

**LÓGICA**

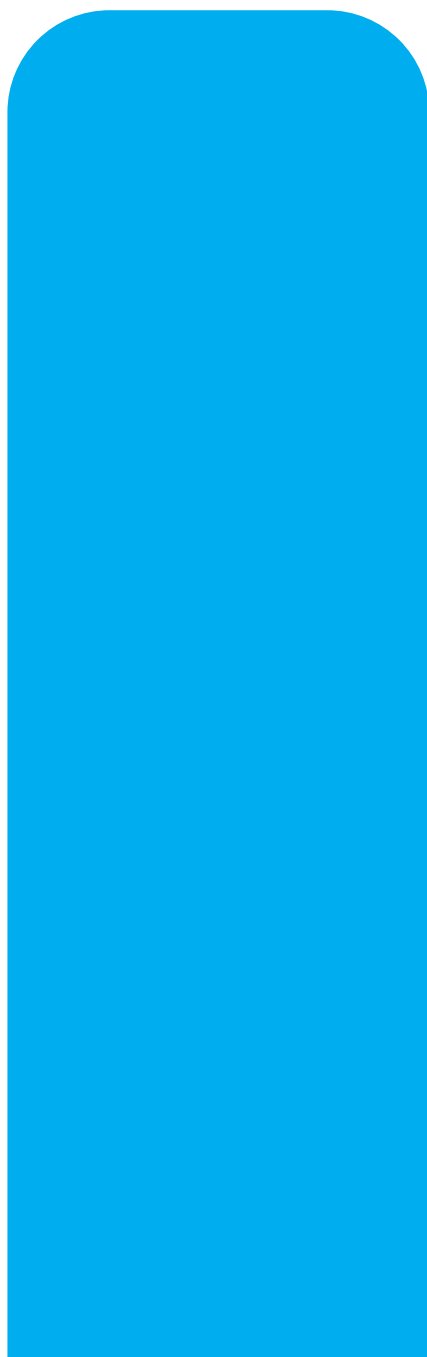
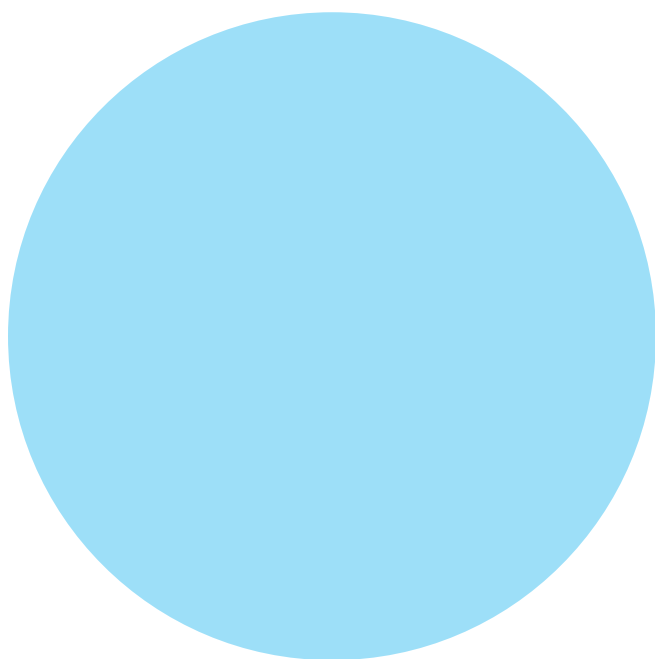
**EM**



**PYTHON**

**BY:** Leonardo José da Silva jr

# Tipo de Dados



# Lista de dados



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



## Tipo 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.

Exemplos:

Entrada:

```
idade = 18  
ano = 2002
```

```
print(type(idade))  
print(type(ano))
```

Saida:

```
<class 'int'>  
<class 'int'>
```



## 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”.

### Exemplos:

```
altura=1.80
```

```
peso = 73.55
```

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

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

Saida:

```
<class 'float'>
```

```
<class 'float'>
```



## 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 contém duas partes: a parte real e a parte imaginária, sendo que a parte imaginária contém 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.

### Exemplos:

```
a = 5+2jb = 20+6j
```

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

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

```
print(complex(2, 5))
```

### Saida:

```
<class 'complex'>
```

```
<class 'complex'>
```

```
(2+5j)
```



## String (str)

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

### Exemplos:

```
nome = 'Guilherme'  
profissao = 'Engenheiro de Software'  
print(type(profissao))  
print(type(nome))
```

### Saida:

```
<class 'str'>  
<class 'str'>
```



## 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.

Exemplos:

```
fim_de_semana = True
```

```
feriado = False
```

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

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

Saida:

```
<class 'bool'>
```

```
<class 'bool'>
```





## Listas (list)

Tipo de dado muito importante e que é muito utilizado no dia a dia do desenvolvedor Python!

