Aula
Conceitos Básicos

Programação de Computadores I

Nilson Figueiredo

www.‘’.com.br

Gov-MG – Secretaria de Estado

Planejamento e Gestão.

Agenda

1. Programação e hierarquia de dados
2. Algoritmos e fluxogramas
3. Variáveis e memória
4. Operadores aritméticos
5. Precedência e associatividade
6. Comandos de entrada e saída
7. Formatando strings com *f-strings*
8. Exemplos e exercícios

Programação e hierarquia de dados

Programa vs Programação Estruturada

Antes de mais nada, precisamos entender a diferença entre **programas de computadores** e **programação estruturada**.

- **Programa**: geralmente referido como software, é uma **sequência de instruções** que o computador segue para realizar uma tarefa.

- **Programação Estruturada**: **metodologia** de programação constituída por sequências, desvios e repetições de instruções de uma linguagem de programação.

Hierarquia de dados!

Lembre-se que um computador digital utiliza bits (**0** ou **1**)

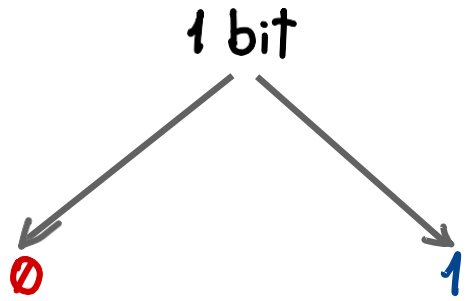
- Assim, os dados são processados por computadores a partir de uma hierarquia de dados que se torna maior e mais complexa em estrutura, a medida que avançamos de bits para caracteres, depois para campos, e assim por diante.
- Ou seja, é a **hierarquia de dados** que permite representar instruções e dados complexos a partir de **bits**.

Bit

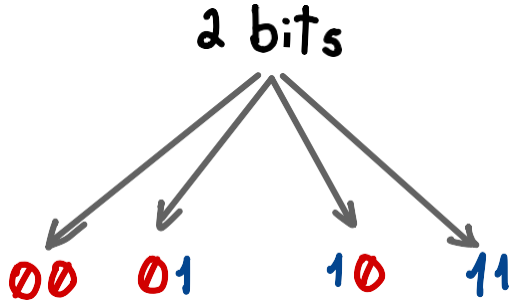
- É o menor item de dado em um computador, o qual pode assumir o valor **0** ou **1** (**Bi**nary Digi**t**).
- É notável como funções importantes de hardware realizam computações por meio da simples mudança do bit 0 para o bit 1, e vice-versa.

