BCM0505-22 – Processamento da Informação

Tipos de dados, variáveis e expressões

Maycon Sambinelli
m.sambinelli@ufabc.edu.br
http://professor.ufabc.edu.br/~m.sambinelli/

Outline

Informação em computadores

Definições importantes

Detalhamento dos tipos em Python

Boas práticas

Pratique!

Informação em computadores

Tipos básicos de dados

Computadores manipulam informação a partir de 4 tipos básicos:

- Inteiro
  - *Ex.:* 4, 7, 1235, -1235, 0, 99999999
- Ponto flutuante
  - *Ex.:* 4, 7, 1235, -1235, 3.23456, 6.7458, -346.234
- Caracteres
  - *Ex.:* ‘a’, ‘h’, ‘P’, ‘%’, ‘(’, ‘~’, ‘_’
- Lógico
  - *Ex.:* Verdadeiro, Falso

Strings

- Sequências de caracteres são chamadas de **strings** e serão representados entre aspas duplas
  - *Ex.:* “Carla”, “Processamento da Informação”, “dia 1 de maio é feriado”
- Strings não são exatamente tipos básicos de dados, porém várias linguagens lidam facilmente com strings

Definições importantes

Literais

Um ***literal*** é um código que representa um valor.

Exemplos: `12356`, `2.567`, `True`, `'Olá, mundo'`

Operadores

Um ***operador*** é um código que representa uma operação que pode ser feita sobre um tipo de dados.

Exemplos: `+`, `*`, `and`, `or`, `>`

Operadores

- É muito importante saber qual tipo de dado o seu programa está processando
  - Cada tipo de dado suporta alguns operadores

Operadores

- É muito importante saber qual tipo de dado o seu programa está processando
  - Cada tipo de dado suporta alguns operadores
- Exemplos em Python:

Tipo	*Valores aceitos*	*Operadores mais comuns*
`int`	Inteiros	`+ - * // % **`
`float`	Pontos flutuantes	`+ - * / *`
`bool`	Lógicos	`and or not`
`str`	Strings	`+`
Vários	Vários	`== != < > <= >=`

Identificadores

Um ***identificador*** é um código que representa um nome.

- Deve ser uma sequência de letras, números e o símbolo underscore `_`
- O primeiro elemento da sequência nunca pode ser um número

Exemplos: `num`, `nome`, `Num1`, `num_2`, `i`, `n`, `aux`

Identificadores

Um ***identificador*** é um código que representa um nome.

- Deve ser uma sequência de letras, números e o símbolo underscore `_`
- O primeiro elemento da sequência nunca pode ser um número

Exemplos: `num`, `nome`, `Num1`, `num_2`, `i`, `n`, `aux`

- Não podem ser **palavras reservadas**
  - `and`, `or`, `not`, `if`, `else`, `elif`, `for`, `while`, `import`, `in`, `def`, `from`, `lambda`
- Alguns outros nomes têm significados especiais e também não devem ser usados
  - `int`, `str`, `float`, `bool`, `sum`, `min`, `max`, `range`, `len`, `input`, `file`, `id`

Identificadores

Um ***identificador*** é um código que representa um nome.

- Deve ser uma sequência de letras, números e o símbolo underscore `_`
- O primeiro elemento da sequência nunca pode ser um número

Exemplos: `num`, `nome`, `Num1`, `num_2`, `i`, `n`, `aux`

- Não podem ser **palavras reservadas**
  - `and`, `or`, `not`, `if`, `else`, `elif`, `for`, `while`, `import`, `in`, `def`, `from`, `lambda`
- Alguns outros nomes têm significados especiais e também não devem ser usados
  - `int`, `str`, `float`, `bool`, `sum`, `min`, `max`, `range`, `len`, `input`, `file`, `id`

Atenção! Python e a várias outras linguagens são *case sensitive*, isto é, o identificador `nome` é **diferente** do identificador `Nome`.

Variáveis

Uma ***variável*** é um nome (identificador) associado a algum valor que pode mudar conforme a computação ocorre.

Variáveis

Uma ***variável*** é um nome (identificador) associado a algum valor que pode mudar conforme a computação ocorre.

- Variáveis são armazenadas na memória de um computador
- Elas podem ser vista como um armário cheio de gavetas
  - Cada gaveta tem uma etiqueta (identificador da variável) e representa uma variável
  - Cada gaveta guarda um único objeto, porém pode ser reutilizada diversas vezes
  - ~~Cada gaveta guarda objetos feitos com o mesmo material (tipo)~~

Variáveis Constantes

Uma ***variável constante*** é um nome (identificador) associado a algum valor que não muda durante a execução do programa.

