

Primeiro Programa: Hello Word

- É muito comum, ao apresentar uma nova linguagem, começar com um exemplo simples que mostra na tela as palavras Hello World. Para não perder o costume, antes de adentrar o mundo do Python, vamos ver como outras linguagens de programação implementam esse exemplo:

- Em C:

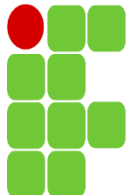
```
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hello, World!\n");
    return 0;
}
```

- Em Java

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

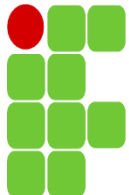
É obrigatório que o código acima esteja em um arquivo chamado *Hello.java*



E em Python ??

- Vamos ver como é o Hello World em Python:

```
>>> print("Hello, World!")  
Hello, World!
```



Sintaxe

- **Indentação:**
 - ➤ Python foi desenvolvido para ser uma linguagem de fácil leitura, com um visual agradável, frequentemente usando palavras e não pontuações como em outras linguagens.
 - ➤ Para a separação de blocos de código, a linguagem usa espaços em branco e indentação ao invés de delimitadores visuais como chaves (C, Java) ou palavras (BASIC, Fortran, Pascal).
 - ➤ Diferente de linguagens com delimitadores visuais de blocos, em Python a indentação é obrigatória. O aumento da indentação indica o início de um novo bloco, que termina da diminuição da indentação.

Sintaxe

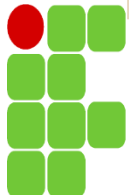
- Indentação:
- O código abaixo está correto para os dois exemplos, mas o analisador léxico verificará se a indentação está coerente.

Indentação incorreta

```
def valor1():  
while True:  
try:  
c = int(raw_input('Primeiro Valor: '))  
return c  
except ValueError:  
print 'Inválido!'
```

Indentação correta

```
def valor1():  
    while True:  
        try:  
            c = int(raw_input('Primeiro Valor: '))  
            return c  
        except ValueError:  
            print 'Inválido!'
```



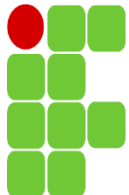
Sintaxe:

- Comentários

- O caractere # marca o início de comentário. Qualquer texto depois do # será ignorado até o fim da linha , com exceção dos comentários funcionais.
- Para comentário em bloco, usa-se três aspas simples ao início fim do bloco.

- Comentário funcional

- É possível usar codificação diferente de ASCII() em arquivos de código Python. A melhor maneira de fazê-lo é através de um comentário adicional logo após a linha #!:
- # -*- coding: encoding -*-
- Definir o interpretador que será utilizado para rodar o programa em sistemas UNIX, através de um comentário começando com “#!” no início do arquivo, que indica o caminho para o interpretador (geralmente a linha de comentário será algo como “#!/usr/bin/envpython”).



Função print()

Usar a letra *P* maiúscula ao invés de minúscula:

- **print()** é uma função nativa do Python. Basta colocar algo dentro dos parênteses que o Python se encarrega de fazer a magia de escrever na tela :)

```
>>> Print("Hello, World!")
Traceback (most recent call last):
...
NameError: name 'Print' is not defined
```

Esquecer de abrir e fechar aspas no texto que é passado para a função print():

```
>>> print(Hello, World!)
Traceback (most recent call last):
...
SyntaxError: invalid syntax
```

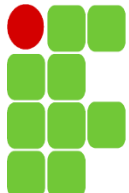
- Erros comuns:

Esquecer de abrir ou fechar aspas:

```
>>> print("Hello, World!)
Traceback (most recent call last):
...
SyntaxError: EOL while scanning string literal
```

Começar com aspas simples e terminar com aspas duplas ou vice-versa:

```
>>> print('Hello, World!")
Traceback (most recent call last):
...
SyntaxError: EOL while scanning string literal
```



Função print()

- **print()** é uma função nativa do Python. Basta colocar algo dentro dos parênteses que o Python se encarrega de fazer a magia de escrever na tela :)
- Erros comuns:

```
>>> print('Hello, World!')
Traceback (most recent call last):
...
IndentationError: unexpected indent

>>>     print('Hello, World!')
Traceback (most recent call last):
...
IndentationError: unexpected indent
```

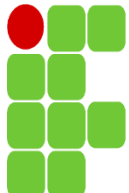
Mas, e se eu precisar usar aspas dentro do texto a ser mostrado na tela? Bem, Caso queira imprimir aspas duplas, envolva tudo com aspas simples e use aspas duplas na parte desejada:

```
>>> print('Python é legal! Mas não o "legal" como dizem pra outras coisas')
Python é legal! Mas não o "legal" como dizem pra outras coisas
```