Byte

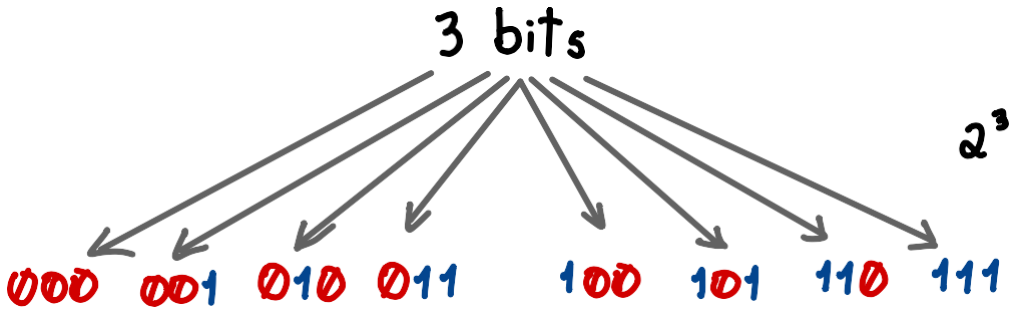
- Conjunto de **8 bits**



2^1 = 2 valores



2^2 = 4 valores



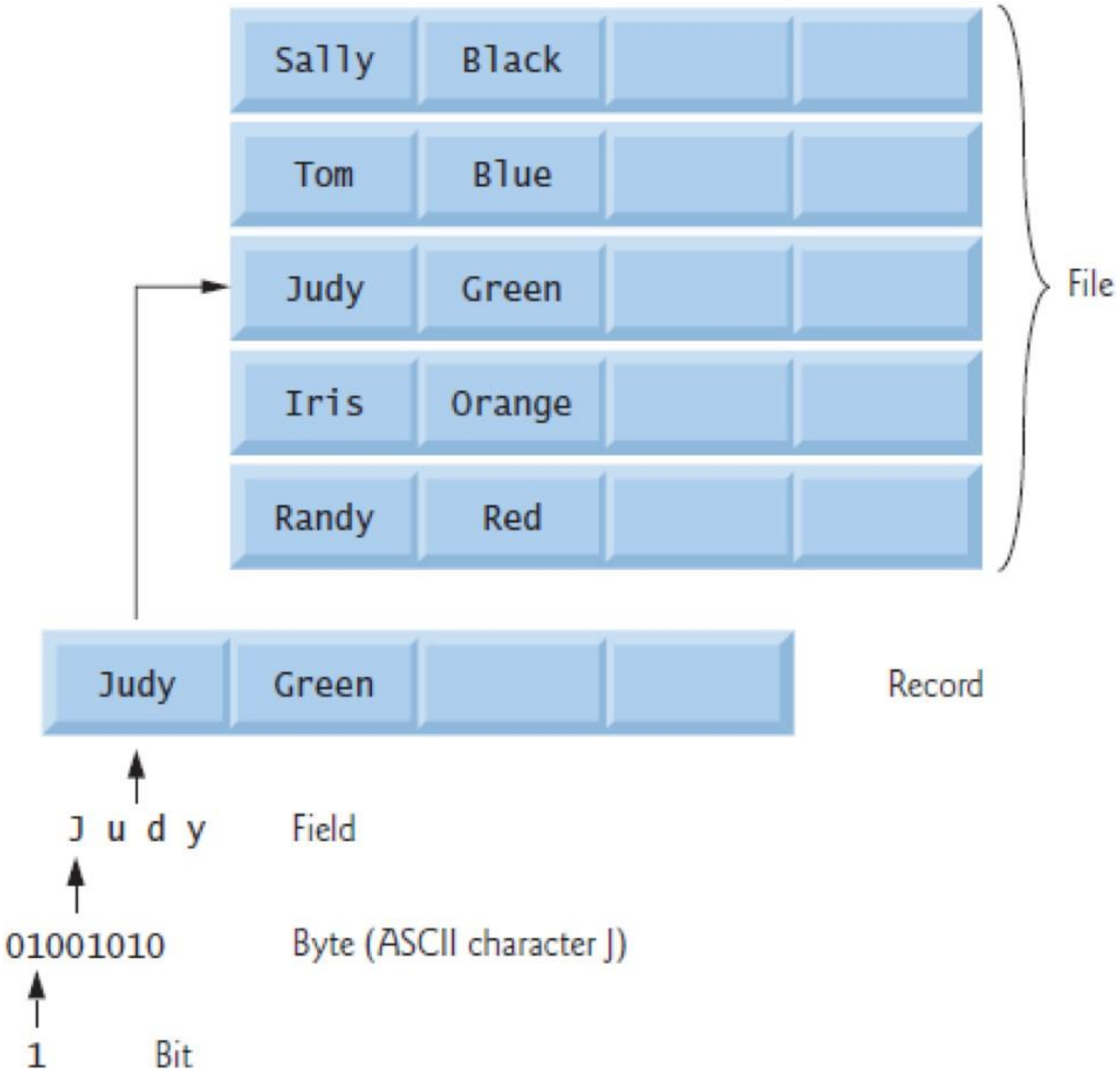
2^3 = 8 valores

Bit vs Byte

- Como um byte tem 8 bits...
  - Cada byte pode representar $2^8 = 256$ valores diferentes
- Quantos valores diferentes 10 bytes podem representar?
  - $2^{80} \approx 1,2 \times 10^{24}$

Sim, podemos representar **muita informação**!

- Mas faremos isso por meio de **hierarquia de dados**.


Sally Black
Tom Blue
Judy Green
Iris Orange
Randy Red
File
Judy Green
Record
J u d y
Field
01001010
Byte (ASCII character J)
1
Bit

Unidades de armazenamento

Prefixo	Abreviação	Valor Geral	Valor em Ciência da Computação	Exemplo de Uso
Kilo	k	1.000	1.024	256 kilobytes
Mega	M	1.000.000	1.048.576	512 megabytes
Giga	G	1.000.000.000	1.073.741.824	1 gigabytes
Tera	T	1.000.000.000.000	1.099.511.627.776	

- 1 **Kilo**byte = 1.024 × 1 Byte
- 1 **Mega**byte = 1.024 × 1 Kilobyte
- 1 **Giga**byte = 1.024 × 1 Megabyte
- 1 **Tera**byte = 1.024 × 1 Gigabyte

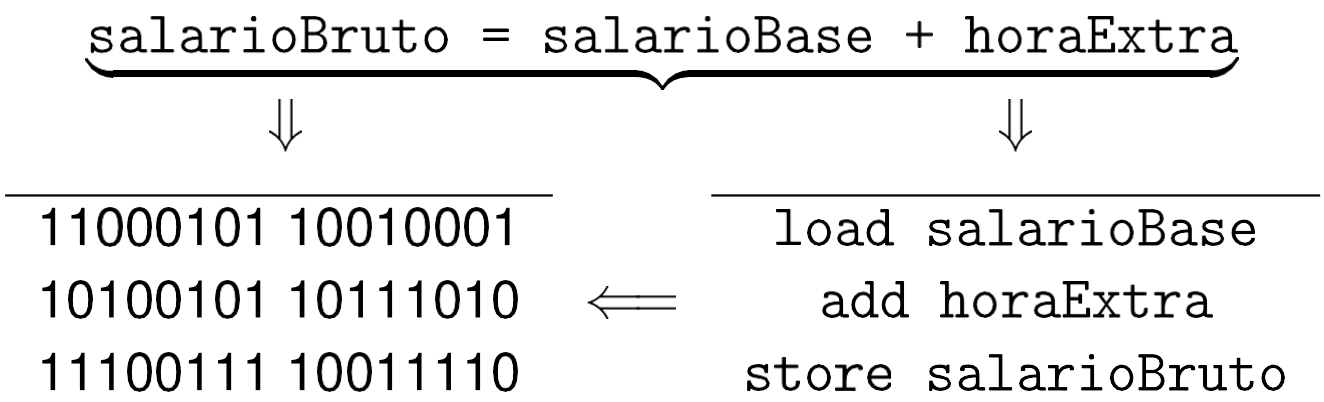
Linguagens de programação

São geralmente divididas em três tipos gerais:

1. Linguagens de **Máquina**
   - Dependentes da arquitetura e definida pelo hardware.
   - ```
     11000101 10010001
     10100101 10111010
     11100111 10011110
     ```
2. Linguagens **Assembly**
   - "Menos difíceis" de entender: utilizam abreviações do inglês.
   - `mov rdi, message`
3. Linguagens de **Alto-nível**
   - Linguagens como C, Fortran, Java, **Python**, etc.
     ```

Linguagens de programação

Note que é necessário traduzir código de **alto nível** para código que o computador é capaz de processar...



Algoritmos e fluxogramas

Algoritmos

Para criar um programa de computador, precisamos de uma **sequência de instruções claras e sem ambiguidade**.

- O conjunto dessas instruções é o que chamamos de **Algoritmo**.
- Utilizamos algoritmos para resolver os mais diversos problemas do dia-a-dia.

Para organização, dividiremos nossos algoritmos em **3 etapas**:

1. **Entrada**
2. **Processamento**
3. **Saída**

Algoritmos

Como converter uma temperatura de celsius para Fahrenheit?

$$T_F = (T_c \cdot \frac{9}{5}) + 32$$

- Entrada:
  - Ler um valor de temperatura em Celsius;
- Processamento
  - Multiplicar o valor lido por 9 e dividir por 5;
  - Somar 32 ao valor encontrado;
- Saída:
  - Imprimir o valor da saída.

Fluxogramas

São muito úteis para representar um algoritmo.

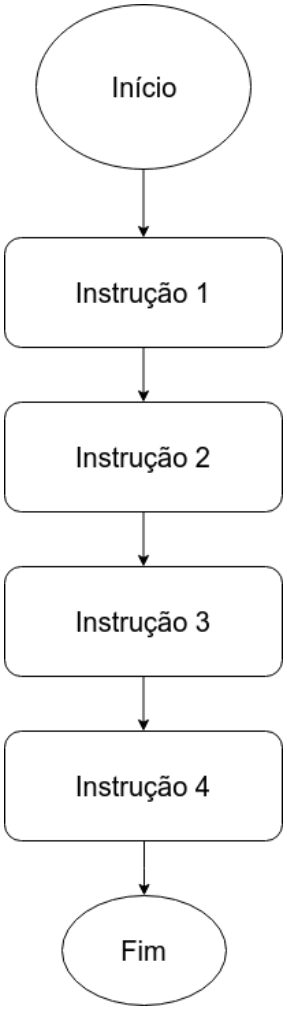
- Representam cada um dos passos realizados por um algoritmo em caixas.
- O caminho tomado pelo algoritmo é representado por setas.
- Esse tipo de representação permite criar visualizações amigáveis e fáceis de ententer.

Por hora, classificaremos as instruções em três tipos de estrutura:

- **Sequência**
- **Decisão**
- **Iteração**

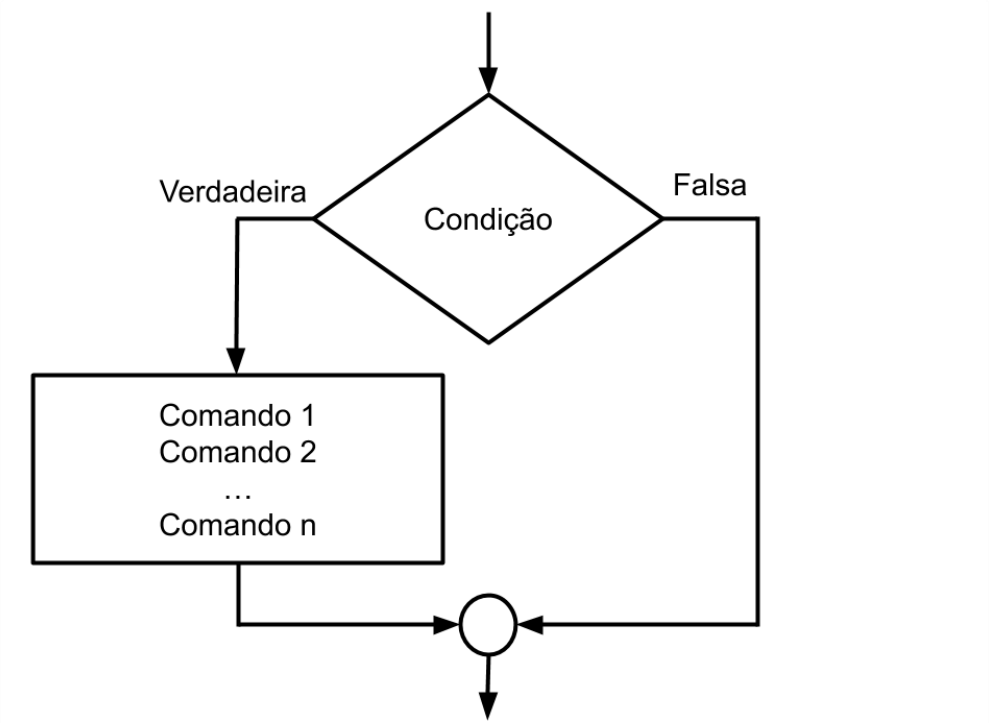
Sequência

- Composta por uma sequência enfileirada de instruções que são executadas em sequência.
- Com esse tipo de estrutura, podemos representar a visão geral do problema que desejamos resolver.



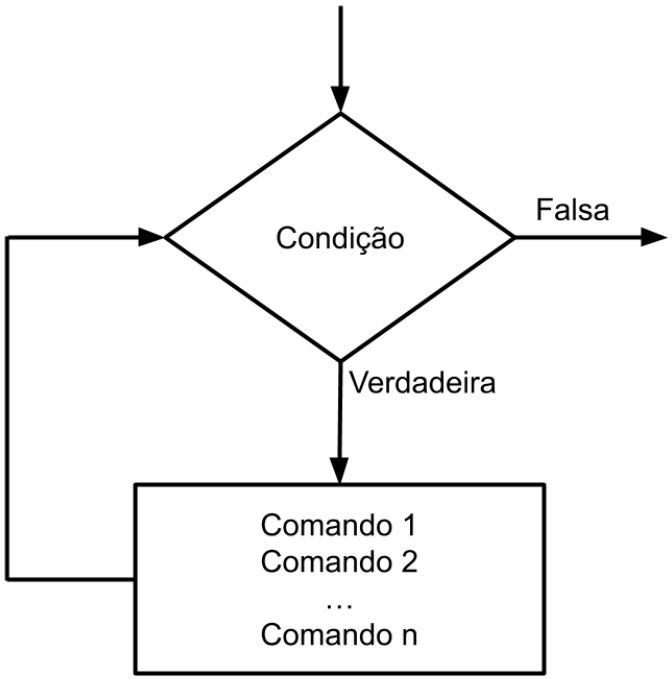
Decisão

- Estruturas de controle que efetuar ações de acordo com a decisão.



Iteração

- Utilizadas para realizar uma tarefa repetitiva de até que alguma condição seja cumprida.

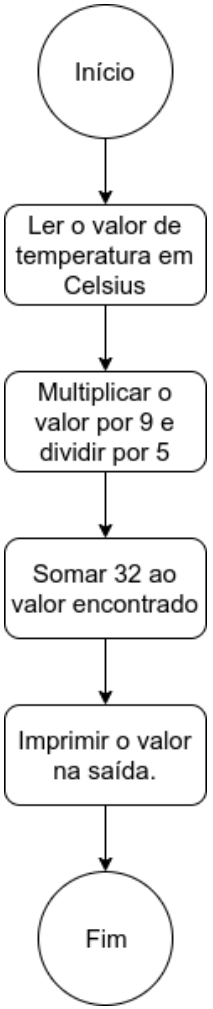


Fluxogramas