- Vamos convencionar que os identificadores de constantes serão sempre escritos com letras maiúsculas.

Exemplos: `PI`, `VELOCIDADE_DA_LUZ`

Expressões

Uma ***expressão*** é uma combinação de literais, variáveis e operadores.

- Ela é ***avaliada*** e produz um único valor.
- *Exemplos:*
  - `x - 3`
  - `5 * x`
  - `4 * x - 3.2`
  - `4 * (x - 3.2)`
  - `"Meu nome é"+" "+"Carla"`

Atribuições

Uma ***atribuição*** associa um valor a uma variável.

- Utilizamos o símbolo `=` como o operador da operação de atribuição.
- O lado **esquerdo** de uma atribuição *deve* ser uma única variável.
- O lado **direito** de uma atribuição pode ser qualquer expressão.

Exemplos:

- `x = 123`
  - define o identificador `x` como uma nova variável (se não existir)
  - associa a variável `x` com o valor inteiro 123 (guarda o valor 123 na "gaveta" de rótulo `x`)
- `x = y + 123`
  - avalia a expressão `y + 123` (ocorrerá um erro caso não exista variável `y` ou caso ela não seja numérica)
  - define o identificador `x` como uma nova variável (se não existir)
  - guarda o valor da expressão `y + 123` na variável `x`
- Outras atribuições válidas: `x = y`, `x = y + z - 34`, `nome = "Carla"`

Atribuições

Uma ***atribuição*** associa um valor a uma variável.

- Utilizamos o símbolo `=` como o operador da operação de atribuição.
- O lado **esquerdo** de uma atribuição *deve* ser uma única variável.
- O lado **direito** de uma atribuição pode ser qualquer expressão.

Exemplos:

- `x = 123`
  - define o identificador `x` como uma nova variável (se não existir)
  - associa a variável `x` com o valor inteiro 123 (guarda o valor 123 na "gaveta" de rótulo `x`)
- `x = y + 123`
  - avalia a expressão `y + 123` (ocorrerá um erro caso não exista variável `y` ou caso ela não seja numérica)
  - define o identificador `x` como uma nova variável (se não existir)
  - guarda o valor da expressão `y + 123` na variável `x`
- Outras atribuições válidas: `x = y`, `x = y + z - 34`, `nome = "Carla"`

Pergunta: O comando `x = x + 1` é válido???

Teste de mesa

O ***teste de mesa*** é uma técnica consagrada que nos ajuda a entender o comportamento de um programa.

- É uma forma manual de verificar a lógica de um algoritmo e simular a sua execução.

Teste de mesa

O ***teste de mesa*** é uma técnica consagrada que nos ajuda a entender o comportamento de um programa.

- É uma forma manual de verificar a lógica de um algoritmo e simular a sua execução.

Em geral consiste de uma tabela com várias linhas (passos do algoritmo) e colunas (variáveis).

Comando	x	y
`x = 5`	5	
`y = 3`	5	3
`x = y - 2`	1	3
`x = x + 8`	9	3
`y = x`	9	9

Teste de mesa

O ***teste de mesa*** é uma técnica consagrada que nos ajuda a entender o comportamento de um programa.

- É uma forma manual de verificar a lógica de um algoritmo e simular a sua execução.

Em geral consiste de uma tabela com várias linhas (passos do algoritmo) e colunas (variáveis).

Comando	x	y
`x = 5`	5	
`y = 3`	5	3
`x = y - 2`	1	3
`x = x + 8`	9	3
`y = x`	9	9

No decorrer do curso, vamos incrementar/melhorar a forma como faremos testes de mesa.

Troca de valores

Antes de continuarmos: no exemplo anterior, o valor 3, que estava associado à variável `y`, se perdeu.

Troca de valores

Antes de continuarmos: no exemplo anterior, o valor 3, que estava associado à variável `y`, se perdeu.

E se quisséssemos mantê-lo?

Troca de valores

Antes de continuarmos: no exemplo anterior, o valor 3, que estava associado à variável y, se perdeu.

E se quisséssemos mantê-lo?

Um conceito importante em programação é o de ***troca*** de valores entre variáveis.

