

Revisão – Python

Prof. Fabio Fernando Kobs, Dr.

Python – Instalação

Linguagem de Programação

Python (www.anaconda.com/distribution)

Ambiente Integrado para Desenvolvimento (IDE)

PyCharm (<https://www.jetbrains.com/pycharm/>)

Ferramenta para os Diagramas em UML

Umbrello (<https://umbrello.kde.org/installation.php>)

Operadores Relacionais:

- Como exemplos de operadores relacionais matematicamente conhecidos tem-se:

Operação	Matemática	Operador em Python
Igualdade	$=$	<code>==</code> ou <code>is</code>
Diferente	\neq	<code>!=</code> ou <code>is not</code>
Maior que	$>$	<code>></code>
Menor que	$<$	<code><</code>
Maior ou igual a	\geq	<code>>=</code>

Operadores Lógicos:

- Tem-se:

Operador	Matemática	Notação em Python	Ordem de hierarquia
Negação	não	not	1º
Conjunção	e	and	2º
Disjunção	ou	or	3º

Expressões

Critérios de precedência dos operadores:

1. A seguir, relacionam-se os critérios de precedência dos operadores.
2. Se precisar alterar esta hierarquia, deve-se usar os parênteses.

Hierarquia	
Primeiro	Parênteses e funções
Segundo	Potência e resto
Terceiro	Multiplicação e divisão
Quarto	Adição e subtração
Quinto	Operadores relacionais
Sexto	Operadores lógicos

Funções – **print**

- A função **print** informa que será exibido algo na tela, ou seja, a função exibe uma mensagem na tela do computador.
- Exemplo:
`print("Oi")`
- Os parênteses são utilizados para separar os parâmetros de uma função, e as aspas ou apóstrofes para indicar o início e o fim do texto da mensagem.

Funções – Tamanho de uma *string*

- Pode ser obtido utilizando-se a função **len**, retornando um valor do tipo inteiro.
- Se a string é vazia (representada por “”, ou seja, duas aspas sem nada entre elas), seu tamanho é igual a zero.
- Exemplos:

```
>>> print(len("A"))
```

```
1
```

```
>>> print(len('AB'))
```

```
2
```

```
>>> print(len(''))
```

```
0
```

```
>>> print(len("O rato roeu a roupa"))
```

Funções – Acessar caracteres de uma *string*

- Deve-se informar o índice ou posição do caractere entre colchetes ([]). Como inicia em 0 (zero), pode-se acessar valores até o tamanho da string menos 1.
- Exemplos: string a

0	1	2	3	4	5	6	7	8	Índice
A	B	C	D	E	F	G	H	I	Conteúdo

```
>>> a = "ABCDEFGHI"
```

```
>>> print(a[0])
```

A

```
>>> print(a[1])
```

B

```
>>> print(a[8])
```

I

```
>>> print(a[9])
```

ERRO

```
>>> print(len(a))
```


Funções – Concatenação

- O conteúdo de variáveis string podem ser somados, ou melhor, concatenados, utilizando o operador de adição (+). Assim, “AB” + “C” é igual a “ABC”.

- Exemplos:

```
>>> s = “ABC”
```

```
>>> print(s + “C”)
```

```
ABCC
```

```
>>> print(s + “D” * 4)
```

```
ABCDDDD
```

```
>>> print(“X” + “-”*10 + “X”)
```

```
>>> print(s + “x4 = ” + s*4)
```

Funções – Composição

- Juntar várias *strings* para construir uma mensagem nem sempre é prático. Por exemplo, exibir que “João tem X anos”, onde X é uma variável numérica.
- Usando a composição, pode-se escrever:

```
print(“João tem %d anos” % X)
```
- O símbolo de % foi utilizado para indicar a composição da *string* anterior com o conteúdo da variável X.
- O %d dentro da primeira string é o marcador de posição de um valor inteiro. Principais marcadores:

Marcador	Tipo
%d	Números inteiros
%f	Números decimais
%s	Strings

Funções – Composição

- Exemplo com números decimais:

```
>>> print(“%f” % 5)
```

```
5.000000
```

```
>>> print(“%5.2f” % 5)
```

```
5.00
```

```
>>> print(“%10.5f” % 5)
```

```
5.00000
```

```
>>> print(“%s tem %d anos e apenas R$%5.2f no bolso” % (“João”, 22, 98.1))
```

```
João tem 22 anos e apenas R$ 98.10 no bolso
```

```
>>> nome = “João”
```

```
>>> idade = 22
```

```
>>> grana = 98.1
```

```
>>> print(“%s tem %d anos e apenas R$%5.2f no bolso” % (nome, idade, grana))
```

```
João tem 22 anos e apenas R$ 98.10 no bolso
```

