int() with base 10: 'd'
```

52

## Erros e exceções

Exceções não são necessariamente fatais, ou seja, é possível trata-las para o programa continuar executando. Vejamos...

53

## Tratamento de Exceções

- Tratando erros de Entrada de Dados

```
try:

    idade = int(input("Digite sua idade: "))
    print("Sua idade em 5 anos será:", idade + 5)

except:

    print("Você não digitou um valor numérico.")
```

54

## Tratamento de Exceções

- Tratando erros de Entrada de Dados

```
try:
    idade = int(input("Digite sua idade: "))
    print("Sua idade em 5 anos será:", idade + 5)
except:
    print("Você não digitou um valor numérico.")
```

Tenta fazer tudo que estiver neste alinhamento (endentação).

55

## Tratamento de Exceções

- Tratando erros de Entrada de Dados

```
try:
    idade = int(input("Digite sua idade: "))
    print("Sua idade em 5 anos será:", idade + 5)
except:
    print("Você não digitou um valor numérico.")
```

Se algo der errado, executa essas linhas.

56

## Tratamento de Exceções

- Tratando erros de Entrada de Dados

```
>>>
Digite sua idade: 20
Sua idade em 5 anos será: 25
>>>
Digite sua idade: Vinte
Você não digitou um valor numérico.
```

Mais sobre erros e exceções pode ser visto nesses links:
https://docs.python.org/pt-br/3/library/exceptions.html
https://docs.python.org/pt-br/3/tutorial/errors.html

57

## Erros e exceções

- Erro de lógica ou semântico
  - Para o computador não representam necessariamente um erro, ele só faz exatamente aquilo que é mandado fazer.
  - Nenhuma mensagem é mostrada. O resultados obtido é diferente do esperado.
  - São os erros mais difíceis de serem identificados. Normalmente é preciso fazer a depuração (ou Teste de Mesa)

58

## Erros e exceções

- Erro de lógica ou semântico

```
# Entrada de dados
a = int(input('Valor de A: '))

# Processamento
dobro = a + 2

# Saída de Dados
print(f'O dobro de {a} é {dobro}')
```

Se for lido o valor 2 o programa apresenta o resultado certo.

59