- Suponha que temos duas variáveis x e y e que ambas já têm valores atribuídos.
- Por algum motivo que não vem ao caso, precisamos trocar os valores entre essas variáveis.
- Assim, se x vale 3 e y vale 5, queremos realizar operações para que x valha 5 e y valha 3.

Troca de valores

Antes de continuarmos: no exemplo anterior, o valor 3, que estava associado à variável `y`, se perdeu.

E se quisséssemos mantê-lo?

Um conceito importante em programação é o de ***troca*** de valores entre variáveis.

- Suponha que temos duas variáveis `x` e `y` e que ambas já têm valores atribuídos.
- Por algum motivo que não vem ao caso, precisamos trocar os valores entre essas variáveis.
- Assim, se `x` vale 3 e `y` vale 5, queremos realizar operações para que `x` valha 5 e `y` valha 3.

Comando	x	y	aux
	5	3	
`aux = x`	5	3	5
`x = y`	3	3	5
`y = aux`	3	5	5

Detalhamento dos tipos em Python

Tipos em Python

- Python é uma linguagem **dinamicamente tipada**, assim não é necessário anotar o tipo de variáveis (e funções).
  - O oposto da linguagem dinamicamente tipada é chamado de linguagem **estaticamente tipada**.

Estaticamente Tipada

```
int a = 10;
float b = 3.15;
```

Python

```
a = 10
b = 3.15
```

Tipos em Python

- Python é uma linguagem **dinamicamente tipada**, assim não é necessário anotar o tipo de variáveis (e funções).
  - O oposto da linguagem dinamicamente tipada é chamado de linguagem **estaticamente tipada**.

Estaticamente Tipada

```
int a = 10;
float b = 3.15;
```

Python

```
a = 10
b = 3.15
```

- **Toda** expressão em Python **tem um tipo!**

Tipos em Python

- Python é uma linguagem **dinamicamente tipada**, assim não é necessário anotar o tipo de variáveis (e funções).
  - O oposto da linguagem dinamicamente tipada é chamado de linguagem **estaticamente tipada**.

Estaticamente Tipada

```
int a = 10;
float b = 3.15;
```

Python

```
a = 10
b = 3.15
```

- **Toda** expressão em Python **tem um tipo!**
- Podemos descobrir o tipo de uma expressão com a função `type()`.

Tipos em Python

- Python é uma linguagem **dinamicamente tipada**, assim não é necessário anotar o tipo de variáveis (e funções).
  - O oposto da linguagem dinamicamente tipada é chamado de linguagem **estaticamente tipada**.

Estaticamente Tipada

```
int a = 10;
float b = 3.15;
```

Python

```
a = 10
b = 3.15
```

- **Toda** expressão em Python **tem um tipo!**
- Podemos descobrir o tipo de uma expressão com a função `type()`.
  - *Exemplos:* `type(b)`, `type(3 + 4 * 12)`, `type("Hello World!")`

Lógicos - Especificação e Operadores

- Um literal do tipo `bool` só pode ser um dentre dois: `True` ou `False`
  - `True` representa o valor **Verdadeiro**
  - `False` representa o valor **Falso**

Lógicos - Especificação e Operadores

- Um literal do tipo `bool` só pode ser um dentre dois: `True` ou `False`
  - `True` representa o valor **Verdadeiro**
  - `False` representa o valor **Falso**

Sejam *a* e *b* variáveis ou literais do tipo `bool`:

a	*b*	`not` *a*	*a* `and` *b*	*a* `or` *b*
`True`	`True`	`False`	`True`	`True`
`True`	`False`	`False`	`False`	`True`
`False`	`True`	`True`	`False`	`True`
`False`	`False`	`True`	`False`	`False`

Inteiros - Especificação

- Um literal do tipo `int` pode ser especificado escrevendo-se uma sequência de digitos entre 0 e 9.

Inteiros - Especificação

- Um literal do tipo `int` pode ser especificado escrevendo-se uma sequência de digitos entre 0 e 9.
  - A sequência **não** pode começar com o dígito 0.