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, veja alguns exemplo abaixo:

```
alunos = ["Amanda", "Ana", "Bruno", "João"]
```

```
notas = [10, 8.5, 7.8, 8.0]
```

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

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

Saida:

```
<class 'list'>
```

```
<class 'list'>
```



## Tuplas (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.

Vamos ver alguns exemplos:

```
valores = (90, 79, 54, 32, 21)
pontos = (100, 94.05, 86.8, 62)
```

```
print(type(valores))
print(type(pontos))
```

Saida:

```
<class 'tuple'>
<class 'tuple'>
```



## 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.

Vamos ver alguns exemplos:

```
altura = {"Amanda": 1.65, "Ana": 1.60, "João": 1.70}
```

```
peso = {"Amanda": 60, "Ana": 58, "João": 68}
```

Saida:

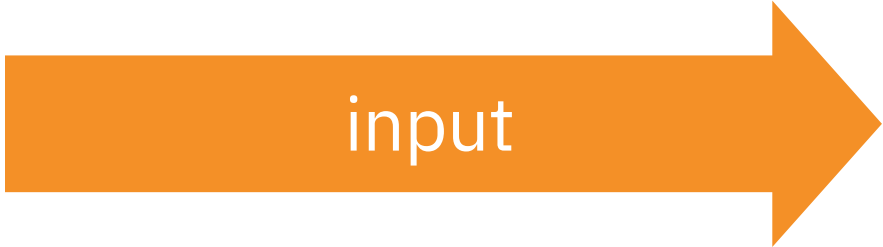
```
print(type(altura))
```

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

<class 'dict'>

<class 'dict'>

Entrada



e



Saída de dados



# Saída de dados

O Comando utilizado para a **saída** de dados na tela do console na Linguagem Python é a função **print()**;

A função **print()** exibe um texto na tela. O texto deve estar entre aspas simples (**' '**) ou aspas duplas (**" "**).

**Exemplo:**

```
print("Olá, Mundo!")
```

**Caracteres especiais:**

**\n** avanço de linha

**\t** tabulação (tab)

**\b** retrocesso (backspace)

**\"** aspas duplas

**\\** barra

Para formatar um texto com várias saídas, utiliza-se uma string formatada (f-String) em conjunto com a função print().

Exemplo:

```
disciplina = "Fundamentos de Programação"
```

```
topico = "f-string no Python"
```

```
dificuldade = 'Básico'
```

```
print(
```

```
f"Site: {disciplina}\n"
```

```
f"Título: {topico}\n"
```

```
f"Dificuldade: {dificuldade}"
```

```
)
```

# Saida de dados



Existe uma construção que permite formatar a saída com certa facilidade: é o método **format**.

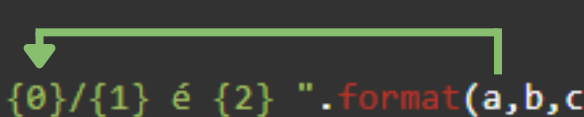
A forma mais simples de usar essa construção é **marcar a posição**, no comando **print**, onde o conteúdo da **variável** (ou resultado da expressão) aparecerá.

Para isso, utiliza-se um número entre chaves, indicando a **posição** que a variável deverá estar no método **format**.

Exemplo:

Entrada:

```
a = 1
b = 7
c = b/a
print("valor de {0}/{1} é {2} ".format(a,b,c))
```



Saida:

```
valor de 1/7 é 7.0
```



Além da formatação posicional das variáveis, é possível, também, incluir caracteres de preenchimento, alinhamento, a especificação da larguras mínima e máxima que se deseja reservar para um número/string, assim como o número de casas decimais que serão exibidos. Para efetuar a formatação, são utilizados códigos dentro da marcação dos parêntesis seguidos de : (dois pontos).

forma geral:

: [preenchimento] [tipo de alinhamento] [largura] . [precisão]







Além da formatação posicional das variáveis, é possível, também, incluir caracteres de preenchimento, alinhamento, a especificação da larguras mínima e máxima que se deseja reservar para um número/string, assim como o número de casas decimais que serão exibidos. Para efetuar a formatação, são utilizados códigos dentro da marcação dos parêntesis seguidos de : (dois pontos).