Caso deseje imprimir aspas simples, faça o contrário (envolva com aspas duplas e use aspas simples onde necessário):

```
>>> print("Python é legal! Mas não o 'legal' como dizem pra outras coisas")
Python é legal! Mas não o 'legal' como dizem pra outras coisas
```

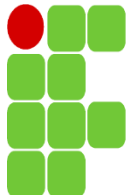
E como faz para imprimir um texto em várias linhas? Bom, para isso precisamos lembrar de um caractere especial, a *quebra de linha*: *n*. Esse *n* é um caractere especial que significa *aqui acaba a linha, o que vier depois deve ficar na linha de baixo*. Por exemplo:



Função print()

- **print()** é uma função nativa do Python. Basta colocar algo dentro dos parênteses que o Python se encarrega de fazer a magia de escrever na tela :)
- Erros comuns:

```
>>> print('Olha esse texto sobre aspas simples e duplas.\nIsso aqui é aspas duplas:  
→"\nIsso aqui é aspas simples: \''  
Olha esse texto sobre aspas simples e duplas.  
Isso aqui é aspas duplas: "  
Isso aqui é aspas simples: '')
```



Operadores matemáticos

- A linguagem Python possui operadores que utilizam símbolos especiais para representar operações de cálculos, assim como na matemática:

- Soma (+)

```
>>> 2 + 3  
5
```

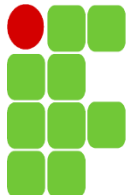
Para utilizar números decimais, use o *ponto* no lugar de vírgula:

```
>>> 3.2 + 2.7  
5.9
```

- Subtração (−)

```
>>> 6 - 4  
2
```

```
>>> 7 - 8  
-1
```



Operadores matemáticos

- A linguagem Python possui operadores que utilizam símbolos especiais para representar operações de cálculos, assim como na matemática:

- Multiplicação (*)

```
>>> 7 * 8  
56
```

```
>>> 2 * 2 * 2  
8
```

- Divisão (/)

```
>>> 100 / 20  
5.0
```

```
>>> 10 / 3  
3.3333333333333335
```



Operadores matemáticos

- A linguagem Python possui operadores que utilizam símbolos especiais para representar operações de cálculos, assim como na matemática:

- Divisão inteira (//)

```
>>> 10 // 3
3
>>> 666 // 137
4
>>> 666 / 137
4.861313868613139
```

- Resto da divisão (%)

```
>>> 10 % 2
0
>>> 10 % 3
1
>>> 666 % 137
118
```



Operadores matemáticos

Agora que aprendemos os operadores aritméticos básicos podemos seguir adiante. Como podemos calcular 2^{10} ? O jeito mais óbvio seria multiplicar o número dois dez vezes:

```
>>> 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2
1024
```

Porém, isso não é muito prático, pois há um operador específico para isso, chamado de *potenciação/exponenciação*:
**

```
>>> 2 ** 10
1024
```

```
>>> 10 ** 3
1000
```

```
>>> (10 ** 800 + 9 ** 1000) * 233
40725400165137782505077408626536591293327155957239892465016990675188990003095518900491634747847069
```

Operadores matemáticos

- E a raiz quadrada?
- Lembrando que $\sqrt{x} = x^{1/2}$, então podemos calcular a raiz quadrada do seguinte modo:

```
>>> 4 ** 0.5
```

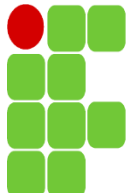
```
2.0
```

- Mas a maneira recomendada para fazer isso é usar a função `sqrt()` da biblioteca `math`:

- ```
>>> import math
```

- ```
>>> math.sqrt(16)
```

- ```
4.0
```



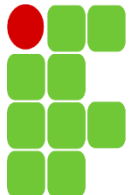
# Operadores matemáticos

- Quando mais de um operador aparece em uma expressão, a ordem de avaliação depende das regras de precedência.
- O Python segue as mesmas regras de precedência da matemática. O acrônimo PEMDAS ajuda a lembrar essa ordem:
  - 1. **P**arênteses
  - 2. **E**xponenciação
  - 3. **M**ultiplicação e **D**ivisão (mesma precedência)
  - 4. **A**dição e **S**ubtração (mesma precedência)

# Comparações

- Os operadores de comparação em Python são:

| Operação | Significado     |
|----------|-----------------|
| <        | menor que       |
| <=       | menor igual que |
| >        | maior que       |
| >=       | maior igual que |
| =        | igual           |
| !=       | diferente       |