Inteiros - Especificação

- Um literal do tipo `int` pode ser especificado escrevendo-se uma sequência de digitos entre 0 e 9.
  - A sequência **não** pode começar com o dígito 0.
  - A memória não é infinita: em geral, o intervalo de inteiros que conseguem ser representados por computadores é limitado

Inteiros - Especificação

- Um literal do tipo `int` pode ser especificado escrevendo-se uma sequência de digitos entre 0 e 9.
  - A sequência **não** pode começar com o dígito 0.
  - A memória não é infinita: em geral, o intervalo de inteiros que conseguem ser representados por computadores é limitado
  - Em Java e C, por exemplo, variam de -2^{31} (-2147483648) a $2^{31} - 1$ (2147483647)

Inteiros - Especificação

- Um literal do tipo `int` pode ser especificado escrevendo-se uma sequência de digitos entre 0 e 9.
  - A sequência **não** pode começar com o dígito 0.
  - A memória não é infinita: em geral, o intervalo de inteiros que conseguem ser representados por computadores é limitado
  - Em Java e C, por exemplo, variam de -2^{31} (-2147483648) a $2^{31} - 1$ (2147483647)
  - Em Python normalmente não tem-se essa preocupação, mas não podemos usar *muitos* números inteiros *muito grandes*

Inteiros - Operadores Aritméticos

Sejam *a* e *b* dois inteiros (variáveis, literais ou expressões):

- `+` *a*: tem valor *a* (não altera o valor)

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$
- a `*` b: tem valor $a \times b$

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$
- a `*` b: tem valor $a \times b$
- a `/` b: tem o valor de $\frac{a}{b}$

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+ a`: tem valor a (não altera o valor)
- `- a`: tem valor $-a$ (inverte o sinal)
- `a + b`: tem valor $a + b$
- `a - b`: tem valor $a - b$
- `a * b`: tem valor $a \times b$
- `a / b`: tem o valor de $\frac{a}{b}$
- `a // b`: tem o valor de $\lfloor \frac{a}{b} \rfloor$

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$
- a `*` b: tem valor $a \times b$
- a `/` b: tem o valor de $\frac{a}{b}$
- a `//` b: tem o valor de $\lfloor \frac{a}{b} \rfloor$
  - **Ex:** `7 // 3` assume o valor 3

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$
- a `*` b: tem valor $a \times b$
- a `/` b: tem o valor de $\frac{a}{b}$
- a `//` b: tem o valor de $\lfloor \frac{a}{b} \rfloor$
  - **Ex:** `7 // 3` assume o valor 3
- a `%` b: tem o valor do resto da divisão inteira de a por b

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$
- a `*` b: tem valor $a \times b$
- a `/` b: tem o valor de $\frac{a}{b}$
- a `//` b: tem o valor de $\lfloor \frac{a}{b} \rfloor$
  - **Ex:** `7 // 3` assume o valor 3
- a `%` b: tem o valor do resto da divisão inteira de a por b
  - **Ex:** `8 % 3` assume o valor 2

Inteiros - Operadores Aritméticos

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- `+` a: tem valor a (não altera o valor)
- `-` a: tem valor $-a$ (inverte o sinal)
- a `+` b: tem valor $a + b$
- a `-` b: tem valor $a - b$
- a `*` b: tem valor $a \times b$
- a `/` b: tem o valor de $\frac{a}{b}$
- a `//` b: tem o valor de $\lfloor \frac{a}{b} \rfloor$
  - **Ex:** `7 // 3` assume o valor 3
- a `%` b: tem o valor do resto da divisão inteira de a por b
  - **Ex:** `8 % 3` assume o valor 2
- a `**` b: tem valor a^b

Inteiros - Operadores Relacionais

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- a `<` b: tem o valor `True` se $a < b$ e `False`, se $a \geq b$

Inteiros - Operadores Relacionais

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- a `<` b: tem o valor `True` se $a < b$ e `False`, se $a \geq b$
- a `<=` b: tem o valor `True` se $a \leq b$ e `False`, se $a > b$

Inteiros - Operadores Relacionais

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- a `<` b: tem o valor `True` se $a < b$ e `False`, se $a \geq b$
- a `<=` b: tem o valor `True` se $a \leq b$ e `False`, se $a > b$
- a `==` b: tem o valor `True` se $a = b$ e `False`, se $a \neq b$

Inteiros - Operadores Relacionais

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- a `<` b: tem o valor `True` se $a < b$ e `False`, se $a \geq b$
- a `<=` b: tem o valor `True` se $a \leq b$ e `False`, se $a > b$
- a `==` b: tem o valor `True` se $a = b$ e `False`, se $a \neq b$
- a `!=` b: tem o valor `True` se $a \neq b$ e `False`, se $a = b$

