

LINGUAGEM DE PROGRAMAÇÃO

**Linguagem de programação:
Conhecendo Python**

Prof. Me. Wesley Viana

- Unidade de Ensino: 01
- Competência da Unidade: Conhecer a linguagem de programação Python
- Resumo: Saber utilizar modelos de programação na linguagem Python.
- Palavras-chave: Linguagem de programação; Python; Programação; Desenvolvimento; Algoritmos.
- Título da Teleaula: Linguagem de programação: Conhecendo Python
- Teleaula nº: 01

Contextualização

- Linguagem de programação: Conhecendo Python
- A linguagem Python
- Ferramentas
- Exemplos de códigos

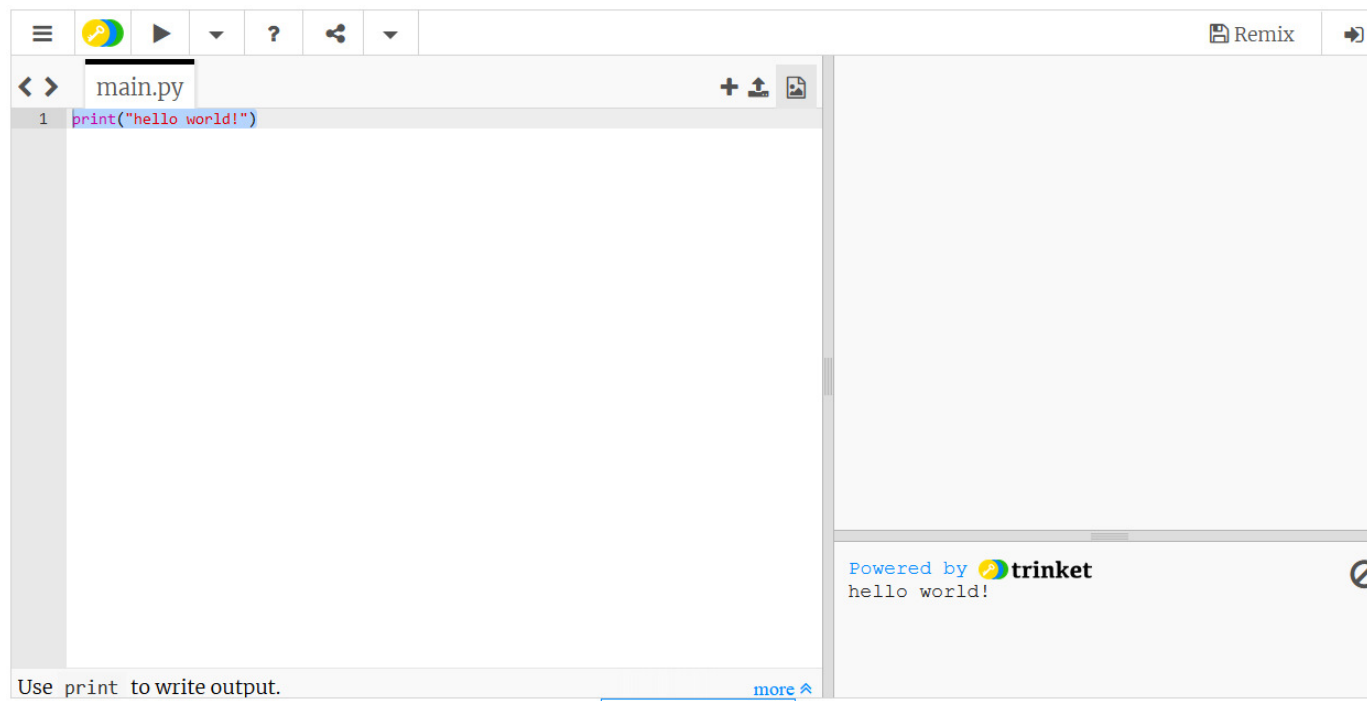
Linguagem de programação: Conhecendo Python

O que é Linguagem de Programação?

- As linguagens de programação foram criadas para solucionar qualquer tipo de problema na área tecnológica computacional.
- Cada linguagem possui suas particularidades.
- Permite que um programador crie programas a partir de um conjunto de ordens, ações consecutivas, dados e algoritmos.
- Python é uma linguagem de *script* de alto nível, de tipagem forte e dinâmica

Primeiros passo em Python:

- Mito "Hello World", acessar <https://trinket.io/python>, digite o comando, `print("hello world!")`



The image shows a screenshot of the Trinket.io Python editor interface. At the top, there is a toolbar with icons for menu, run, undo, redo, help, and share. The main editor area on the left shows a file named 'main.py' with a single line of code: `print("hello world!")`. The right side of the interface is a large empty space for output or a preview. At the bottom right, there is a status bar that says 'Powered by trinket' and 'hello world!'. At the bottom left, there is a footer that says 'Use print to write output.' and a link to 'more'.

```
main.py
1 print("hello world!")
```

Powered by trinket
hello world!