Observe na figura que o fluxograma apenas organiza os passos antes representados por uma lista.

- A sequência de passos a ser tomada para atingir o objetivo final é exibida.

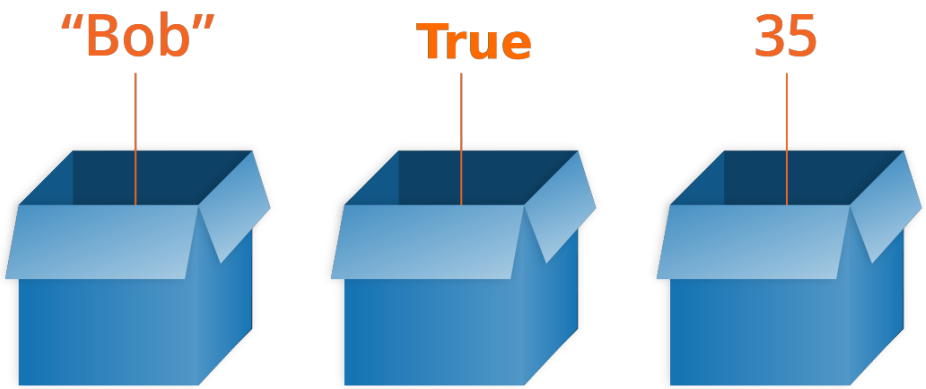
Mas, como fazer este programa em **Python**?



Variáveis e memória

Variáveis

Uma **variável** é utilizada para armazenar dados na memória do computador. Note que variáveis não são os próprios valores, mas são contêineres para valores.



Para que serve uma variável?

- Reservam um espaço na memória do computador para armazenar valores.
- Dependendo do tipo de variável que você criar, você pode armazenar inteiros, caracteres, valores decimais, *strings*, etc.

Nomes de variáveis

Os nomes de variáveis devem obedecer a algumas regras:

- Não podem conter acentos e nem espaços.
- Não podem iniciar com números;
- Além das letras e caracteres alfanuméricos, podem conter alguns caracteres como `_`.

Python diferencia letras maiúsculas e minúsculas, ou seja:

- Nome ≠ nome ≠ NOME

Nomes de variáveis

Variáveis devem ter nomes significativos!

Nomes válidos:

- `idade`
- `nome1`
- `total_de_alunos`

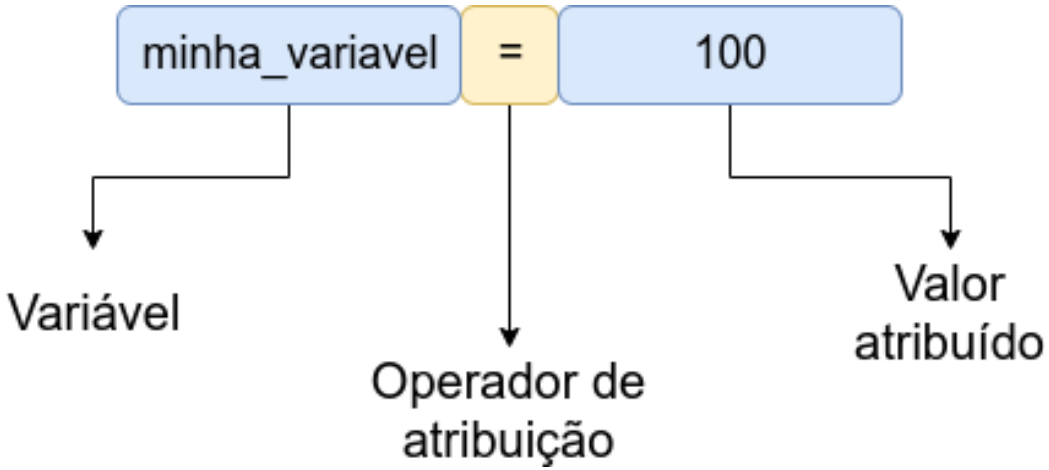
Nomes inválidos:

- `1Aluno` (o primeiro caractere é um algarismo)
- `total de alunos` (tem espaços)
- `#funcionarios` (o uso do caracter # inválido)
- `%valor` (o uso do caracter % inválido)

Atribuição de Valores para uma Variável

A Atribuição de Valores substitui a informação armazenada por uma determinada variável.

Em Python, `=` é o operador de atribuição.



Tipos de variáveis

Quando criamos uma variável:

- Reservamos um espaço na memória do computador para armazenar valores.
- Dependendo do tipo de variável, armazenaremos inteiros, caracteres, *strings*, etc.

Portanto, variáveis podem ser de **tipos** diferentes. Exemplos:

- Tipos númericos (`int`, `float`, `complex`)
- *Strings* (`str`)
- Lógicos (`bool`)
- Listas (`list`)
- Dicionários (`dict`)

Tipos de variáveis

Já vimos e utilizamos alguns **tipos**:

- `int:` inteiro
- `float:` ponto flutuante (número real)
- `str:` *string* (texto)
- `bool`: booleano (verdadeiro ou falso)

Mas... como saber o tipo de uma variável?

- Podemos utilizar a função `type()`

Tipos de variáveis

Exemplo (lembrem-se que `=` é o operador de **atribuição**):