Inteiros - Operadores Relacionais

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- a `<` b: tem o valor `True` se $a < b$ e `False`, se $a \geq b$
- a `<=` b: tem o valor `True` se $a \leq b$ e `False`, se $a > b$
- a `==` b: tem o valor `True` se $a = b$ e `False`, se $a \neq b$
- a `!=` b: tem o valor `True` se $a \neq b$ e `False`, se $a = b$
- a `>` b: tem o valor `True` se $a > b$ e `False`, se $a \leq b$

Inteiros - Operadores Relacionais

Sejam a e b dois inteiros (variáveis, literais ou expressões):

- a `<` b: tem o valor `True` se $a < b$ e `False`, se $a \geq b$
- a `<=` b: tem o valor `True` se $a \leq b$ e `False`, se $a > b$
- a `==` b: tem o valor `True` se $a = b$ e `False`, se $a \neq b$
- a `!=` b: tem o valor `True` se $a \neq b$ e `False`, se $a = b$
- a `>` b: tem o valor `True` se $a > b$ e `False`, se $a \leq b$
- a `>=` b: tem o valor `True` se $a \geq b$ e `False`, se $a < b$

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:
  - com uma sequência de digitos entre 0 e 9 com um ponto “.” em algum momento

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:
  - com uma sequência de digitos entre 0 e 9 com um ponto "." em algum momento
  - em notação científica usando-se ponto "." e *"e"*: `6.022e23` representa 6.022×10^{23}

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:
  - com uma sequência de digitos entre 0 e 9 com um ponto "." em algum momento
  - em notação científica usando-se ponto "." e *"e"*: `6.022e23` representa 6.022×10^{23}
- A memória não é infinita, mas os números reais são!

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:
  - com uma sequência de digitos entre 0 e 9 com um ponto "." em algum momento
  - em notação científica usando-se ponto "." e *"e"*: `6.022e23` representa 6.022×10^{23}
- A memória não é infinita, mas os números reais são!
  - Definitivamente não conseguimos representar todos

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:
  - com uma sequência de digitos entre 0 e 9 com um ponto "." em algum momento
  - em notação científica usando-se ponto "." e *"e"*: `6.022e23` representa 6.022×10^{23}
- A memória não é infinita, mas os números reais são!
  - Definitivamente não conseguimos representar todos
  - Podemos ter problemas com precisão (p. ex., `10/3` tem valor `3.3333333333333335`)

Pontos flutuantes - Especificação e Operadores

- Um literal to tipo `float` pode ser especificado de duas formas:
  - com uma sequência de digitos entre 0 e 9 com um ponto "." em algum momento
  - em notação científica usando-se ponto "." e *"e"*: `6.022e23` representa 6.022×10^{23}
- A memória não é infinita, mas os números reais são!
  - Definitivamente não conseguimos representar todos
  - Podemos ter problemas com precisão (p. ex., `10/3` tem valor `3.3333333333333335`)
- Os operadores aritméticos e relacionais para o tipo `float` são definidos de forma análoga ao tipo `int`

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como
    - *tabulação*: `'\t'`

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como
    - *tabulação*: `'\t'`
    - *nova linha*: `'\n'`

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como
    - *tabulação*: `'\t'`
    - *nova linha*: `'\n'`
    - *contrabarra*: `'\\'`

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como
    - *tabulação*: `'\t'`
    - *nova linha*: `'\n'`
    - *contrabarra*: `'\\'`
    - *aspas simples* (caso dentro de aspas simples): `'\''`

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como
    - *tabulação*: `'\t'`
    - *nova linha*: `'\n'`
    - *contrabarra*: `'\\'`
    - *aspas simples* (caso dentro de aspas simples): `'\''`
    - *aspas duplas* (caso dentro de aspas duplas): `"\""`

Strings - Especificação

- Um literal to tipo `float` pode ser especificado de duas formas:
- Um literal do tipo `str` pode ser definido envolvendo uma sequência de caracteres por aspas duplas (ou simples).
  - Em geral, os caracteres são letras, números, símbolos ou espaços.
  - Pode-se ainda usar a contrabarra `\` (***caractere de escape***) para especificar caracteres especiais, como
    - *tabulação*: `'\t'`
    - *nova linha*: `'\n'`
    - *contrabarra*: `'\\'`
    - *aspas simples* (caso dentro de aspas simples): `'\''`
    - *aspas duplas* (caso dentro de aspas duplas): `"\""`
  - Exemplos: `"Meu nome é Carla!"`, `"'Vendo' o carro"`, `"\"Vendo\" o carro"`, `"Que dia é hoje?\nNão sei"`