Use print to write output. [more](#)

Primeiros passo em Python:

- Criado no início dos anos 1990 por Guido van Rossum no Stichting Mathematisch Centrum (CWI), na Holanda, foi sucessor de uma linguagem chamada ABC.
- Em 2001, a Python Software Foundation (PSF) foi formada, uma organização sem fins lucrativos criada especificamente para possuir a propriedade intelectual relacionada ao Python.

Primeiros passo em Python:

- Porque Python?

Python é uma linguagem de programação orientada a objetos, clara e poderosa.

Mais referências em:

<https://wiki.python.org/moin/BeginnersGuide/Overview>

Primeiros passo em Python:

- Usa sintaxe clara, facilitando a leitura dos programas que você escreve;
- Linguagem fácil, ideal para o desenvolvimento de protótipos e outras tarefas de programação;
- Grande biblioteca padrão, suporta muitas tarefas de programação;

Primeiros passo em Python:

- Inúmeras bibliotecas que estendem seu poder de atuação.
- Linguagem interpretada, ou seja, uma vez escrito o código, este não precisa ser convertido em linguagem de máquina por um processo de compilação;
- Permite atribuição múltipla. Podemos atribuir valores a mais de uma variável em uma única instrução. Por exemplo, `a, b = 2, 3`.

Primeiros passo em Python:

- Uma das grandes características da linguagem é sua sintaxe. As regras são definidas pelo PEP 8 (Python Enhancement Proposal) e dizem respeito a formatação, indentação, parâmetros em funções e tudo mais que possa estar relacionado à sintaxe do código.
- O interpretador Python 3 utiliza unicode por padrão, o que torna possível usar nomes de variáveis com acento e até outros caracteres especiais, porém não é uma boa prática.
- Códigos em Python pode ser feito tanto em local quanto em nuvem.

Primeiros passo em Python:

- Instalação do interpretador Python

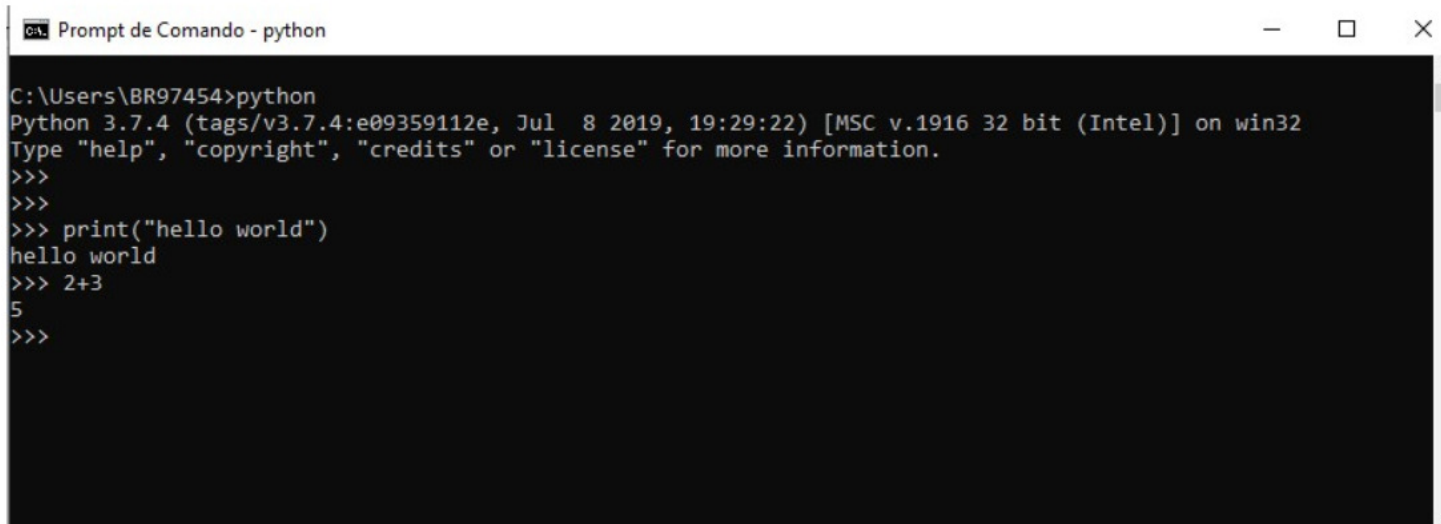
<https://www.python.org/downloads/>

Na instalação marcar a opção Add Python 3.X to PATH.