```
1  var1 = 10
2  var2 = 20.5
3  var3 = 'Meu nomé é Túlio!!!'
4  var4 = True
5
6  print(type(var1))
7  print(type(var2))
8  print(type(var3))
9  print(type(var4))
```

```
1  <class 'int'>
2  <class 'float'>
3  <class 'str'>
4  <class 'bool'>
```

Tipos de variáveis

A função `type()` também pode ser utilizada com valores.

Exemplo:

```
print(type(10)) print(type("Olá meu amigo!")) print(type(10.0)) print(type(False))
```

```
# 10 é um 'int'
# "Olá meu amigo!" é uma 'str' #
10.0 é um 'float'
# False é um 'bool'
```

Resultará em:

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

Tipos de variáveis

Em **Python**, as variáveis podem mudar de tipo:

```
# x é inicializado com o valor 100.5 e tipo 'float'
 x = 100.50
print(type(x))

# o tipo de x será alterado para 'str'
x = "Mudei de tipo"
print(type(x))
```

O código acima imprimirá:

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

Operadores aritméticos

Operadores aritméticos

Operadores aritméticos são símbolos especiais, usados para cálculos matemáticos como adição, multiplicação e divisão.

Os principais operadores aritméticos em Python são:

Operador	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
**	Potência
//	Divisão inteira
%	Resto da divisão inteira

Operadores de atribuição

Lembrem-se que **=** é o operador de atribuição.

Podemos combinar o operador de atribuição com alguns operadores aritméticos:

Operador	Exemplo	Equivalente a
=	x = 1	x = 1
+=	x += 1	x = x + 1
-=	x -= 1	x = x - 1
*=	x *= 1	x = x * 1
/=	x /= 1	x = x / 1
%=	x %= 1	x = x % 1

Exemplo

Criar um programa que faça a **soma** de 100 e 200 e em seguida apresente o resultado na tela do computador.

```
1 a = 100
2 b = 200
3 c = a + b
4 print(c)
```

- Perceba que criamos uma variável adicional para armazenar o resultado da soma.
- Poderíamos apresentar o resultado fazendo `print(a + b)`.
  - A expressão `a + b` é avaliada antes da execução do `print`.

A criação de varíaveis para estes casos ajuda a compreender melhor o código, deixando-o mais "**legível**" para o programador.

Exemplo

Criar um programa para calcular **9 elevado ao cubo** e em seguida o programa deve apresentar o resultado na tela do computador.

