

# Revisão de Python

**Prof. Saulo Oliveira**  
**Técnico em Informática para Internet**  
**Instituto Federal do Ceará**

# Introdução

O algoritmo é uma sequência de passos lógicos e finitos que permite solucionar problemas;

- O objetivo de aprender a criar algoritmos é que este é a base de conhecimentos para as linguagens de programação;
- Em geral, existem muitas maneiras de resolver o mesmo problema. Ou seja, podem ser criados vários algoritmos diferentes para resolver o mesmo problema;
- Assim, ao criarmos um algoritmo, indicamos uma dentre várias possíveis sequências de passos para solucionar o problema.

# Propriedades essenciais

Completo	Sem redundância	Determinístico	Finito
Todas as ações precisam ser descritas e devem ser únicas.	Um conjunto de instruções só pode ter uma única forma de ser interpretada.	Se as instruções forem executadas, o resultado esperado será sempre atingido.	As instruções precisam terminar após um número limitado de passos.

# Instruções e Tipos de dados

As informações manipuladas pelo computador podem ser classificadas em:

Instruções	Dados
Coordenam o funcionamento do computador, determinando a maneira como os dados devem ser tratados.	São as informações a serem processadas pelo computador.

Em Python, os dados podem ser dos tipos: numéricos ( `int` e `float` ), lógicos ( `bool` ), literais ( `string` ), listas ( `list` ) e dicionários ( `dict` ). Há outros, mas deixemos para depois!

# Exercício

Classifique os dados abaixo de acordo com seu tipo, assinalando com **I** os dados do tipo inteiro, com **R** os reais, com **L** os literais, com **B** os lógicos (booleanos), e com **N** aqueles para os quais não é possível definir a priori um tipo de dado.

	0.21		"josé"		"+3257"		True
	1		0,35		+3257.		False.
	V		TRUE		"-0.0"		"abc"
	"0"		+3257		"False"		+36
	1%		'a'		True		± 3

# Declaração de variáveis

Em Python é necessário apenas o nome da variável, seguido do símbolo = e o valor que ela irá armazenar. O tipo da variável será o mesmo tipo de dado que ela armazena.

```
idade = 32
preco = 100.21
teste = True
nome = "SAULO"
```

Regras de nomenclatura (o nome das variáveis):

- Podem ter dígitos, letras maiúsculas e minúsculas, e underscore (\_);
- Não pode ser iniciado por dígito e não são permitidos espaços em branco e nem podem conter caracteres especiais (@, \$, +, -, %, !, /, ?, #);
- Nem palavras reservadas ( `keywords` ).

# Saída de dados

Usaremos a função `print`.

- A função `print` mostra em formato texto para o usuário o conteúdo de um variável;
- Também pode mostrar strings ou combinações de strings e variáveis, bastando separar por (,) vírgula.
- Podemos usar o `print` com valores dos tipos primitivos;
- Podemos executar expressões e só o valor do resultado vai ser impresso.

```
idade = 32
nome = 'Saulo'
print('Meu primeiro programa')
print(f'Meu nome é {nome} e minha idade é:', idade)
print('Faltam', 65 - idade, 'anos para eu me aposentar!')
```

# Entrada de dados

Usaremos a função `input`.

- A função `input` requer um texto que será mostrado para o usuário e retorna o que o usuário digitou (sempre do tipo literal `str`, precisamos converter depois, se necessário).
- Guardamos o valor retornado pela função `input` em uma variável.

```
nome = input('Digite seu nome: ')\nprint(f'Seja muito bem-vindo(a), {nome}!')\n\nidade = int(input('Digite sua idade: '))\nprint(f'Legal, que você tem {idade} anos!')\n\naltura = float(input('Digite sua altura: '))\nprint(f'E com altura de {altura} metros.')
```



## Entrada de dados descasada

```
idade = input('Me diga novamente quantos anos tem:')  
proxima_idade = idade + 1  
print(f'Ano que vem, você terá {proxima_idade} anos')
```

**QUAL O PROBLEMA COM O  
CÓDIGO ACIMA?**

# Conversão de tipos

A conversão de tipos é o ato de forçar uma expressão a utilizar e retornar um determinado tipo. Podemos ter dois tipos de conversões de tipos, pode ser implícita ou explícitas, que são conversões especificadas.

Para saber o tipo de dado de uma expressão ou variável, basta usarmos a função `type`.