Primeiros passo em Python:

- Já podemos digitar comandos python



```
Prompt de Comando - python
C:\Users\BR97454>python
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 19:29:22) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> print("hello world")
hello world
>>> 2+3
5
>>>
```

Primeiros passo em Python:

- Mais ferramentas:

Para implementação de soluções, utiliza-se a IDE, (Integrated Development Environment) ou Ambiente de Desenvolvimento Integrado.

Duas IDE's disputam a preferência dos desenvolvedores Python, o PyCharm e o Visual Studio Code (VSCode).

PyCharm: Professional e Community, sendo a primeira paga e a segunda gratuita.

VSCode: Gratuito

Primeiros passo em Python:

- Mais ferramentas (Iniciante):

Python Anaconda (<https://www.anaconda.com/distribution/>). Consiste na união de ferramentas Python, compostas por bibliotecas e IDE's

Possuí tanto o interpretador Python quanto bibliotecas, duas interfaces de desenvolvimento: a IDE spyder e o projeto Jupyter.

Grande diferencial do projeto Anaconda é ter o Jupyter Notebook (<https://jupyter.org/>) integrado na instalação, principalmente para o uso sistemas de controle de versão (como git / GitHub).

Primeiros passo em Python:

- Mais ferramentas :

Google Colaboratory (Colab) <https://colab.research.google.com/notebooks/>

Especialmente adequado para aprendizado de máquina, análise de dados e educação. Colab é um serviço de notebook Jupyter hospedado que não requer configuração para ser usado.

Devido a facilidade e vantagens do Colab, indicamos essa ferramenta como principal meio de trabalho para essa disciplina.

Variáveis e tipos básicos de dados em Python:

- Variáveis são espaços alocados na memória RAM para guardar valores temporariamente.
- Em Python, esses espaços não precisam ser tipados, a variável pode ser alocada sem especificar o tipo de dado que ela aguardará.
- As variáveis são tipadas dinamicamente nessa linguagem.

Variáveis e tipos básicos de dados em Python:

- Veja alguns exemplos:

Para saber o tipo de dado que uma variável guarda, podemos imprimir seu tipo usando a função `type()` , veja como:



The screenshot shows a Jupyter Notebook window titled 'Untitled0.ipynb'. The interface includes a top menu bar with options like 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. Below the menu, there are tabs for '+ Código' and '+ Texto'. The main area contains a code cell with the following Python code:

```
x = 10
nome = 'aluno'
nota = 8.75
faz_inscricao = True

print(type(x))
print(type(nome))
print(type(nota))
print(type(faz_inscricao))
```

Below the code cell, the output is displayed, showing the type of each variable:

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'bool'>
```

Variáveis e tipos básicos de dados em Python:

- Em Python, tudo é objeto! Por isso os tipos de dados aparecem com a palavra "class".
- Função input() faz a leitura de um valor digitado. Veja como usar:

```
nome = input("Digite um nome: ")  
print(nome)
```



Variáveis e tipos básicos de dados em Python:

- Temos uma variedade de formas de imprimir texto e variável em Python. Vejamos algumas: podemos usar formatadores de caracteres (igual em C), podemos usar a função `format()` e podemos criar uma string formatada. Vejamos cada um: <https://colab.research.google.com>

Modo 1: usando formatadores de caracteres (igual na linguagem C):

`print("Olá %s, bem vindo a disciplina de programação. Parabéns pelo seu primeiro hello world" % (nome))`

Variáveis e tipos básicos de dados em Python:

Modo 2: usando a função format() para imprimir variável e texto:

print("Olá {}, bem vindo a disciplina de programação. Parabéns pelo seu primeiro hello world".format(nome))

Modo 3: usando strings formatadas

print(f"Olá {nome}, bem vindo a disciplina de programação. Parabéns pelo seu primeiro hello world")

Variáveis e tipos básicos de dados em Python:

- Usamos o hash `#` para criar comentários de uma linha.
- Nessa PEP, a 498, o autor destaca o uso do "modo 3" como a melhor opção chamando-a de "f-strings".
- As strings formatadas com "f-strings" só podem ser compiladas com o interpretador Python na versão 3.6.

Variáveis e tipos básicos de dados em Python:

- Resumo das operações matemáticas suportadas por Python. Com exceção das funções `abs()` e `pow()` e da notação de potência `**`, as outras operações e sintaxe são similares a diversas linguagens de programação.

Quadro 1.1 | Operações matemáticas em Python

Operação	Resultado
<code>x + y</code>	soma de <code>x</code> e <code>y</code>
<code>x - y</code>	Diferença de <code>x</code> e <code>y</code>
<code>x * y</code>	Produto de <code>x</code> e <code>y</code>
<code>x / y</code>	Quociente de <code>x</code> e <code>y</code>
<code>x // y</code>	Parte inteira do quociente de <code>x</code> e <code>y</code>
<code>x % y</code>	Resto de <code>x / y</code>
<code>abs(x)</code>	Valor absoluto de <code>x</code>
<code>pow(x, y)</code>	<code>x</code> elevado a <code>y</code>
<code>x ** y</code>	<code>x</code> elevado a <code>y</code>

Fonte: adaptada de (<https://docs.python.org/3/library/stdtypes.html>)

Variáveis e tipos básicos de dados em Python:

- Repare como é fundamental conhecer a ordem de procedência das operações para não criar cálculos errados durante a implementação de uma solução. Teste você mesmo os códigos no emulador a seguir e aproveite para explorar outras operações.
1. Primeiro resolvem-se os parênteses, do mais interno para o mais externo.
 2. Exponenciação.
 3. Multiplicação e divisão.
 4. Soma e subtração.

Variáveis e tipos básicos de dados em Python:

Qual o resultado armazenado na variável operacao_1: 25 ou 17?