Estruturas Sequenciais

Atribuição:

- É a principal forma de se armazenar um dado em uma variável. Esse comando permite que se forneça um valor a uma variável. É definido por:

identificador = expressão

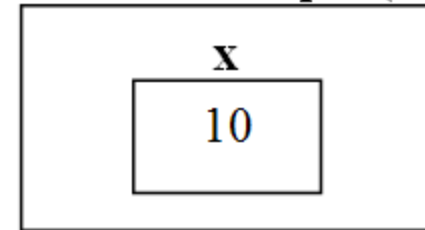
Onde...

Estruturas Sequenciais

Atribuição - Exemplos:

- Exemplo: **$x = 10$**

Memória Principal (MP)



- Como se lê?**

A variável x recebe o valor 10 ou x recebe 10.

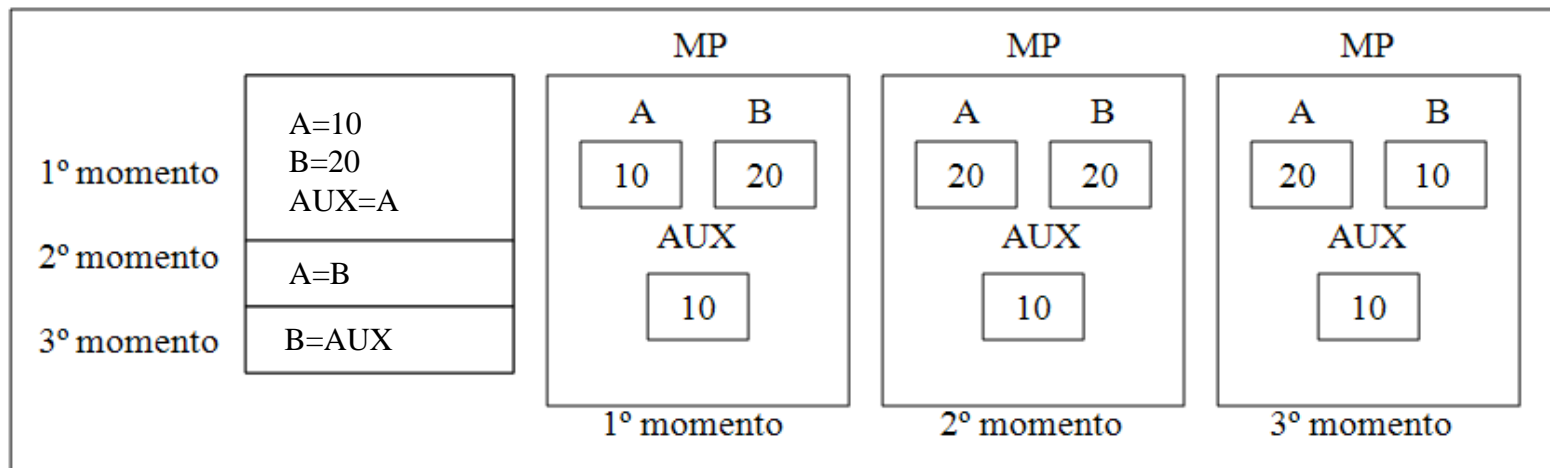
- O que faz o computador?**

Nesse momento, na memória do computador, essa variável recebe o valor 10.

Estruturas Sequenciais

Atribuição - Exemplos:

Exemplo:



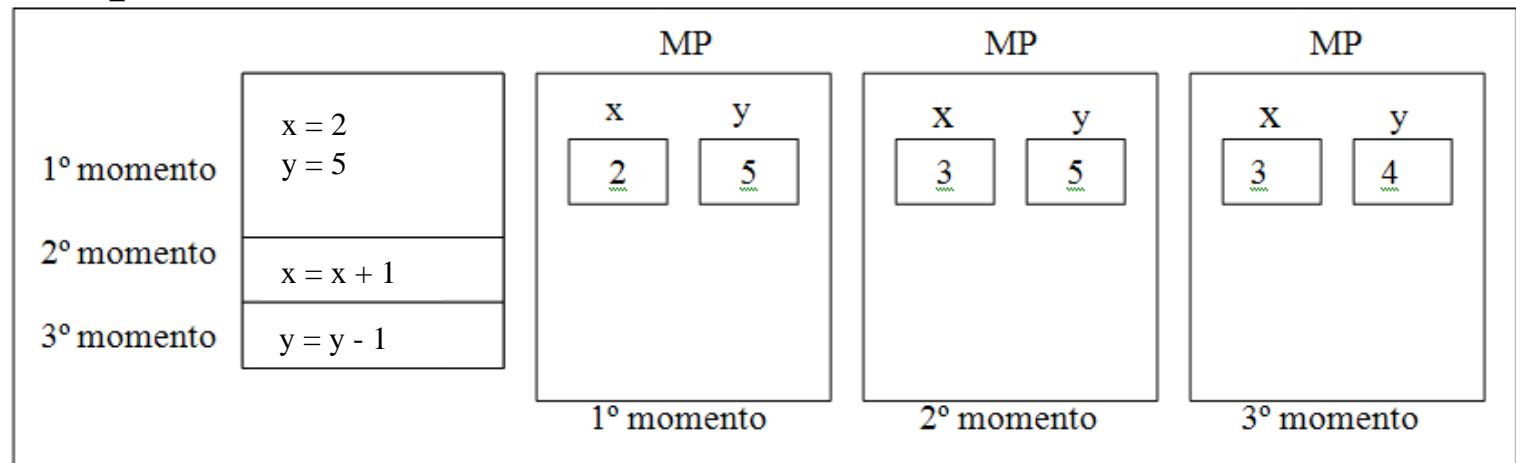
Qual o objetivo do algoritmo acima?

O conteúdo das variáveis A e B é trocado. No final a variável A está com o valor 20 e a variável B está com o valor 10. Notem a necessidade da variável auxiliar (AUX).

Estruturas Sequenciais

Atribuição - Exemplos:

Exemplo:



$x += 1$ equivalente a: $x = x + 1$

$y -= 1$ equivalente a: $y = y - 1$

Estruturas Sequenciais

Comando de Saída:

- É o comando responsável por enviar um resultado, uma informação ao usuário.
- Por meio deste comando o computador pode emitir os resultados e outras mensagens para o usuário por meio da tela do computador. É definido por:

print(expressão ou variável ou constantes)

O valor de cada variável é buscado na memória e inserido em um dispositivo de saída.

Estruturas Sequenciais

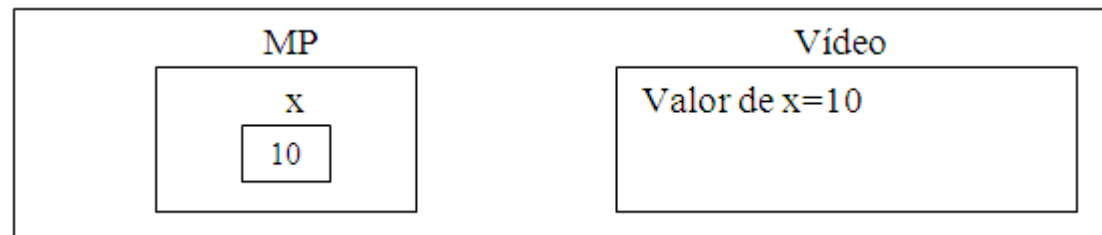
Saída - Exemplos:

x = 10

print("Valor de x=", x)

#Python

Saída:



Esse trecho permite a exibição de uma mensagem e do conteúdo de uma variável na tela do computador.

Estruturas Sequenciais

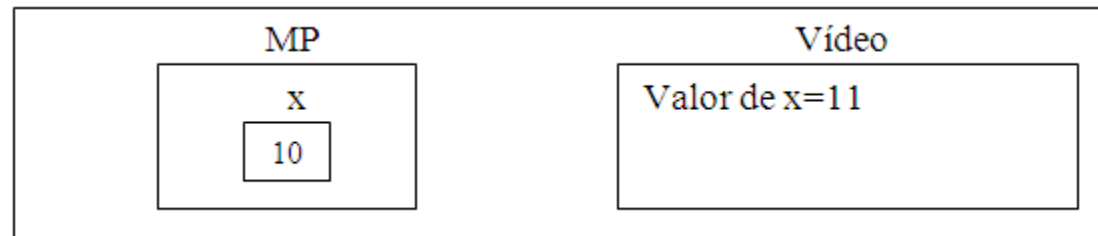
Saída - Exemplos:

x = 10

print("Valor de x=", x+1)