```
1 a = 9
2 b = 3
3 c = a ** b
4 print(c)
```

Note que poderíamos fazer, simplesmente:

```
1 print(9 ** 3)
```

Uma vez que os valores são fixos (dados pelo enunciado), não é necessário criar as variáveis `a`, `b` e `c` neste caso.

Precedência e associatividade

Agrupando expressões

Deve-se **tomar cuidado** ao agrupar expressões com mais de um operador. Por exemplo:

```
1 a = 3
2 b = 6
3 c = a + b * 2
4 print(c)
```

- O código calcula o dobro da soma de dois números? **Não!**
- Ele calcula o valor de $a + (b \times 2)$. O correto seria:

```
1 a = 3
2 b = 6
3 c = (a + b) * 2
4 print(c)
```

Precedência de operadores

A **precedência** de operadores indica qual operador será executado primeiro:

- Por exemplo, na expressão aritmética `2 + 3 * 6` a subexpressão `3 * 6` é executada primeiro.
- Portanto, tem-se que `2 + 3 * 6` é igual a `20`.

Mas... como saber qual a precedência de operadores?

- Em geral, a precedência segue as regras da matemática!

Precedência de operadores em Python

Operador(es)	Descrição
( )	Parênteses
**	Potência
*, /, //, %	Multiplicação, divisão, resto
+, -	Adição, subtração

Qual será o resultado de `10 ** 1 / 2`?

- Primeiro executamos a operação de potência.
- Em seguida, efetuamos a divisão.
- Resultado: `5`

Associatividade de operadores

A **associatividade** define a regra usada quando os operadores possuem a mesma precedência:

- A ordem de avaliação depende da operação.
  - Será da esquerda para a direita ou o contrário, da direita para a esquerda.
- Na expressão `A - B + C + D`, o primeiro avaliado será `A - B`.
- Em `A ** B ** C ** D`, o primeiro avaliado será `C ** D`.

Passo 1. `y = 2 * 5 * 5 + 3 * 5 + 7` (Multiplicação mais à esquerda)

`2 * 5 é 10`

Passo 2. `y = 10 * 5 + 3 * 5 + 7` (Multiplicação mais à esquerda)

`10 * 5 é 50`

Passo 3. `y = 50 + 3 * 5 + 7` (Multiplicação antes da adição)

`3 * 5 é 15`

Passo 4. `y = 50 + 15 + 7` (Adição mais à esquerda)

`50 + 15 é 65`

Passo 5. `y = 65 + 7` (Última adição)

`65 + 7 é 72`

Passo 6. `y = 72` (Última operação — coloca **72** em **y**)

Comandos de entrada e saída

Imprimindo na saída padrão

Utilizaremos a função `print()` para imprimir na saída.

O uso é muito simples. Exemplos:

```
1 print("Olá mundo!")
2 print(10.5) print(True)
3
```

O código acima imprimirá:

```
1 Olá mundo!
2 10.5  True
3
```

Entrada

Usaremos a função `input()` para entrada de dados.

O uso de `input()` também é bastante simples. Exemplo:

```
1 x = input("Digite seu nome: ")
2 print(x)
```

Notem que `input()` retorna o **texto** digitado na entrada.

- Se quisermos ler um **número**, será necessário **converter**!

Conversão da entrada

Neste início, utilizaremos duas funções para converter a entrada:

- `int()` para converter para um **número inteiro**
- `float()` para converter para um **número real**

Exemplo:

```
1 peso = float(input("Digite seu peso: "))
2 altura = float(input("Digite sua altura: "))
3 imc = peso / (altura * altura)
4
5 print(imc)
```

Formatando strings com *f-strings*

Introdução a *f-strings*

Python 3.6+ fornece uma sintaxe bastante intuitiva para formatar *strings*, chamada *f-strings*

- Para usar, basta adicionar a letra **f** antes da *string*:
  - `f"Esta é uma f-string sem nenhum argumento"`
- *f-strings* permitem misturar frases e valores/variáveis:
  - `f" Esta string contém o número{10}"`
  - `f" Esta string mostra o tipo de 10:{type(10)}"`
  - `f" Esta outra mostra que 10 * 10 ={10*10}"`

Note que é possível inserir código entre `{` e `}` numa *f-string*!

Introdução a *f-strings*

Observe o exemplo a seguir:

```
1 idade = 6
2 frase = f "Eu tenho {idade} anos de idade!"
```

O texto `{idade}` será substituído pelo valor da variável `idade`.

- Esta sintaxe é muito útil!
- Note que colocamos um `f` antes do texto (f de *f-string*).

Exemplo de uso:

```
1 x = 5.6
2 y = 8.8
3 media = (x + y) / 2
4 print(f"A média aritmética de {x} e {y} é {media}")
```

```
1 A média aritmética de 5.6 e 8.8 é 7.2
```

Também poderíamos fazer:

```
1 x = 5.6
2 y = 8.8
3 print(f"A média aritmética de {x} e {y} é {(x + y) / 2}")
```

```
1 A média aritmética de 5.6 e 8.8 é 7.2
```

Especificando a formatação

É possível formatar o texto de usando o caractere `:` em *f-strings*.

A especificação de formatação utiliza os seguintes caracteres:

- `s` para string (sequência de caracteres)
- `d` para decimal (valor numérico inteiro em notação decimal)
- `f` para float (valor numérico real)

Especificando a formatação

Podemos por exemplo indicar a largura do campo (quantidade mínima de caracteres a ser usada).

- `f"Número ocupando 10 espaços:{787:10d}"`
- `f"Nome ocupando 10 espaços:{'Túlio':10s}"`

Podemos ainda indicar a precisão de números reais usando o caracter `.`

- `f"Número real com 2 casas de precisão:{10.836:.2f}"`
- `f"Número real sem casas de precisão:{10.836:.0f}"`
- `f"Número sem casas decimais usando 5 espaços:{10.836:5.0f}"`