```
operacao_1 = 2 + 3 * 5
```

Qual o resultado armazenado na variável operacao_2: 25 ou 17?

```
operacao_2 = (2 + 3) * 5
```

Qual o resultado armazenado na variável operacao_3: 4 ou 1?

```
operacao_3 = 4 / 2 ** 2
```

Qual o resultado armazenado na variável operacao_4: 1 ou 5?

```
operacao_4 = 13 % 3 + 4
```

```
print(f"Resultado em operacao_1 = {operacao_1}")
```

```
print(f"Resultado em operacao_2 = {operacao_2}")
```

```
print(f"Resultado em operacao_3 = {operacao_3}")
```

```
print(f"Resultado em operacao_4 = {operacao_4}")
```

Variáveis e tipos básicos de dados em Python:

```
a = 2
```

```
b = 0.5
```

```
c = 1
```

```
x = input("Digite o valor de x: ")
```

```
print(type(a))
```

```
print(type(b))
```

```
print(type(c))
```

```
print(type(x))
```

```
x = float(x) # aqui fazemos a conversão da string para o  
tipo numérico
```

```
y = a * x ** 2 + b * x + c
```

```
print(f"O resultado de y para x = {x} é {y}.")
```

```
print(type(a))
```

```
print(type(b))
```

```
print(type(c))
```

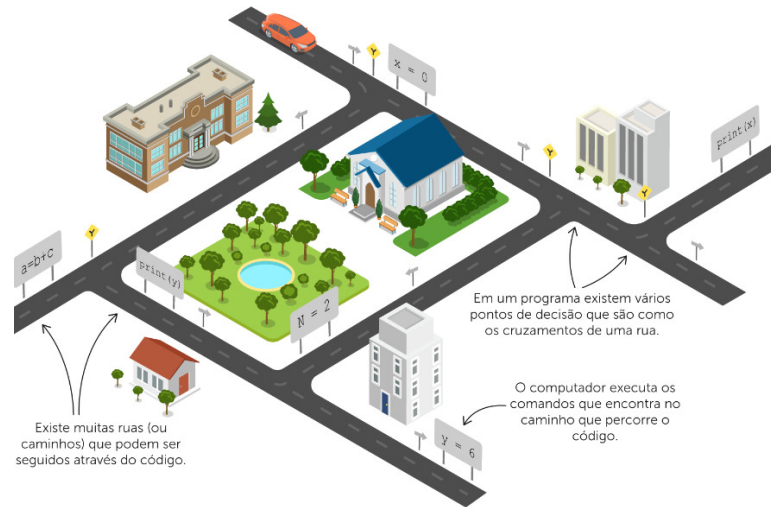
```
print(type(x))
```

Estruturas Lógicas, Condicionais e de Repetição em Python

Estruturas Lógicas, Condicionais e de Repetição em Python

Em geral, em um programa você tem opções de caminhos ou lista de comandos que nada mais são que trechos de códigos que podem ser executados, devendo-se tomar decisões sobre qual trecho de código será executado em um determinado momento.

Figura 1.5 | Fluxo de tomada de decisões de um algoritmo para a escolha de um caminho em uma cidade



Fonte: adaptada de Griffiths e Barry (2009, p. 13).

Estruturas Lógicas, Condicionais e de Repetição em Python

Para tomarmos decisões, precisamos dos operadores relacionais:

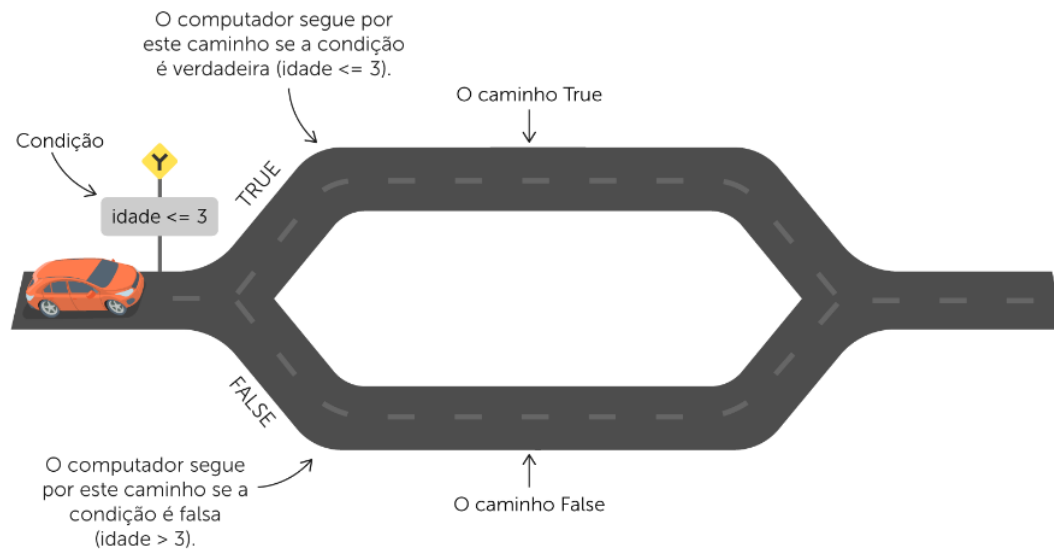
Operação	Significado
<code>a < b</code>	O valor de a é menor que b?
<code>a <= b</code>	O valor de a é menor OU igual que b?
<code>a > b</code>	O valor de a é maior que b?
<code>a >= b</code>	O valor de a é maior OU igual que b?
<code>a == b</code>	O valor de a é igual ao de b?
<code>a != b</code>	O valor de a é diferente do valor de b?
<code>a is b</code>	O valor de a é idêntico ao valor de b?
<code>a is not b</code>	O valor de a não é idêntico ao valor de b?

Fonte: adaptado de <https://docs.python.org/3/library/stdtypes.html>.

Estruturas Lógicas, Condicionais e de Repetição em Python

O comando `if.. else..` significam `se.. senão..` e são usados para construir as estruturas condicionais.

if / else



Fonte: adaptada de Griffiths e Barry (2009, p. 13).

Estruturas Lógicas, Condicionais e de Repetição em Python

Estrutura condicional simples:

```
a = 5
```

```
b = 10
```

```
if a < b:
```

```
    print("a é menor do que b")
```

```
    r = a + b
```

```
    print(r)
```

Estrutura composta:

```
a = 10
```

```
b = 5
```

```
if a < b:
```

```
    print("a é menor do que b")
```

```
    r = a + b
```

```
    print(r)
```

```
else:
```

```
    print("a é maior do que b")
```

```
    r = a - b
```

```
    print(r)
```

Estrutura encadeada, devemos usar o comando "elif", que é uma abreviação de else if.

```
codigo_compra = 5111
```

```
if codigo_compra == 5222:
```

```
    print("Compra à vista.")
```

```
elif codigo_compra == 5333:
```

```
    print("Compra à prazo no boleto.")
```

```
elif codigo_compra == 5444:
```

```
    print("Compra à prazo no cartão.")
```

```
else:
```

```
    print("Código não cadastrado")
```

Estruturas Lógicas, Condicionais e de Repetição em Python

Estruturas lógicas em Python: and, or, not

Podemos usar os operadores booleanos para construir estruturas de decisões mais complexas.

Operador booleano and: o resultado será True, quando os dois argumentos forem verdadeiros.

Operador booleano or: o resultado será True, quando pelo menos um dos argumentos for verdadeiro.

Operador booleano not: ele irá inverter o valor do argumento. Portanto, se o argumento for verdadeiro, a operação o transformará em falso e vice-versa.

Estruturas Lógicas, Condicionais e de Repetição em Python

Estrutura condicional usando os operadores booleanos. Um aluno só pode ser aprovado caso ele tenha menos de 5 faltas e média final igual ou superior a 7.

```
qtde_faltas = int(input("Digite a quantidade de faltas: "))
```

```
media_final = float(input("Digite a média final: "))
```

```
if qtde_faltas <= 5 and media_final >= 7:
```

```
    print("Aluno aprovado!")
```

```
else:
```

```
    print("Aluno reprovado!")
```

Assim como as operações matemáticas possuem ordem de precedência, as operações booleanas também têm. Essa prioridade obedece à seguinte ordem: not primeiro, and em seguida e or por último (BANIN, 2018).