# Exercício

Declaração	Tipo da variável	Conversão	Resultado	Tipo do resultado
<code>x = "42"</code>		<code>int(x)</code>		
<code>n = 123</code>		<code>str(n)</code>		
<code>pi = 3.14</code>		<code>int(pi)</code>		
<code>saldo = 567</code>		<code>float(saldo)</code>		
<code>tem_pix = True</code>		<code>str(tem_pix)</code>		
<code>poupanca = 0.0</code>		<code>bool(poupanca)</code>		
<code>nome = 'Saulo'</code>		<code>bool(nome)</code>		
<code>cpf = ''</code>		<code>bool(cpf)</code>		
<code>faltei = False</code>		<code>int(faltei)</code>		
<code>merenda = True</code>		<code>float(merenda)</code>		

## Exercício prático

1. Faça um algoritmo para converter uma temperatura dada em Fahrenheit para Celsius.
2. Faça um algoritmo que receba duas notas e seus respectivos pesos, calcule e mostre a média ponderada.
3. Faça um algoritmo que receba um valor referente a uma compra em dólar no cartão de crédito, calcule e mostre o valor de conversão para real. Além disso, sabendo que em compras internacionais incide-se O IOF sobre o total, adicione o valor da taxa (valor de 6,38%). Ademais, adote o valor do dólar R\$ 5,00.

# Operadores e Expressões - I

## Operadores aritméticos.

São operadores matemáticos básicos que envolvem números e são utilizados para realizar cálculos e manipular quantidades numéricas.

Operador	Uso	Comentário
<code>+</code>	<code>x + y</code>	Soma o conteúdo de <code>x</code> e de <code>y</code> .
<code>-</code>	<code>x - y</code>	Subtrai o conteúdo de <code>y</code> do conteúdo de <code>x</code> .
<code>*</code>	<code>x * y</code>	Multiplica o conteúdo de <code>x</code> por pelo conteúdo de <code>y</code> .
<code>/</code>	<code>x / y</code>	Divide o conteúdo de <code>x</code> pelo conteúdo de <code>y</code> .
<code>%</code>	<code>x % y</code>	Obtém o resto da divisão de <code>x</code> por <code>y</code> .
<code>//</code>	<code>x // y</code>	Obtém o quociente inteiro da divisão de <code>x</code> por <code>y</code> .
<code>**</code>	<code>x ** y</code>	Eleva o conteúdo de <code>x</code> à potência do conteúdo de <code>y</code> .

# Operadores e Expressões - II

## Operadores relacionais.

São operadores que comparam valores para determinar relações como igualdade, desigualdade, maior ou menor, retornando um valor lógico ( `True` ou `False` ).

Operador	Uso	Comentário
<code>==</code>	<code>x == y</code>	O conteúdo de <code>x</code> é igual ao conteúdo de <code>y</code> .
<code>!=</code>	<code>x != y</code>	O conteúdo de <code>x</code> é diferente do conteúdo de <code>y</code> .
<code>&lt;=</code>	<code>x &lt;= y</code>	O conteúdo de <code>x</code> é menor ou igual ao conteúdo de <code>y</code> .
<code>&gt;=</code>	<code>x &gt;= y</code>	O conteúdo de <code>x</code> é maior ou igual ao conteúdo de <code>y</code> .
<code>&lt;</code>	<code>x &lt; y</code>	O conteúdo de <code>x</code> é menor que o conteúdo de <code>y</code> .
<code>&gt;</code>	<code>x &gt; y</code>	O conteúdo de <code>x</code> é maior que o conteúdo de <code>y</code> .

# Operadores e Expressões - III

## Operadores Lógicos.

Operadores lógicos permitem a realização de operações lógicas sobre valores booleanos. Esses operadores geralmente são utilizados para tomar decisões condicionais e controlar o fluxo de execução de um programa.

Operador	Uso	Comentário
<code>not</code>	<code>not x</code>	Equivale a modificar o conteúdo de <code>x</code> pela negação.
<code>and</code>	<code>x and y</code>	Retorna <code>True</code> se e somente se <code>x</code> e <code>y</code> forem <code>True</code> . Caso contrário, retorna <code>False</code> .
<code>or</code>	<code>x or y</code>	Retorna <code>False</code> se e somente se <code>x</code> e <code>y</code> forem <code>False</code> . Caso contrário, retorna <code>True</code> .

# Precedência

A precedência determina a ordem em que os operadores são avaliados em uma expressão, quando ela envolve múltiplos operadores.

É possível também alterar a ordem de avaliação usando parênteses.