```
1  x = 1
2  y = 10
3  print("Sem formatação:")
4  print(f"x = {x}")
5  print(f"y = {y}")
6  print("Com formatação para inteiro (d):")
7  print(f"x = {x:d}")
8  print(f"y = {y:d}")
9  print("Com formatação para inteiro com largura mínima igual a 10 (10d):") print(f"x
10 = {x:10d}")
11 print(f"y = {y:10d}")
```

```
Sem  formatação:
1  x = 1
2  y = 10
3  Com formatação para inteiro (d):
4  x = 1
5  y = 10
6  Com formatação para inteiro com largura mínima igual a 10 (10d):
7  x = 1
8  y = 10
```

```
1  oi = 'Olá Mundo' tchau
2  = 'Adeus mundo'
3  print('Sem formatação:')
4  print(f'{oi}!!!')
5  print(f'{tchau}!!!')
6  print('Com formatação para string (s):')
7  print(f'{oi:s}!!!')
8  print(f'{tchau:s}!!!')
9  print('Com formatação para string com largura mínima igual a 25 (25s):')
10 print(f'{oi:25s}!!!')
11 print(f'{tchau:25s}!!!')
```

```
1  Sem formatação:
2  Olá Mundo!!!
3  Adeus mundo!!!
4  Com formatação para string (s):
5  Olá Mundo!!!
6  Adeus mundo!!!
7  Com formatação para string com largura mínima igual a 25 (25s):
8  Olá Mundo                 !!!
9  Adeus mundo               !!!
```

Alinhamento

Usando *f-strings*, além do número de espaços podemos também definir o **alinhamento** (esquerda, centro ou direita):

- `<`: alinha o conteúdo à esquerda; `print(f"{x:<30d}")`
- `^`: centraliza o conteúdo; `print(f"{x:^30d}")`
- `>`: alinha o conteúdo à direita; - `print(f"{x:>30d}")`

```
1 x = 10
2 print(f"{x:<30d}")
3 print(f"{x:^30d}")
4 print(f"{x:>30d}")
```

```
1 1
2 0                    1
3                      0                    1
                                            0
```

Caracteres especiais em uma *string*

Python permite o uso de sequências de escape dentro de uma string para a inclusão destes caracteres.

- Uma sequência de escape é formada pelo caracter `\` seguido de um código que identifica o caracter especial a ser incluído na string.

- Em uma *f-string* os caracteres `{` e `}` são especiais e indicam a inserção do valor de uma expressão.

- Para inserir { ou } na string precisamos escrevê-los duas vezes em sequência: `{{` e `}}`.

Caracteres especiais em uma *string*

Alguns caracteres especiais:

- `\n`: nova linha.
- `\t`: tabulação horizontal.
- `\"`: aspas. Deve ser usado para inserir aspas em um literal string delimitado por aspas.
- `\'`: apóstrofo. Deve ser usado para inserir um apóstrofo em um literal string delimitado por apóstrofos.
- `\\`: barra inclinada para a esquerda.

Dica: Se uma *string* for prefixada com `r` ou `R`, o mecanismo de sequências de escape é desativado e **todos os caracteres são inseridos literalmente** na string.

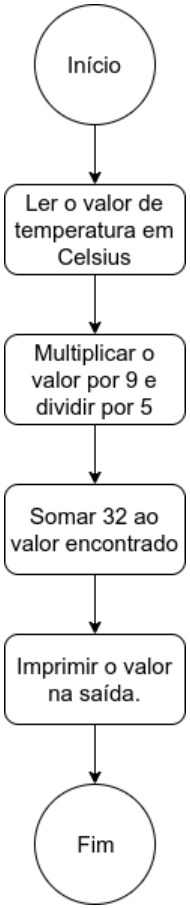
Exemplos de uso:

```
1 print("Ouro\tPreto\nMinas\tGerais")
2 print("\n")
3 print("Uma aspa \" no meio da string")
4 print('Um apóstrofo \' no meio da string')
5 print('Uma barra \\ no meio da string')
6 print(r"Aqui o caracter \ não é especial")
7 print(R'Nesta string \n não é nova linha')
```

```
1 Ouro    Preto
2 Minas   Gerais
3
4
5 Uma aspa " no meio da string
6 Um apóstrofo ' no meio da string
7 Uma barra \ no meio da string Aqui
8 o caracter \ não é especial Nesta
9 string \n não é nova linha
```

Exemplos e exercícios

Exemplo 1



Implemente o fluxograma ao lado.

```
1  # Entrada
2  celsius = float(input("Digite a temperatura em celsius: "))
3
4  # Processamento
5  fahrenheit = celsius * 9 / 5
6  fahrenheit += 32
7
8  # Saída
9  print(f"A temperatura em Fahrenheit é {fahrenheit}")
```

Exemplo 2 (com erro)

Suponha que você deseja resolver o seguinte problema matemático:

> *Calcule o valor de x a partir do valor de y, dado que o dobro de x equivale ao triplo de y.*

O programa abaixo funcionará?