Strings - Operadores

- O operador `+` pode ser usado para ***concatenar*** duas strings.
  - Ele recebe dois objetos do tipo `str` e produz um único objeto `str`

Expressão	Valor
`"Olá, " + "Mundo!"`	`"Olá, Mundo!"`
`"Olá," + "Mundo!"`	`"Olá,Mundo!"`
`"Olá," + " " + "Mundo!"`	`"Olá, Mundo!"`
`"123" + "456"`	`"123456"`
`"123" + "+" + "456"`	`"123+456"`
`"123" + " + " + "456"`	`"123 + 456"`
`"123" + 456`	*run-time error*

Precedência e Associatividade de Operadores em Python I

Se uma *expressão* tem mais de um operador, então usamos a *precedência* e a *associativade* de operadores para determinar a ordem no qual os operadores serão aplicados para avaliar o valor a expressão

- A **precedência** estabelece uma ordem de prioridade na aplicação dos diferentes operadores

Precedência e Associatividade de Operadores em Python I

Se uma *expressão* tem mais de um operador, então usamos a *precedência* e a *associativade* de operadores para determinar a ordem no qual os operadores serão aplicados para avaliar o valor a expressão

- A **precedência** estabelece uma ordem de prioridade na aplicação dos diferentes operadores
  - **Ex:** multiplicação é feita antes da adição

Precedência e Associatividade de Operadores em Python I

Se uma *expressão* tem mais de um operador, então usamos a *precedência* e a *associativade* de operadores para determinar a ordem no qual os operadores serão aplicados para avaliar o valor a expressão

- A **precedência** estabelece uma ordem de prioridade na aplicação dos diferentes operadores
  - **Ex:** multiplicação é feita antes da adição
- A **associatividade**, que pode ser ***à esquerda*** ou ***à direita***, define como os operandos de um operadores são processados (se da esquerda para a direita ou da direita para a esquerda)

Precedência e Associatividade de Operadores em Python I

Se uma *expressão* tem mais de um operador, então usamos a *precedência* e a *associativade* de operadores para determinar a ordem no qual os operadores serão aplicados para avaliar o valor a expressão

- A **precedência** estabelece uma ordem de prioridade na aplicação dos diferentes operadores
  - **Ex:** multiplicação é feita antes da adição
- A **associatividade**, que pode ser ***à esquerda*** ou ***à direita***, define como os operandos de um operadores são processados (se da esquerda para a direita ou da direita para a esquerda)
  - O operador `-` é associativo à esquerda, então `7 - 3 - 2` é avaliada como `((7 - 3) - 2)`, que resulta no valor `2`

Precedência e Associatividade de Operadores em Python I

Se uma *expressão* tem mais de um operador, então usamos a *precedência* e a *associativade* de operadores para determinar a ordem no qual os operadores serão aplicados para avaliar o valor a expressão

- A **precedência** estabelece uma ordem de prioridade na aplicação dos diferentes operadores
  - **Ex:** multiplicação é feita antes da adição
- A **associatividade**, que pode ser ***à esquerda*** ou ***à direita***, define como os operandos de um operadores são processados (se da esquerda para a direita ou da direita para a esquerda)
  - O operador `-` é associativo à esquerda, então `7 - 3 - 2` é avaliada como `((7 - 3) - 2)`, que resulta no valor `2`
  - O operador `**` é associativo à direita, então `2 ** 3 ** 4` é avaliada como `(2 ** (3 ** 4))`, que resulta no valor `2417851639229258349412352`

Precedência e Associatividade de Operadores em Python II

Quanto menor o valor, maior a prioridade!

Precedência	Operadores	Associatividade
1	`( )`	à esquerda
2	`**`	à direita
3	`+x -x`	à direita
4	`* / // %`	à esquerda
5	`+ -`	à esquerda
6	`< <= == >= > !=`	à esquerda
7	`not`	à direita
8	`and`	à esquerda
9	`or`	à esquerda

Precedência e Associatividade de Operadores em Python III

Ex.: `a + b - c * d * e ** f - g // h` é avaliado na seguinte ordem:

Precedência e Associatividade de Operadores em Python III

Ex.: `a + b - c * d * e ** f - g // h` é avaliado na seguinte ordem:

- Avalia-se `e ** f`; seja tal valor `i`: `a + b - c * d * i - g // h`
- Avalia-se `c * d`; seja tal valor `j`: `a + b - j * i - g // h`
- Avalia-se `j * i`; seja tal valor `k`: `a + b - k - g // h`
- Avalia-se `g // h`; seja tal valor `l`: `a + b - k - l`
- Avalia-se `a + b`; seja tal valor `m`: `m - k - l`
- Avalia-se `m - k`; seja tal valor `n`: `n - l`
- Avalia-se `n - l`

Conversões entre tipos

- A função `str()` converte números em strings.