Para os números inteiros usa-se o código:

**:nd**

**n** número mínimo de espaços reservados para números ou caracteres de preenchimento.

**d** tipo inteiro

**:nf**

**n** número mínimo de espaços reservados para números ou caracteres de preenchimento.

**f** tipo real

# Entrada de dados



Para os números reais (float), pode-se fixar o número de casas decimais que será exibido usando o código:

## Sem formatação:

Entrada :

```
a = 1.0  
b = 7.0  
c = a/b  
  
print("valor de {0}/{1} é {2} ".format(a,b,c))
```

Saida:

```
valor de 1.0/7.0 é 0.14285714285714285
```

## Formatado: Foramatado para mostrar apenas 2 casas decimais

Entrada :

```
a = 1.0  
b = 7.0  
c = a/b  
  
print("valor de {0}/{1} é {2:.2f} ".format(a,b,c))
```

Saida:

```
valor de 1.0/7.0 é 0.14
```

# Entrada de dados



Para a entrada de dados através do teclado do usuário, temos a função `input()` em Python;

A função `input()` lê um texto qualquer informado pelo usuário. Este texto pode ser armazenado numa variável.

Exemplo:

```
print("Informe seu nome: ")
nome = input()
print("Bem-vindo, ", nome)
```

Um detalhe importante a ser lembrado é que a função `input()` sempre lê **strings**. Por isso, caso deseje ler um número, deverá converter o dado retornado por essa função para o formato numérico apropriado.

Exemplo:

```
print("Informe sua idade: ")
Idade = int(input())
print(f"{nome}, você tem {idade} anos.")
```

# Operadores

## LÓGICOS

## Relacionais

and

## Aritméticos





# o que são operadores lógicos ?

Operador lógico é um elemento que liga as condições que compõem um comando de pesquisa. É o operador lógico que indica para o sistema a maneira como se quer que uma palavra esteja em relação à outra dentro da condicional, para que essa condicional seja atendida ou não.

## Operadores lógicos em python:

Operador	Descrição	Tipo
and	Retorna True se ambas as condições forem atendidas	Conjunção
or	Retorna True mesmo se apenas uma condição for atendida	Disjunção
not	Nega o resultado	Negação



# o que são operadores relacionais ?

Os operadores relacionais trabalham como comparações, igualdades e desigualdades. Eles verificam os valores dos operandos, que ficam cada um de um lado da operação, retornando VERDADEIRO ou FALSO.

## Operadores relacionais em python:

Considere que a variável **A** contém o valor **10** e a variável **B** contém **20**

Operadores	Descrição	Comparação	Resultado
==	Igual	(A == B)	False
!=	Diferente	(A != B)	True
>	maior que	(A > B)	False
<	Menor que	(A < B)	True
>=	Maior ou igual a	(A >= B)	False
<=	Menor ou igual a	(A <= B)	True



# o que são operadores Aritméticos?

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

Operador	Descrição	Exemplo	Resultado
<b>+</b>	Adição	2 + 2	4
<b>-</b>	Subtração	2 - 2	0
<b>*</b>	Multiplicação	2 * 2	4
<b>/</b>	Divisão	2 / 2	1
<b>//</b>	Divisão inteira	8 // 2	4
<b>%</b>	Resto da divisão	8 % 2	0
<b>**</b>	Potências	4 ** 3	64
<b>**(1/n)</b>	Raiz	9**(1/2)	3



# Condicionais

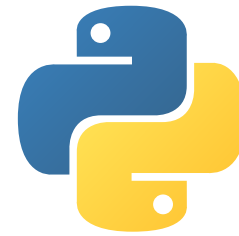
if  
elif  
else

match  
case





# IF, Elif e Else



## IF (se) Exemplo = `if 5 >= 3:`

O IF deve propor alguma coisa. É preciso escrever o IF e logo depois colocar a condição analisada. Então, em seguida, o bloco de comandos.

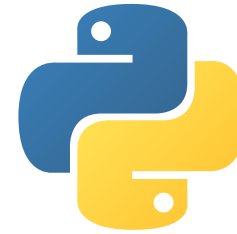
## Elif (se não se) Exemplo = `elif 5 == 3:`

O elif é uma estrutura intermediária dentro da seção if-else no python e deve vir como um complemento a ambos. Quando você já tem um IF e um ELSE, mas precisa de uma condição para especificar outra regra, pode usar o elif.

## Else (se não) Exemplo = `else:`

O ELSE surge depois do IF, em complemento lógico a ele. Então, não existe hipótese de escrever um ELSE sem um IF antes. Geralmente, o ELSE não requer um teste, uma comparação, pois ele executa algo caso a comparação do IF não passe.