```
1 y = 10
2 2 * x = 3 * y
3 print(x)
```

Não! O código está (muito) **errado**!

- A atribuição deve ser definida como `nome_variavel = valor`

Exemplo 2

Se $2x = 3y$, então $x = \frac{3y}{2}$

Ficou fácil?

```
1 y = 10
2 x = 3 * y / 2
3 print(x)
```

Exemplo 3

Crie um programa que lê os valores de a e b do usuário e em seguida calcula e imprime o resultado da seguinte expressão:

$$x = \frac{a^{\sqrt{b}} + 10}{b}$$

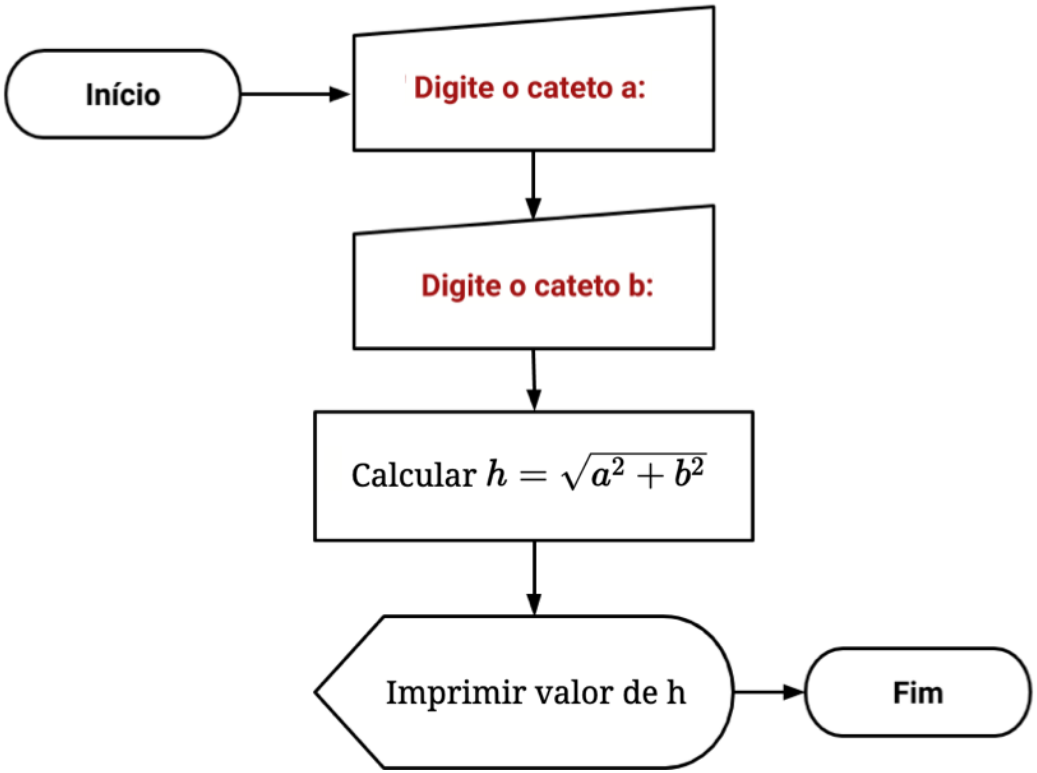
```
# Entrada
a = float(input("Digite o valor de a: "))
b = float(input("Digite o valor de b: "))

# Processamento
x = (a ** (b ** 0.5) + 10) / b

# Saída
print(f"O resultado é x = {x}")
```

Exemplo 4

Implemente o fluxograma a seguir.



Exemplo 4

Implementação do fluxograma:

```
1  # Entrada
2  a = float(input('Digite o cateto a: '))
3  b = float(input('Digite o cateto b: '))
4
5  # Processamento
6  h = (a * a + b * b) ** 0.5
7
8  # Saída
9  print(f'A hipotenusa é h = {h:.2f}')
```

Exemplo 5

Implemente um programa que recebe como entrada o valor de um veículo para em seguida calcular e imprimir o valor do IPVA (Imposto sobre a Propriedade de Veículos Automotores) considerando a alíquota de 4.0%.

```
1  # Entrada
2  valor_veiculo = float(input('Digite o valor do veículo: '))
3
4  # Processamento
5  ipva = valor_veiculo * 0.04
6
7  # Saída
8  print(f'Valor do IPVA: R$ {ipva:.2f}')
```

Exemplo 6

Um prêmio em dinheiro será dividido entre três ganhadores de um concurso.

- O primeiro ganhador receberá 46% do valor total do prêmio O
- segundo ganhador receberá 32% do valor total do prêmio O
- terceiro receberá o restante

Escreva um programa que lê o valor do prêmio e, em seguida, calcula e imprime a quantia que deve ser distribuída para cada um dos ganhadores.

Exemplo 6

```
# Entrada
total = float(input('Digite o valor do prêmio: '))

# Processamento
premio_1 = total * 0.46
premio_2 = total * 0.32
premio_3 = total - premio_1 - premio_2

# Saída
print(f'Prêmio do ganhador 1: R$ {premio_1:.2f}')
print(f'Prêmio do ganhador 2: R$ {premio_2:.2f}')
print(f'Prêmio do ganhador 3: R$ {premio_3:.2f}')
```

Perguntas?