Operadores	Descrição
**	Exponenciação.
+, -	Operadores unários (modificam o sinal).
*, /, %, //	Produto, divisão, resto da divisão e divisão inteira.
+, -	Adição e subtração.
<=, <, >, >=	Operadores de comparação.
==, !=	Operadores de igualdade.
not, and, or	Operadores lógicos.



# Exercícios

Dada a seguinte expressão:

```
2 - 2 ** 3 * 2
```

1. Adicione um conjunto de parênteses para que a expressão seja avaliada como -12.
2. Agora mova seus parênteses para que a expressão seja avaliada como -62.
3. Mova seus parênteses uma última vez para que a expressão seja avaliada como 0.

# Exercícios

Dada a seguinte expressão:

```
2 / 3 * 4 ** 2
```

- Adicione dois conjuntos de parênteses que deixam o valor da expressão inalterado. Os parênteses não podem incluir a expressão inteira nem um único número.

# QUANTO VALE A EXPRESSÃO ABAIXO?

`-2 ** 2`

# Exercícios

Analise o programa abaixo e, para cada uma das saídas (comandos `print`), detalhe passo a passo como o Python (segundo suas prioridades) resolveria as equações e o resultado final obtido.

```
x = 2
y = 3
z = 0.5

print(x + x * x ** (y * x) / z)

print(not x + z < y or x + x * z >= y and True)

print(x + y == z)
```

# Exercícios

Indique o resultado das expressões mostrando passo a passo a ordem de avaliação, sendo: `x = 6.0`, `y = 2`, `z = 4.0`, `a = 8`, `b = 7.5`, `c = 12`. Indique quando ela não puder ser realizada e informe por qual motivo.

a) `x - y * (a + 1) == z * -c`

b) `x - y * a > c % y`

c) `c % y <= y % c`

d) `(b * 4) >= (a + a * 2) and a >= c ** - 2`

e) `a + 3 > -b + -c`

f) `b + a > c + c and a != c < b != a`

g) `a // c < (b % 2) or (c ** b * 3) < a * 3`

# Condicionais

- Condicionais em lógica de programação são estruturas que permitem a execução de diferentes blocos de código dependendo da avaliação de uma condição específica.
- Essas estruturas são fundamentais para controlar o fluxo de execução de um programa, permitindo que partes específicas do código sejam executadas ou ignoradas com base em condições lógicas;
- Eles avaliam uma expressão booleana e, com base no resultado (verdadeiro ou falso), direcionam o programa para diferentes caminhos de execução.

# Condicionais simples

Permite a execução condicional de um bloco de código.

Avalia-se uma expressão booleana e, se a condição for verdadeira ( `True` ), o bloco de código indentado após o `if` é executado; caso contrário, o bloco é ignorado.

```
idade = int(input('Digite sua idade: '))

if idade > 18:
    print('Você é maior de idade')

saldo = float(input('Digite seu saldo: '))

if saldo > 0:
    print('Você não está liso!')
```

# Exercícios

1. Faça um programa que receba dois números e retorne o valor do maior.
2. Faça um programa que solicite ao usuário dois números e imprima "São iguais" se forem iguais; caso contrário, imprima "São diferentes".
3. Crie um programa que verifica se um número é múltiplo de 3 e de 5 e imprime mensagens correspondentes.
4. Peça ao usuário para digitar uma palavra. Se a palavra é "python", imprima uma mensagem de confirmação ou correção (você errou a senha secreta).
5. Crie um programa em Python que solicite um número ao usuário. Verificar se o número é um múltiplo de 10 e imprima uma mensagem apropriada.
6. Faça um programa que recebe a média final de um aluno e mostra sua situação, que pode ser aprovado ( $M \leq 7$ ), reprovado ( $M < 4$ ) e AF ( $4 \leq M < 7$ ).



# Condicional composto

Já na estrutura composta, existe um bloco de código que executa para cada um dos possíveis resultados ( `True` ou `False` ).

Avalia-se uma expressão booleana e, se a condição for verdadeira ( `True` ), o bloco de código indentado após o `if` é executado; caso contrário, o bloco após o `else` é executado.

```
ano = int(input('Digite o ano: '))

if ano % 2 == 0:
    print('O ano é par!')
else:
    print('O ano é ímpar!')
```

# Exercícios

1. Faça um programa para receber dois números positivos e mostre-os em ordem crescente.
2. Faça um programa para ler dois números inteiros e informar se estes números são iguais ou diferentes.
3. Faça um programa que receba um número e imprima se ele é positivo ou negativo.
4. Solicite ao usuário sua renda mensal e o valor desejado para empréstimo. Imprima se o empréstimo pode ser realizado, sabendo que somente é aprovado caso o valor desejado seja menor do 5x o valor da renda mensal.