```
A = 15
```

```
B = 9
```

```
C = 9
```

```
print(B == C or A < B and A < C)
```

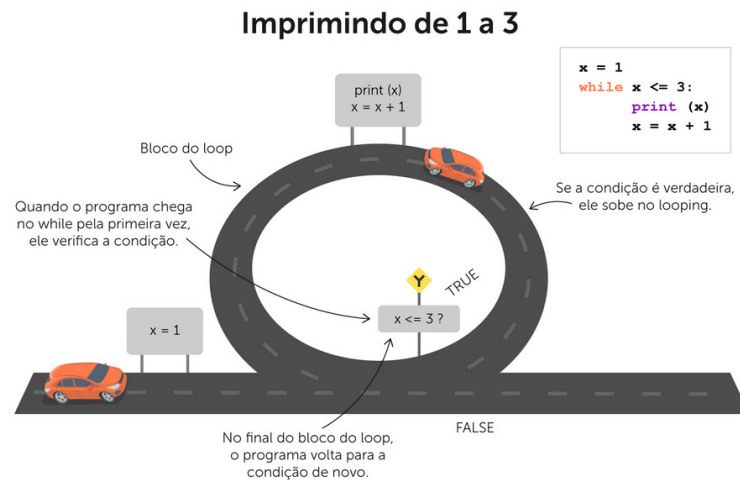
```
print((B == C or A < B) and A < C )
```

Estruturas Lógicas, Condicionais e de Repetição em Python

Estruturas de repetição em Python: while e for

Em uma estrutura de repetição sempre haverá uma estrutura decisão, pois a repetição de um trecho de código sempre está associada a uma condição. Ou seja, um bloco de comandos será executado repetidas vezes, até que uma condição não seja mais satisfeita.

Figura 1.7 | Fluxograma estrutura de repetição



Fonte: adaptada de Griffiths e Barry (2009, p. 28).

Estruturas Lógicas, Condicionais e de Repetição em Python

O comando `while` deve ser utilizado para construir e controlar a estrutura decisão, sempre que o número de repetições não seja conhecido.

```
numero = 1

while numero != 0:

    numero = int(input("Digite um número: "))

    if numero % 2 == 0:

        print("Número par!")

    else:

        print("Número ímpar!")
```

Todo o bloco com a indentação de uma tabulação (4 espaços) faz parte da estrutura de repetição.

Lembre: todos os blocos de comandos em Python são controlados pela indentação.

Estruturas Lógicas, Condicionais e de Repetição em Python

Na prática é comum utilizarmos esse tipo de estrutura de repetição, com `while`, para deixarmos serviços executando em servidores.

A instrução Python `for` itera sobre os itens de qualquer sequência, por exemplo, iterar sobre os caracteres de uma palavra, pois uma palavra é um tipo de sequência.

Estruturas Lógicas, Condicionais e de Repetição em Python

o comando "for" seguido da variável de controle "c", na sequência o comando "in", por fim, a sequência sobre a qual a estrutura deve iterar. Os dois pontos marcam o início do bloco que deve ser repetido.

```
nome = "Guido"
for c in nome:
    print(c)
```

Com o comando for, podemos usar a função enumerate() para retornar à posição de cada item, dentro da sequência. Considerando o exemplo dado, no qual atribuímos a variável "nome" o valor de "Guido", "G" ocupa a posição 0 na sequência, "u" ocupa a posição 1, "i" a posição 2, e assim por diante. Veja que a variável "i" é usada para capturar a posição e a variável "c" cada caractere da palavra.

```
nome = "Guido"
for i, c in enumerate(nome):
    print(f"Posição = {i}, valor = {c}")
```

Estruturas Lógicas, Condicionais e de Repetição em Python

Controle de repetição com range, break e continue:

Python requer uma sequência para que ocorra a iteração. Para criar uma sequência numérica de iteração em Python, podemos usar a função range().

```
for x in range(5):  
    print(x)
```

No comando, "x" é a variável de controle, ou seja, a cada iteração do laço, seu valor é alterado, já a função range() foi utilizada para criar um "iterable" numérico (objeto iterável) para que as repetições acontecesse.

Estruturas Lógicas, Condicionais e de Repetição em Python

A função `range()` pode ser usada de três formas distintas:

Método 1: passando um único argumento que representa a quantidade de vezes que o laço deve repetir;

Método 2: passando dois argumentos, um que representa o início das repetições e outro o limite superior (NÃO INCLUÍDO) do valor da variável de controle;

Método 3: Passando três argumentos, um que representa o início das repetições; outro, o limite superior (NÃO INCLUÍDO) do valor da variável de controle e um que representa o incremento. Observe as três maneiras a seguir.

Método 1