Conversões entre tipos

- A função `str()` converte números em strings.
- A função `int()` converte strings em inteiros ou trunca pontos flutuantes.

Conversões entre tipos

- A função `str()` converte números em strings.
- A função `int()` converte strings em inteiros ou trunca pontos flutuantes.
- A função `float()` converte strings ou inteiros em pontos flutuantes.

Conversões entre tipos

- A função `str()` converte números em strings.
- A função `int()` converte strings em inteiros ou trunca pontos flutuantes.
- A função `float()` converte strings ou inteiros em pontos flutuantes.
- A função `round()` arredonda pontos flutuantes para o inteiro mais próximo.

Conversões entre tipos

- A função `str()` converte números em strings.
- A função `int()` converte strings em inteiros ou trunca pontos flutuantes.
- A função `float()` converte strings ou inteiros em pontos flutuantes.
- A função `round()` arredonda pontos flutuantes para o inteiro mais próximo.

Conversões entre tipos

- A função `str()` converte números em strings.
- A função `int()` converte strings em inteiros ou trunca pontos flutuantes.
- A função `float()` converte strings ou inteiros em pontos flutuantes.
- A função `round()` arredonda pontos flutuantes para o inteiro mais próximo.

Expressão	Valor
`str(123)`	`"123"`
`str(3.1415)`	`"3.1415"`
`"123" + str(456)`	`"123456"`
`int("456")`	`456`
`int("45.6")`	*run-time error*
`int(5/2)`	`2`
`float("45.6")`	`45.6`
`float("5/2")`	*run-time error*
`float(4)`	`4.0`
`round(4.2)`	`4`
`round(4.8)`	`5`

Conversão de Tipo Implícita

Em Python, quando realizamos uma operação aritmética envolveldo um operando do tipo `int` e outro do tipo `float`, o tipo do valor da expressão resultante é `float`. Isso é chamado de **promoção de tipo**, ou conversão implícita.

Conversão de Tipo Implícita

Em Python, quando realizamos uma operação aritmética envolveldo um operando do tipo `int` e outro do tipo `float`, o tipo do valor da expressão resultante é `float`. Isso é chamado de **promoção de tipo**, ou conversão implícita.

Cada uma das expressões abaixo resulta em resultado com tipo `float`:

- `4 + 6.5`
- `3.2 * 4`
- `4 / 2.0`
- `3.0 * 2`
- `5 / 2`

Boas práticas

Comentários

- Qualquer conteúdo entre um símbolo `#` e o final da linha será ignorado pelo interpretador
- Qualquer conteúdo entre `"""` e `"""` também
- Usamos comentários para:
  - Explicar a ideia geral de um trecho de código
  - Desabilitar um trecho do código para testes
  - Definir autoria e licença do código
  - Etc
- Comentários não devem ser grandes demais, pois um bom código é simples e autoexplicativo
  - Os identificadores (nomes de variáveis) devem ser significativos e cuidadosamente escolhidos

Pratique!

Exercícios

IMPORTANTE: Faça use o operador de seleção `if` em sua solução

1. Escreva um programa que converta o valor de uma temperatura de Fahrenheit para Celsius. A seguinte fórmula estabelece a relação entre o valor da temperatura em Fahrenheit T_F e Celsius T_C.

$$T_C = 5\frac{T_F - 32}{9}.$$

Utilize o código seguinte para ler um número real

```
1 float(input())
```

1. Escreva um programa que decida se um dado ano é bissexto ou não. Todos os anos múltiplos de 400 são bissextos. Não sendo múltiplo de 400, são bissextos todos os anos múltiplos de 4 mas que não são múltiplos de 100.
   Utilize o código seguinte para ler um número inteiro

```
1 int(input())
```

1. Escreva um programa que recebe um número e encontre o primeiro número ímpar que é **maior ou igual** a ele.