



INSTITUTO  
FEDERAL  
Alagoas

# ATLETAS DA PROGRAMAÇÃO

AULA 4



REVISÃO GERAL

OK

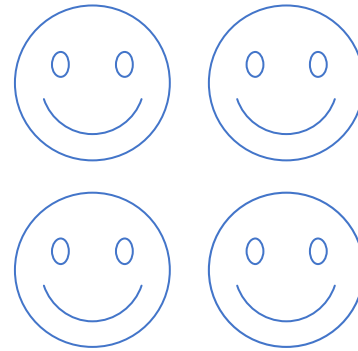
# ASSUNTOS QUE VAMOS REVISAR HOJE



- Comentários;
- Variáveis e tipos de dados primitivos;
- Operadores;
- Strings e Indexação;
- Inputs;
- Estruturas de Decisão;
- Laços;
- Estruturas de Dados;



# COMENTÁRIOS



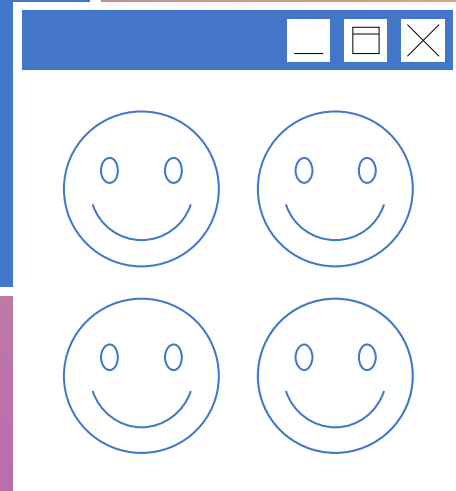
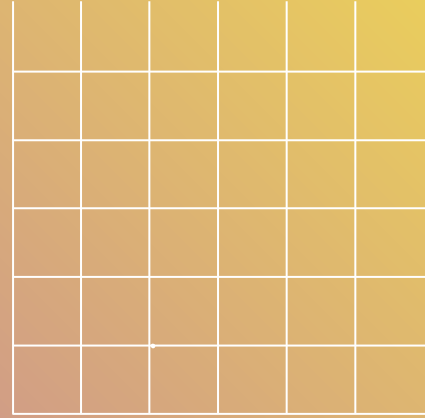
# O que é comentar algo?

- São importantes para facilitar a compreensão do código, tanto para si próprio, como para quem lerá ele.

Forma 1: `#comentario`

Forma 2: `'''comentario'''`

# VARIÁVEIS E TIPOS DE DADOS PRIMITIVOS



# O que são variáveis?

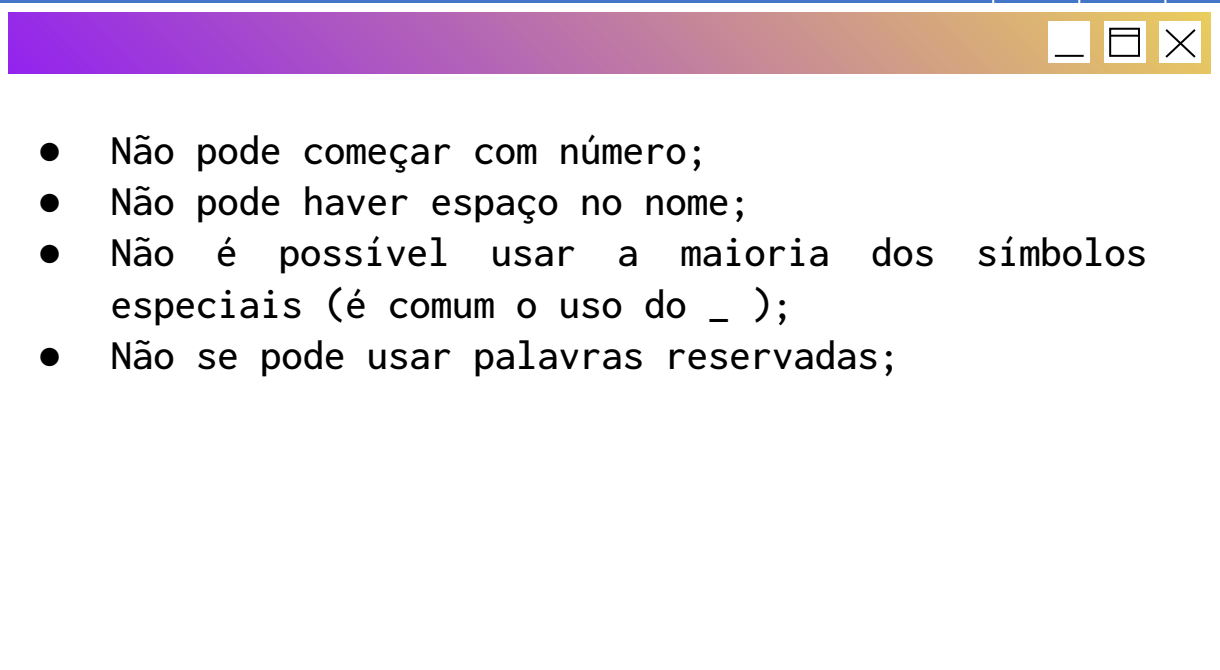
- Usadas para armazenar valores que queremos usar posteriormente;