```
for i in range(10):  
    print(i)
```

Método 2

```
for i in range(0, 5):  
    print(i)
```

#Método 3

```
for i in range(0, 20, 2):  
    print(i)
```

Estruturas Lógicas, Condicionais e de Repetição em Python

Além de controlar as iterações com o tamanho da sequência, outra forma de influenciar no fluxo é por meio dos comandos "break" e "continue". O comando break para a execução de uma estrutura de repetição, já com o comando continue, conseguimos "pular" algumas execuções, dependendo de uma condição.

Estruturas Lógicas, Condicionais e de Repetição em Python

Exemplo de uso do break

disciplina = "Linguagem de programação"

for c in disciplina:

if c == 'a':

break

else:

print(c)

Exemplo de uso do continue

disciplina = "Linguagem de
programação"

for c in disciplina:

if c == 'a':

continue

else:

print(c)

**Pense onde podemos
utilizar as vantagens da
programação em
Python**

Exercício: Procura letras e posição

Procura letras e posição

Vamos criar uma solução que procura pelas vogais "A", "o" em um texto. Toda vez que essas letras são encontradas, devemos informar que encontramos e qual posição do texto ela está. Nosso texto será:

texto = A inserção de comentários no código do programa é uma prática normal. Em função disso, toda linguagem de programação tem alguma maneira de permitir que comentários sejam inseridos nos programas. O objetivo é adicionar descrições em partes do código, seja para documentá-lo ou para adicionar uma descrição do algoritmo implementado (BANIN, 2018, p. 45).

Procura letras e posição

```
texto = ""
```

A inserção de comentários no código do programa é uma prática normal.

Em função disso, toda linguagem de programação tem alguma maneira de permitir que comentários sejam inseridos nos programas.

O objetivo é adicionar descrições em partes do código, seja para documentá-

lo ou para adicionar uma descrição do algoritmo implementado (BANIN, p. 45, 2018)."

```
"""
```

```
for i, c in enumerate(texto):
```

```
    if c == 'A' or c == 'o':
```

```
        print(f"Vogal '{c}' encontrada, na posição {i}")
```

```
    else:
```

```
        continue
```

Implementando Soluções em Python Mediante Funções

Implementando Soluções em Python Mediante Funções

Solução dividindo-a em funções (blocos), além de ser uma boa prática de programação, tal abordagem facilita a leitura, a manutenção e a escalabilidade da solução.

- `print()` é uma função built-in do interpretador Python.

<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

Fonte: <https://docs.python.org/3/library/functions.html>

Implementando Soluções em Python Mediante Funções

Solução dividindo-a em funções (blocos), além de ser uma boa prática de programação, tal abordagem facilita o desenvolvimento. Função **built-in** é um objeto que está integrado ao núcleo do interpretador, não precisa ser feita nenhuma instalação adicional.

- *print()* é uma função built-in do interpretador Python.
- algumas funções que já usamos:
 - enumerate()*, para retornar a posição de um valor em uma sequência.
 - input()*, para capturar um valor digitado no teclado.
 - int()* e *float()*, para converter um valor no tipo inteiro ou float.
 - range()*, para criar uma sequência numérica.

Implementando Soluções em Python Mediante Funções

A função `eval()` usada no código recebe como entrada uma *string* digitada pelo usuário, que nesse caso é uma equação linear. Essa entrada é analisada e avaliada como uma expressão Python pela função `eval()`. Veja que, para cada valor de x , a fórmula é executada como uma expressão matemática (linha 8) e retorna um valor diferente. Prudência para o uso, pois é fácil alguém externo à aplicação fazer uma "injection" de código intruso.

```
a = 2
```

```
b = 1
```

```
equacao = input("Digite a fórmula geral  
da equação linear (a * x + b): ")
```

```
print(f"\nA entrada do usuário  
{equacao} é do tipo {type(equacao)}")
```

```
for x in range(5):
```

```
    y = eval(equacao)
```

```
    print(f"\nResultado da equação para x  
= {x} é {y}")
```

Implementando Soluções em Python Mediante Funções

Função definida pelo usuário

Podemos escolher o nome da função, sua entrada e sua saída. Nomes das funções devem estar em minúsculas, com as palavras separadas por underline, conforme necessário, para melhorar a legibilidade.

Os nomes de variáveis seguem a mesma convenção que os nomes de funções.

É preciso abrir e fechar parênteses, pois é dentro dos parênteses que os parâmetros de entrada da função devem ser definidos.

Usamos o comando "def" para indicar que vamos definir uma função. Em seguida, escolhemos o nome da função "imprimir_mensagem"

Implementando Soluções em Python Mediante Funções

A função recebe dois parâmetros. Esses parâmetros são variáveis locais, ou seja, são variáveis que existem somente dentro da função. Na linha 2, imprimimos uma mensagem usando as variáveis passadas como parâmetro e na linha 5, invocamos a função, passando como parâmetros dois valores do tipo string. O valor "Python" vai para o primeiro parâmetro da função e o valor "ADS" vai para o segundo parâmetro.