# Match case



**match case** é uma estrutura de condição que define o código a ser executado com base em uma comparação de valores.

## Explicação:

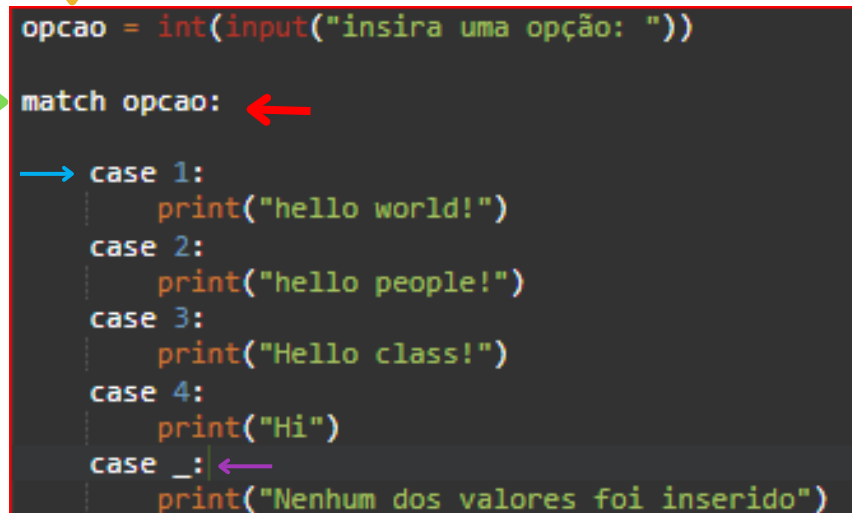
Instanciando a variável; >

Iniciando Match case; >

Atribuindo a variável que contém o valor que será comparado; >

Atribuindo casos e os valores desse caso; >

Atribuimos o valor " " no ultimo caso para sinalizar que nenhuma das opções acima foi atendida >



```
opcao = int(input("insira uma opção: "))  
  
match opcao:  
    case 1:  
        print("hello world!")  
    case 2:  
        print("hello people!")  
    case 3:  
        print("Hello class!")  
    case 4:  
        print("Hi")  
    case _:  
        print("Nenhum dos valores foi inserido")
```

Inseirindo o valor 3:

```
insira uma opção: 3  
Hello class!
```

Inseirindo um valor que não atende nenhum dos casos:

```
insira uma opção: 6  
Nenhum dos valores foi inserido
```

# 1.0



## Repetição

# while



# Estrutura de repetição - While (Enquanto)



**While** é conhecido como estrutura de repetição de laço **infinito**, por ser geralmente utilizada quando **não sabemos a quantidade exata** de vezes que precisaremos executar uma mesma função.

Para tornar o **while** uma estrutura de repetição infinita devemos utilizar o valor **True** para que o computador entenda a seguinte mensagem:

"Enquanto (**While**) verdadeiro(**True**) repita(:)"

**Entrada:**

```
contador = 1
while True:
    print(f"{contador}º")
    if contador == 4:
        print("Repetição finalizada")
        break
    contador += 1
```

**Saída:**

segunda



## Estrutura de repetição - While (Enquanto)

**While** é conhecido como estrutura de repetição de laço **infinito**, por ser geralmente utilizada quando **não sabemos a quantidade exata** de vezes que precisaremos executar uma mesma função.

Para tornar o **while** uma estrutura de repetição infinita devemos utilizar o valor **True** para que o computador entenda a seguinte mensagem:

"Enquanto (**While**) verdadeiro(**True**) repita(:)"

### Entrada:

```
contador = 1
while True:
    print(f"{contador}º")
    if contador == 4:
        print("Repetição finalizada")
        break
    contador += 1
```

### Saída:

```
segunda
```



# Estrutura de repetição - **While** - contador

Contador é variável utilizada para contabilizar e/ou limitar a quantidade de repetições.

O contador deve ser instanciado com algum valor atribuído e antes da estrutura de repetição.

**Incremento:**

`contador += n;`

**Decremento :**

`contador -= n;`

Tanto o incremento podem ser utilizados com qualquer operador aritmético:

**`*=` , `\=` , `**=` e etc..**

**Entrada:**

```
1 contador = 0
2
3 while contador < 5:
4     print(contador)
5     contador += 1
6
```

**Saida:**

```
0
1
2
3
4
```



# While-else

Ao final do **while** podemos utilizar a instrução **else**. O propósito disso é executar alguma instrução ou bloco de código ao final do **loop**, como podemos ver no exemplo a seguir:

## Código:

```
x = 0
while x < 4:
    print(x)
    x += 1
else:
    print("fim while")
```

## Mostra:

```
0
1
2
3
fim while
```

Se dentro da repetição for executado o comando **break**, o loop será encerrado sem executar o conjunto da cláusula **else**.

## Código:

```
x = 0
while x < 4:
    x += 1
    if x == 2:
        continue
    print(f"x é igual a {x}")
else:
    print("fim while")
```

## Mostra:

```
x é igual a 1
x é igual a 3
x é igual a 4
fim while
```

Se dentro da repetição for executado o comando **continue**, a próxima ação será ignorada e logo após atender os requisitos do **while** o conjunto da cláusula **else** será executado.