Nome = "gabriel"

Idade = 33

Altura = 1.75

# Regras de nomeação de variáveis

- 
- Não pode começar com número;
  - Não pode haver espaço no nome;
  - Não é possível usar a maioria dos símbolos especiais (é comum o uso do \_ );
  - Não se pode usar palavras reservadas;

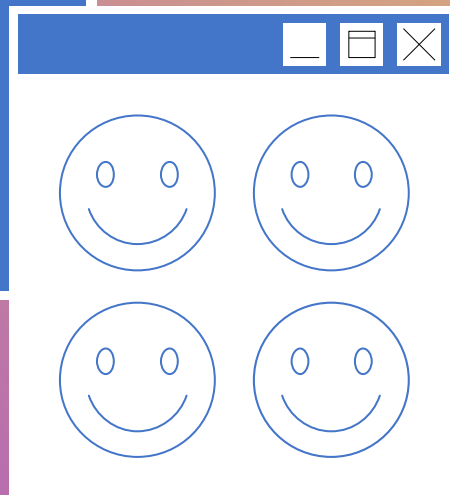
# O que são tipos primitivos?

- São maneiras de classificar determinadas informações:

int  
float  
str  
bool




# OPERADORES



# OPERADORES ARITMÉTICOS

| Operação                     | Símbolo  | Descrição  |
|------------------------------|----------|--|
| Adição                       | +        | ✓ Operadores aritméticos tradicionais de <b>adição</b> , <b>subtração</b> , <b>multiplicação</b> e <b>divisão</b> .<br>✓ Ex.: $2 + 4 = 6$ ; $2 * 10 = 20$ ; $10 / 2 = 5$ . |
| Subtração                    | -        |  |
| Multiplicação                | *        |  |
| Divisão                      | /        |  |
| Resto da divisão inteira     | mod ou % | ✓ Operador de módulo (isto é, resto da divisão inteira).<br>✓ Ex.: $8 \bmod 3 = 2$ .   |
| Quociente da divisão inteira | \        | ✓ Operador de divisão inteira.<br>✓ Por exemplo, $7 \backslash 2 = 3$ .  |
| Potenciação                  | ^        | ✓ Operador de potenciação.<br>✓ Ex.: $5^2 = 25$ .  |

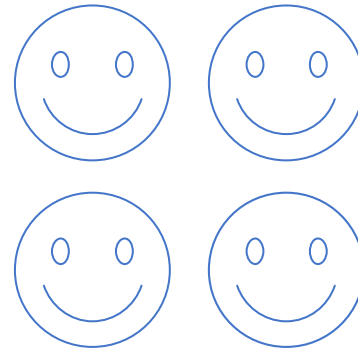
# OPERADORES RELACIONAIS



| Operador | Comparação       |
|----------|------------------|
| =        | Igual a          |
| <>       | Diferente de     |
| <        | Menor que        |
| <=       | Menor ou igual a |
| >        | Maior que        |
| >=       | Maior ou igual a |



# INPUTS

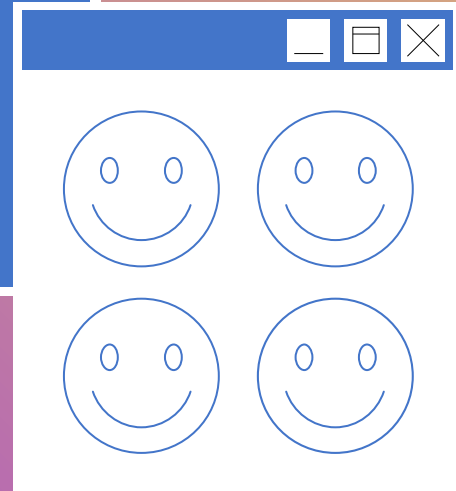


# 0 que são inputs?

- Usados para recebermos algum valor/informação dos usuários.

```
Nome = input()
```