```
def imprimir_mensagem(disciplina, curso):  
    print(f"Minha primeira função em Python  
desenvolvida na disciplina: {disciplina}, do curso:  
{curso}.")  
  
imprimir_mensagem("Python", "ADS")
```

Implementando Soluções em Python Mediante Funções

Funções com parâmetros definidos e indefinidos

Sobre os argumentos que uma função pode receber, para nosso estudo, vamos classificar em seis grupos:

01. Parâmetro posicional, obrigatório, sem valor default (padrão), tentar um invocar a função, sem passar os parâmetros, acarreta um erro.

```
def somar(a, b):  
    return a + b
```

```
r = somar(2, 3)  
print(r)
```

Implementando Soluções em Python Mediante Funções

02. Parâmetro posicional, obrigatório, com valor default (padrão), quando a função for invocada, caso nenhum valor seja passado, o valor default é utilizado.

```
def calcular_desconto(valor, desconto=0): # O parâmetro desconto possui zero valor default
```

```
    valor_com_desconto = valor - (valor * desconto)
```

```
    return valor_com_desconto
```

```
valor1 = calcular_desconto(100) # Não aplicar nenhum desconto
```

```
valor2 = calcular_desconto(100, 0.25) # Aplicar desconto de 25%
```

```
print(f"\nPrimeiro valor a ser pago = {valor1}")
```

```
print(f"\nSegundo valor a ser pago = {valor2}")
```

Implementando Soluções em Python Mediante Funções

03. Parâmetro nominal, obrigatório, sem valor default (padrão). Não mais importa a posição dos parâmetros, pois eles serão identificados pelo nome, a chamada da função é obrigatório passar todos os valores e sem valor default

```
def converter_maiuscula(texto, flag_maiuscula):
```

```
    if flag_maiuscula:
```

```
        return texto.upper()
```

```
    else:
```

```
        return texto.lower()
```

```
texto = converter_maiuscula(flag_maiuscula=True, texto="João") # Passagem nominal de parâmetros
```

```
print(texto)
```

Implementando Soluções em Python Mediante Funções

04.Parâmetro nominal, obrigatório, com valor default (padrão), nesse grupo os parâmetros podem possuir valor default.

```
def converter_minuscula(texto, flag_minuscula=True): # O parâmetro flag_minuscula possui True como valor default
```

```
    if flag_minuscula:
```

```
        return texto.lower()
```

```
    else:
```

```
        return texto.upper()
```

```
texto1 = converter_minuscula(flag_minuscula=True, texto="LINGUAGEM de Programação")
```

```
texto2 = converter_minuscula(texto="LINGUAGEM de Programação")
```

```
print(f"\nTexto 1 = {texto1}")
```

```
print(f"\nTexto 2 = {texto2}")
```

Implementando Soluções em Python Mediante Funções

A função poderá receber um número diferente de parâmetros a cada invocação. Esse cenário é o que caracteriza os grupos 5 e 6.

05. Parâmetro posicional e não obrigatório (args), a passagem de valores é feita de modo posicional, porém a quantidade não é conhecida.

```
def imprimir_parametros(*args):  
    qtde_parametros = len(args)  
    print(f"Quantidade de parâmetros = {qtde_parametros}")  
  
    for i, valor in enumerate(args):  
        print(f"Posição = {i}, valor = {valor}")  
  
print("\nChamada 1")  
imprimir_parametros("São Paulo", 10, 23.78, "João")  
  
print("\nChamada 2")  
imprimir_parametros(10, "São Paulo")
```


Implementando Soluções em Python Mediante Funções

06. Parâmetro nominal e não obrigatório (kwargs), agora a passagem é feita de modo nominal e não posicional, o que nos permite acessar tanto o valor do parâmetro quanto o nome da variável que o armazena.

```
def imprimir_parametros(**kwargs):  
    print(f"Tipo de objeto recebido = {type(kwargs)}\n")  
    qtde_parametros = len(kwargs)  
    print(f"Quantidade de parâmetros = {qtde_parametros}")  
  
    for chave, valor in kwargs.items():  
        print(f"variável = {chave}, valor = {valor}")  
  
print("\nChamada 1")  
imprimir_parametros(cidade="São Paulo", idade=33,  
                    nome="João")  
print("\nChamada 2")  
imprimir_parametros(desconto=10, valor=100)
```

Implementando Soluções em Python Mediante Funções

Funções anônimas em Python

Uma função anônima é uma função que não é construída com o "def" e, por isso, não possui nome. Esse tipo de construção é útil, quando a função faz somente uma ação e é usada uma única vez. Poderoso recurso da linguagem Python: a expressão "lambda".

```
somar = lambda x, y: x + y
```

```
somar(x=5, y=3)
```

**Qual a importância de
saber utilizar funções em
linguagem de
programação Python?**

Recapitulando

Recapitulando

- Primeiros passos em Python
- Variáveis e tipos básicos de dados em Python
- Estruturas Lógicas, Condicionais e de Repetição em Python
- Implementando Soluções em Python Mediante Funções
- Importância de saber utilizar modelos de estrutura de dados