# Condicional encadeado

Já na estrutura composta encadeada, somente uma condição pode ocorrer dentro do encadeamento.

Permite avaliar condições adicionais se as anteriores forem falsas. A instrução `elif` é a junção das instruções `else` e `if`. Leia como se fosse *senão se*.

```
nota = float(input('Digite sua nota na disciplina'))

if nota >= 7:
    print('Aprovado.')
elif nota < 4:
    print('Reprovado.')
else:
    print('Prova final.')
```

# Exercícios

Faça um algoritmo que mostre a classificação do IMC de um indivíduo. O peso é dado em quilogramas (Kg) e a altura em metros (m), conforme a fórmula que segue:

$$\text{IMC} = \frac{\text{peso}}{\text{altura}^2}$$

IMC	Classificação do IMC
< 16	Magreza grave
16 a < 17	Magreza moderada
17 a < 18,5	Magreza leve
18,5 a < 25	Saudável
25 a < 30	Sobrepeso
30 a < 35	Obesidade Grau I
35 a < 40	Obesidade Grau II (severa)
> 40	Obesidade Grau III (mórbida)

# Estruturas de repetição

Essas estruturas são fundamentais para automatizar tarefas repetitivas e controlar o fluxo de execução de um programa.

As repetições **podem ser de uma quantidade fixa ou depender de uma condição.**

Comando `for` é utilizado quando o número exato de iterações é conhecido antecipadamente. Geralmente, emprega-se uma variável de controle que percorre uma sequência ou intervalo predefinido.

Já o comando `while` é utilizado quando o número de iterações não é conhecido antecipadamente e depende de uma condição. Assim, o bloco de código é repetido enquanto a condição especificada for verdadeira.

# Computador chinês

- Elaborar uma tabela onde cada linha se refere a cada variável envolvida e o resultado de uma operação em particular, ou número da linha do algoritmo (ou observação pertinente);
- Executar os passos previstos no algoritmo;
- Verificar se os resultados obtidos são coerentes. Senão, corrigir o algoritmo e testar novamente para as entradas anteriores;
- Realizar o teste para diferentes entradas e concluir quando todos os testes forem bem sucedidos.

# Exemplo do Computador chinês

```
x = 1
while x < 10:
    print( x )
    x = x + 2
```

Variável	Valor	1	2	3	4	5
x						

# Exemplo do Computador chinês

```
x, y = 1, 5
while x < y:
    y = y * 2
    x = x + y
print(x, y)
```

Variável	Valor	1	2	3	4	5
x	1					
y	5					



# Laços infinitos

É preciso ter muito cuidado com estruturas de repetição.

Sempre que você for escrever uma estrutura de repetição, certifique-se de que a condição será avaliada como falsa em algum momento da execução do laço. Se isso não acontecer, há o risco de o *loop* ser executado indefinidamente (*loop* infinito).

```
x = 1

while x < x + 1:
    x = x + 1
    print( x )

print( 'Acabou!??' )
```

# Exercícios

1. Faça um algoritmo para verificar se o usuário é maior de idade pedindo somente o ano de nascimento. O programa só deve encerrar quando ele digitar um ano que corresponda a maior idade.
2. Escreva um algoritmo de troca de senhas. Ele deve pedir a senha antiga e a senha nova. O algoritmo só encerra quando as senhas forem diferentes.
3. Escreva um algoritmo de troca de senhas mais robusto. Ele deve pedir a senha antiga, pedir a senha nova e a confirmação a senha nova. O algoritmo só deve encerrar sob a condição da senha antiga ser diferente da nova e a confirmação da senha for igual a senha nova.

# Strings

Representadas por dados do tipo texto e podem ser utilizadas para armazenar e manipular informações, como palavras, frases ou dados textuais.

- Uma `str` se comporta como uma sequência;
- Em Python, uma sequência é um tipo de dado composto, que armazena um conjunto de elementos em uma ordem específica;
- No caso de uma `str`, os elementos são os caracteres;
- Cada caractere entre as aspas ocupa uma **posição (índice)** e pode ser acessado usando esse **índice** e colchetes.

# Acesso por índice

Os índices sempre iniciam por 0.

Índice neg.	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Posição	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Caractere	S	a	u	l	o		O	l	i	v	e	i	r	a

```
nome = 'Saulo Oliveira'
print(nome[4])
print(nome[3])
print(nome[13])
```

```
nome = 'Saulo Oliveira'
print(nome[-8])
print(nome[-3])
print(nome[-130])
```