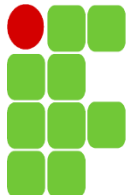
# Atribuição

- Atribuição é o processo de criar uma nova variável e dar um novo valor a ela. Alguns exemplos de atribuições:

```
>>> numero = 11
>>> numero
11
```

```
>>> frase = "Me dá um copo d'água"
>>> frase
"Me dá um copo d'água"
```

```
>>> pi = 3.141592
>>> pi
3.141592
```



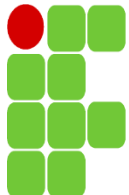


# Nomes de Variáveis

- Atribuição é o processo de criar uma nova variável e dar um novo valor a ela. Alguns exemplos de atribuições:
  - Bons programadores escolhem nomes significativos para as suas variáveis - eles documentam o propósito da variável.
  - Nomes de variáveis podem ter o tamanho que você achar necessário e podem conter tanto letras como números,
  - Porém não podem começar com números. É possível usar letras maiúsculas, porém a convenção é utilizar somente letras minúsculas para nomes de variáveis.

# Nomes de Variáveis

- Atribuição é o processo de criar uma nova variável e dar um novo valor a ela. Alguns exemplos de atribuições:
  - Bons programadores escolhem nomes significativos para as suas variáveis - eles documentam o propósito da variável.
  - Nomes de variáveis podem ter o tamanho que você achar necessário e podem conter tanto letras como números,
  - Porém não podem começar com números. É possível usar letras maiúsculas, porém a convenção é utilizar somente letras minúsculas para nomes de variáveis.
  - O Python possui diversas palavras que são utilizadas na estrutura dos programas, por isso não podem ser utilizadas como nomes de variáveis

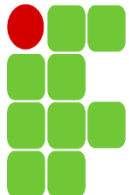


# Nomes de Variáveis

- Atribuição é o processo de criar uma nova variável e dar um novo valor a ela. Alguns exemplos de atribuições:
  - Tentar acessar uma variável sem definí-la anteriormente ocasiona em um «erro de nome».
  - Também podemos atribuir expressões a uma variável:

```
>>> x = 3 * 5 - 2
>>> x
13
>>> y = 3 * x + 10
>>> y
49
>>> z = x + y
>>> z
62
```

```
>>> n = 10
>>> n + 2 # 10 + 2
```



# Atribuição

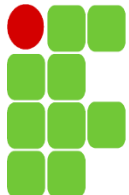
- É importante lembrar que para mudar o valor de uma variável é preciso utilizar a atribuição. Nos dois exemplos anteriores não atribuímos as expressões à `n`, portanto seu valor continuou o mesmo.

Vamos alterar o valor de `n`:

```
>>> n
10
>>> n = n + 2
>>> n
12
>>> 9 - n
-3
```

Outra forma de somar na variável:

```
>>> num = 4
>>> num += 3
>>> num
7
```



# Atribuição múltipla

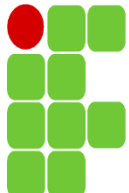
- Uma funcionalidade interessante do Python é que ele permite atribuição múltipla. Isso é muito útil para trocar o valor de duas variáveis:

```
>>> a = 1
>>> b = 200
```

Para fazer essa troca em outras linguagens é necessário utilizar uma variável auxiliar para não perdemos um dos valores que queremos trocar. Vamos começar da maneira mais simples:

```
>>> a = b # perdemos o valor de a
>>> a
200
```

```
>>> b = a # como perdemos o valor de a, b vai continuar com seu valor original de 200
>>> b
200
```

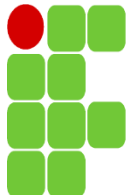


# Atribuição múltipla

- A troca é bem sucedida se usamos uma variável auxiliar:

```
>>> a = 1
>>> b = 200
>>> print(a, b)
1 200

>>> aux = a
>>> a = b
>>> b = aux
>>> print(a, b)
200 1
```

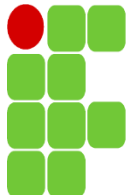


# Atribuição múltipla

- Porém, como o Python permite atribuição múltipla, podemos resolver esse problema de uma forma muito mais simples:

```
>>> a = 1
>>> b = 200
>>> print(a, b)
1 200
```

```
>>> a, b = b, a
>>> print(a, b)
200 1
```



# Atribuição múltipla

- A atribuição múltipla também pode ser utilizada para simplificar a atribuição de variáveis, por exemplo:

```
>>> a, b = 1, 200
>>> print(a, b)
1 200
```

```
>>> a, b, c, d = 1, 2, 3, 4
>>> print(a, b, c, d)
1 2 3 4
```

```
>>> a, b, c, d = d, c, b, a
>>> print(a, b, c, d)
4 3 2 1
```