#Python

Saída:



Esse trecho é bem parecido com o anterior. O conteúdo da variável **x** é copiado da memória e acrescido de um, sendo impresso após a string, **sem alterar o valor de x na MP.**

Estruturas Sequenciais

Entrada de Dados:

- Permite que o usuário digite dados, possibilitando um “diálogo com o computador”, ou seja, permite que novos valores sejam fornecidos durante a execução, sem alterar os programas em si.
- O dado digitado é armazenado em uma variável.
- A função **input** é utilizada para solicitar dados do usuário. Possui um parâmetro que é a mensagem a ser exibida, e retorna o valor digitado.

Estruturas Sequenciais

Entrada - Exemplos:

Exemplo 1:

```
x = input("Digite um nome: ")    #Python
```

O que faz o computador?

O computador fica “esperando” o usuário digitar um dado; neste exemplo, um nome. A variável x, recebe o valor que o usuário digitar. Para facilitar, o dado digitado é sempre mostrado na tela.

Estruturas Sequenciais

Entrada - Exemplos:

Exemplo:

```
nome = input("Digite um nome: ")  
print("Olá, %s!" % nome)
```

O que faz o computador?

O computador fica “esperando” o usuário digitar um dado; neste exemplo, um nome. A variável nome recebe o valor que o usuário digitar. Então é impresso o nome na tela.

Estruturas Sequenciais

Entrada - Exemplos:

Exemplo:

```
x = input("Digite um número: ")  
print(x)
```

O que faz o computador?

O computador fica “esperando” o usuário digitar um dado; neste exemplo, um número. A variável x, recebe o valor que o usuário digitar. E então imprime-se o valor da variável x na tela.

Estruturas Sequenciais

Conversão da entrada de dados

A função **input** (em Python) sempre retorna valores do tipo *string*, ou seja, não importa se digitar somente números, o resultado será sempre uma *string*.

Para resolver este problema, deve-se utilizar a função *int* para converter o valor retornado em um número inteiro, e a função *float* para convertê-lo em um número decimal ou de ponto flutuante.

Estruturas Sequenciais

Conversão da Entrada

Exemplo:

```
anos = int(input("Anos de serviço: "))  
valor_por_ano = float(input("Valor do bônus por ano: "))  
bônus = anos * valor_por_ano  
print("Bônus de R$ %5.2f" % bônus)
```


Estruturas de Seleção

Estrutura de Seleção SIMPLES:

Sintaxe:

```
if condição:  
    comando ou  
    bloco verdadeiro
```

- Se a condição for verdadeira (True) o comando ou o bloco deslocados do início da linha para a direita são executados.
- O bloco verdadeiro continua até a primeira linha com deslocamento diferente.

ESTRUTURAS DE SELEÇÃO

Exemplo:

```
idade = int(input("Digite a idade do seu carro: "))
```

```
if idade <= 3:
```

```
    print("Seu carro é seminovo")
```

```
if idade > 3:
```

```
    print("Seu carro é usado")
```

Estruturas de Seleção

Estrutura de Seleção COMPOSTA:

Sintaxe:

```
if condição:  
    Comando ou  
    Bloco Verdadeiro  
else:  
    Comando ou  
    Bloco Falso
```

Se a condição for verdadeira (True) o comando ou o bloco verdadeiro serão executados, senão o comando ou o bloco falso serão executados.

ESTRUTURAS DE SELEÇÃO

Exemplo: Na mensagem do carro usado, a condição é simplesmente o inverso da primeira.

```
idade = int(input("Digite a idade do seu carro: "))  
if idade <= 3:  
    print("Seu carro é seminovo")  
else:  
    print("Seu carro é usado")
```

Estruturas de Seleção

Estruturas aninhadas (Encaixadas):

- Às vezes é necessário ter um outro teste de condição dentro da estrutura if.

```
if condição:  
    Comando ou  
    Bloco Verdadeiro  
else:  
    if condição:  
        Comando ou  
        Bloco Verdadeiro  
    else:  
        if condição:  
            Comando ou  
            Bloco Verdadeiro  
        else:  
            Comando ou  
            Bloco Falso
```

ESTRUTURAS DE SELEÇÃO

Exemplo: Cinco categorias são necessárias.