# ESTRUTURAS DE DECISÃO

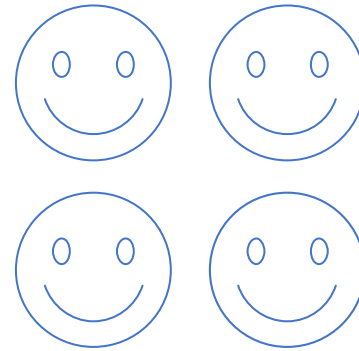


# ELSE, IF E ELIF

- `else`: a palavra-chave **else** é chamada, inevitavelmente, caso as condições anteriores tenham sido falsas.
- `if`: uma estrutura de decisão é construída usando a palavra-chave **if**.
- `elif`: A palavra-chave **elif** é uma maneira de dizer: "se as condições anteriores não forem verdadeiras, tente esta condição."



LAÇOS





# WHILE E FOR

- **while:** com a função **while**, podemos repetir a execução de um conjunto de instruções enquanto uma condição for verdadeira.
- **for:** a função **for** é usada similarmente ao **while**. A diferença, aqui, é que o **for** precisa de um número definido de loops, enquanto o **while** pode ser usado quando você desconhece a quantidade total de repetições.

# AUXILIADORES

Existem 3 comandos que nos auxiliam quando queremos alterar o fluxo de uma estrutura de repetição.

São eles: **break**, **continue** e **pass**.

# AUXILIADORES: BREAK

É usado para finalizar um loop, isto é, é usado para parar sua execução. Geralmente vem acompanhado de alguma condição para isso, com um if.

**Veja um exemplo em for no próximo slide:**

# AUXILIADORES: BREAK

```
for num in range(10):  
    # Se o número for igual a 5, devemos parar o loop  
    if num == 5:  
        # Break faz o loop finalizar  
        break  
    else: print(num)
```

# AUXILIADORES: BREAK

Output:

0

1

2

3

4

# AUXILIADORES: BREAK

Já com `while`, também podemos utilizar o `break` em uma condição utilizando `if`, assim:

```
num = 0

while num < 5:
    num = num + 1
    if num == 3:
        break
    print(num)
```

# AUXILIADORES: BREAK

Quando a variável atribuir o valor 4 o laço é finalizado pelo break, encerrando o loop.

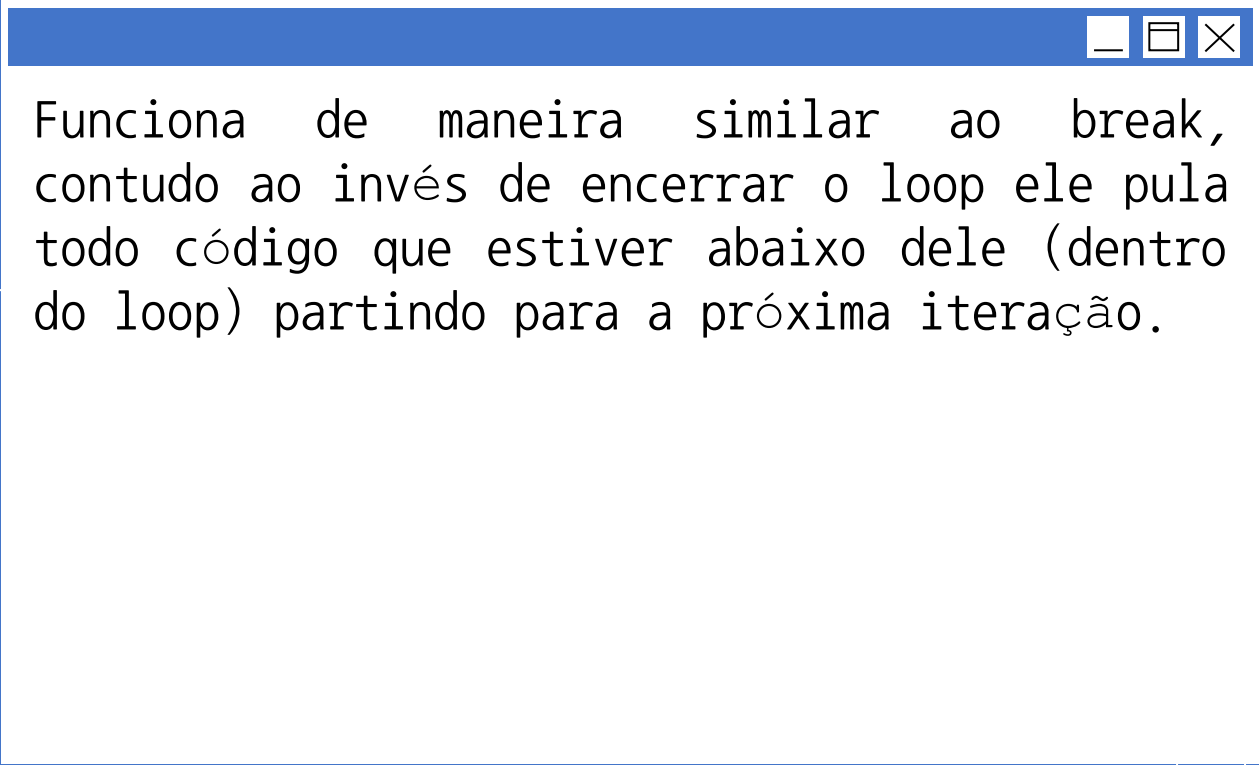
Output:

1

2

3

# AUXILIADORES: CONTINUE



Funciona de maneira similar ao break, contudo ao invés de encerrar o loop ele pula todo código que estiver abaixo dele (dentro do loop) partindo para a próxima iteração.



# AUXILIADORES: CONTINUE

## EXEMPLO COM FOR

```
for num in range(5):  
    if num == 3:  
        print("Encontrei o 3")  
        # Executa o continue, pulando para  
        # o próximo laço continue  
    else: print(num)  
print("Estou abaixo do IF")
```

## OUTPUT

```
0  
Estou abaixo do IF  
1  
Estou abaixo do IF  
2  
Estou abaixo do IF  
Encontrei o 3  
4
```

ATENÇÃO: Repare no output anterior que quando a condição `num == 3` for satisfeita, a string "Estou abaixo do IF" não será exibida.



# AUXILIADORES: CONTINUE

## EXEMPLO COM WHILE

```
num = 0
while num < 5:
    num += 1
    if num == 3:
        continue
    print(num)
```

## OUTPUT

No resultado desse código o 3 não deve aparecer, pois o `print()` que imprime os números está abaixo do `continue`.

1  
2  
4  
5

# AUXILIADORES: PASS

O **pass** nada mais é que uma forma de fazer um código que não realiza operação nenhuma.

Como os escopos de Classes, Funções, If/Else e loops for/while são definidos pela indentação do código (e não por chaves {} como geralmente se vê em outras linguagens de programação), usamos o **pass** para dizer ao Python que o bloco de código está vazio.

Veja alguns exemplos no próximo slide:



Caso não utilizemos o pass, veja o que acontece:

```
class Classe:  
  
    def funcao():  
  
        pass
```

Output:

```
File "", line 2
```

```
^
```

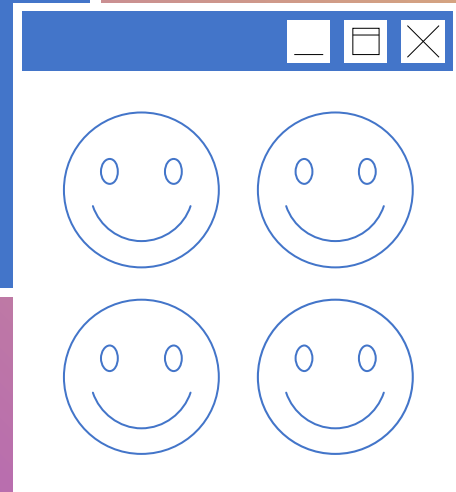
```
IndentationError: expected an indented block
```

Isso acontece pois o Python entende que as próximas linhas de código fazem parte do mesmo escopo, mas como não estão indentadas um erro `IndentationError` é lançado.



# Estruturas de Dados

Uma rápida revisão



# Estruturas de Dados

- Listas

- Armazena vários itens em uma única variável.

- Mutável

- Indexação

- Fatiamento

- Listas aninhadas

- Métodos comuns



# Estruturas de Dados

- Dicionários

- Armazenam pares chave:valor

- Mutável

- Não permite chaves duplicadas

- Podem ser vistos como mapeamentos

- Acessando itens de um dict

- Dicionários aninhados

- Métodos comuns

# Estruturas de Dados

- Tuplas

Imutabilidade

Para que servem



**ATÉ A  
PRÓXIMA  
AULA!**



**MONDAY**