Fazer um programa que leia a categoria de um produto e determine o preço a partir da tabela:

Categoria	Preço
1	10,00
2	18,00
3	23,00
4	26,00
5	31,00

Exemplo em Python

```
categoria = int(input("Digite a categoria do produto: "))
if categoria == 1:
    preço = 10
else:
    if categoria == 2:
        preço = 18
    else:
        if categoria == 3:
            preço = 23
        else:
            if categoria == 4:
                preço = 26
            else:
                if categoria == 5:
                    preço = 31
                else:
                    print("Categoria inválida!")
                    preço = 0
print("O preço do produto é R$%6.2f" % preço)
```

ESTRUTURAS DE SELEÇÃO

Exemplo 5: Cinco categorias são necessárias. Fazer um programa que leia a categoria de um produto e determine o preço a partir da tabela:

Categoria	Preço
1	10,00
2	18,00
3	23,00
4	26,00
5	31,00

Agora substituindo um par **else if** por **elif**, mas sem criar outro nível de estrutura (somente para Python).


```
categoria = int(input("Digite a categoria do produto: "))
if categoria == 1:
    preço = 10
elif categoria == 2:
    preço = 18
elif categoria == 3:
    preço = 23
elif categoria == 4:
    preço = 26
elif categoria == 5:
    preço = 31
else:
    print("Categoria inválida!")
    preço = 0
print("O preço do produto é R$%6.2f" % preço)
```

Estrutura de Repetição *while* (enquanto)

Imprimindo de 1 a 3 com while:

```
#Python  
x = 1  
while x <= 3:  
    print(x)  
    x = x+1
```

Estrutura de Repetição *while*

Contadores

- São variáveis que atuam contando os valores a cada vez que o código é executado.
- Exemplo da impressão de 1 até um número digitado pelo usuário:

Exemplo:

```
fim=int(input("Digite o último número a imprimir:"))
```

```
x = 1
```

```
while x <= fim:
```

```
    print(x)
```

```
    x = x + 1
```

Estrutura de Repetição *while*

Contadores

- Exemplo da impressão de números pares de 0 até um número digitado pelo usuário:

Exemplo:

```
fim = int(input("Digite o último número a imprimir:"))  
x = 0  
while x <= fim:  
    if x % 2 is 0:  
        print(x)  
    x = x + 1
```

Estrutura de Repetição *while*

Acumuladores

- São variáveis que atuam acumulando os valores a cada vez que o código é executado.
- Utilizado em programas para calcular o total de uma soma, por exemplo.
- A diferença entre um contador e um acumulador é que nos contadores o valor adicionado é constante e, nos acumuladores, variável.

Estrutura de Repetição *while*

Acumuladores

- Exemplo de um programa que calcule a soma de 5 números digitados pelo usuário.

```
n = 1          # contador
```

```
soma = 0       # acumulador
```

```
while n <= 5:
```

```
    x = int(input("Digite o %d número: " % n))
```

```
    soma = soma + x
```

```
    n += 1
```

```
print("Soma:", soma)
```

Estrutura de Repetição *while*

Interrompendo a repetição

- Dependendo do problema, a habilidade de terminar o *while* dentro do bloco a repetir pode ser necessário.
- A instrução ***break*** é utilizada para interromper a execução de *while* independentemente do valor atual de sua condição.

Estrutura de Repetição *while*

Interrompendo a repetição

- Exemplo para somar vários números até que o usuário digite 0 (zero) para parar a leitura.

Exemplo:

```
soma = 0
```

```
while True:
```

```
    v=int(input("Digite um número a somar ou 0 para sair:"))
```

```
    if v is 0:
```

```
        break
```

```
    soma += v
```

```
print("Soma:", soma)
```


Estrutura de Repetição *while*

Repetições aninhadas

- Dependendo do problema, pode ser necessário combinar vários *while* com incremento de duas variáveis.
- Considere, por exemplo, imprimir as tabuadas de multiplicação de 1 a 10.

Estrutura de Repetição *while*

Repetições aninhadas

- Exemplo, imprimir as tabuadas de multiplicação de 1 a 10.

Exemplo:

```
tab = 1
```

```
while tab <= 10:
```

```
    núm = 1
```

```
    while núm <= 10:
```

```
        print("%d x %d = %d" % (tab,núm,tab*núm))
```

```
        núm += 1
```

```
    tab += 1
```

Lista

Uma lista vazia:

$$L = []$$

Os colchetes ([]) após o símbolo de igualdade servem para indicar que L é uma lista.

Uma lista com três elementos:

$$z = [15,8,9]$$

Lista

Acesso a uma lista:

```
>>> z = [15,8,9]
```

```
>>> z[0]
```

```
15
```

```
>>> z[1]
```

```
8
```

```
>>> z[2]
```

```
9
```

Lista

Exemplo, cálculo da média:

```
notas = [6,7,5,8,9]
```

```
soma = 0
```

```
i = 0
```

```
while i < 5:
```

```
    soma += notas[i]
```

```
    i += 1
```

```
print("Média: %5.2f" % (soma/i))
```

Lista – Cópia e fatiamento

Exemplo, tentativa de copiar listas:

```
>>> L = [1,2,3,4,5]
```

```
>>> V = L
```

```
>>> L
```

```
[1,2,3,4,5]
```

```
>>> V
```

```
[1,2,3,4,5]
```

```
>>> V[0] = 6
```

```
>>> V
```

```
[6,2,3,4,5]
```

```
>>> L
```

```
[6,2,3,4,5]
```

Em Python lista é um objeto e, quando atribui-se um objeto a outro, copia-se a mesma referência da lista, e não seus dados em si.

Lista – Cópia e fatiamento

Exemplo para criar uma cópia independente:

```
>>> L = [1,2,3,4,5]
```

```
>>> V = L[:]
```

```
>>> V[0] = 6
```

```
>>> V
```

```
[6,2,3,4,5]
```

```
>>> L
```

```
[1,2,3,4,5]
```

Ao escrever `L[:]`, refere-se a uma nova cópia de `L`. Assim, `L` e `V` se referem a áreas diferentes na memória, permitindo alterá-las de forma independente.

Lista – Cópia e fatiamento

Exemplos de fatiamento em Python:

```
>>> L = [1,2,3,4,5]
>>> L[0:5]          # elementos com índices 0 < 5
[1,2,3,4,5]
>>> L[:5]           # elementos com índice < 5
[1,2,3,4,5]
>>> L[:-1]          # menos o último elemento
[1,2,3,4]
>>> L[1:4]          # elementos com índice 1 até < 4
[2,3,4]
>>> L[3:]            # elementos a partir do índice 3
[4,5]
>>> L[:3]           # elementos com índices < 3
[1,2,3]
>>> L[-2]           # penúltimo elemento
```


Lista – Tamanho

Exemplo de repetição com tamanho de lista usando *len*

```
L = [1,2,3]
```

```
x = 0
```

```
while x < len(L):
```

```
    print(L[x])
```

```
    x += 1
```

A vantagem é que se trocar L para L=[7,8,9,10,11,12,13] o resto do programa continuaria funcionando, pois usou-se a função *len* para calcular e retornar o tamanho da lista.

Lista – Adição de elementos

- Uma das principais vantagens de usar Listas é poder adicionar novos elementos durante a execução do programa. Para tal, utiliza-se o método *append* (método do objeto lista).
- Métodos são recursos de orientação a objetos, pode-se imaginar como se fossem funções do objeto. Quando invocado, ele já sabe a que objeto está se referindo, pois é informado à esquerda do ponto.
- Exemplo para adicionar um valor na lista L:
L.append(valor)

Lista – Adição de elementos

Exemplo de adição de elementos à lista em Python:

```
L = []
```

```
while True:
```

```
    n = int(input("Digite um número (0 para sair): "))
```

```
    if n is 0:
```

```
        break
```

```
    L.append(n)
```

```
x = 0
```

```
while x < len(L):
```

```
    print(L[x])
```

```
    x += 1
```

Lista – Remoção de elementos

Pode-se retirar alguns elementos da lista, ou mesmo todos eles. Para isso, utilizar-se-á em Python a instrução *del*.

```
>>> L = ["a", "b", "c"]
```

```
>>> del L[1]
```

```
>>> L
```

```
['a', 'c']
```

```
>>> del L[0]
```

```
>>> L
```

```
['c']
```

O elemento removido não ocupa mais lugar na lista, fazendo com que os índices sejam reorganizados sem esse elemento.

Lista – Remoção de elementos

Pode-se apagar fatias inteiras de uma só vez.

```
>>> L = list(range(101))  # *
```

```
>>> L
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 94, 95, 96, 97, 98, 99, 100]
```

```
>>> del L[1:99]
```

```
>>> L
```

```
[0, 99, 100]
```

range: gera uma lista de números. Usado em loops for.*

Lista – Usando *for*

- Estrutura de repetição projetada para percorrer listas. A instrução *for* funciona de forma parecida a *while*, mas a cada repetição utiliza um elemento diferente da lista.
- A cada repetição, o próximo elemento da lista é utilizado, o que se repete até o fim da lista.
- Exemplo da impressão de todos os elementos da lista:

```
L = [8,9,15]
```

```
for e in L:
```

```
    print(e)
```

```
L = [8,9,15]
```

```
x = 0
```

```
while x < len(L):
```

```
    e = L[x]
```

```
    print (e)
```

```
    x += 1
```

Lista – Usando *for*

Exemplo de pesquisa usando *for*:

```
L = [7,9,10,12]
```

```
p = int(input("Digite um valor a procurar: "))
```

```
for e in L:
```

```
    if e is p:
```

```
        print("%d foi encontrado" % p)
```

```
        break
```

```
else:
```

```
    print("%d não encontrado" % p)
```

Lista – Usando *range*

A função *range* é utilizada para gerar listas simples.

Exemplo para imprimir de 0 a 9 na tela:

```
for v in range(10):  
    print(v)
```

Com a mesma função pode-se indicar qual é o primeiro número a gerar. Para isso, deve-se utilizar dois parâmetros: início e fim. Exemplo:

```
for v in range(5,8):  
    print(v)
```


Lista – Usando *range*

Se adicionar um terceiro elemento, terá como saltar entre os valores gerados. Exemplo:

```
for t in range(3,33,3):  
    print(t)
```

Pode-se transformar o resultado de *range* em uma lista.
Exemplo:

```
L=list(range(100,1100,50))  
print(L)
```

Lista – Usando *range* – Exemplo

Programa para ler N notas e imprimir ao final:

```
N = int(input("Digite a quantidade de Notas: "))
alu = []
qtde = 0

for i in range(0,N,1):
    alu.append(float(input("Digite a %da. nota: " %
                           (qtde+1))))
    qtde += 1

print(alu)
```

Altere o programa para imprimir ao final a média aritméticas das notas digitadas.

Lista – Usando *enumerate*

Com a função *enumerate* pode-se ampliar as funcionalidades de *for*. Exemplo de como imprimir uma lista, onde se tem o índice entre colchetes e o valor à sua direita:

```
L = [5,9,13]
```

```
x = 0
```

```
for e in L:
```

```
    print("[%d] %d" % (x,e))
```

```
    x += 1
```

```
L = [5,9,13]
```

```
for x, e in enumerate(L):
```

```
    print("[%d] %d" % (x,e))
```

```
# o primeiro valor é o índice e o
```

```
# segundo é o elemento da lista
```

Funções – Criando

- Pode-se definir as próprias funções, permitindo que a solução seja reutilizada em outras partes do programa. Sabe-se como usar várias funções, como **len**, **int**, **float**, **print** e **input**.
- Para definir uma nova função, usa-se a instrução **def** (em Python).
- Exemplo de uma nova função:

```
def soma (a, b):    # soma é o nome da função  
    print(a + b)    # bloco da função com recuo à direita
```

```
soma(2,9)           # chamar a função soma, com a valendo 2 e b 9  
soma(7,8)  
soma(10,15)
```

Funções – Retorno de valor

- Pode-se retornar o valor de uma função utilizando a instrução **return**. Deixa-se, por exemplo, a função realizar os cálculos e na sequência retornar o resultado para impressão.
- Exemplo de uma nova função com retorno de um valor:

```
def soma (a, b):           # soma é o nome da função  
    return (a + b) # bloco da função com recuo à direita
```

```
print(soma(2, 9))         # imprime o valor retornado da função soma
```

- Pode-se dizer que **return** marca o fim da execução da função.

Funções – Retorno de valor

- Exemplo de retorno que verifica se valor é par ou não em Python:

```
def épar(x):  
    return (x%2 is 0)
```

```
print(épar(2))      # imprime se o valor 2 é par  
print(épar(3))  
print(épar(10))
```

E para imprimir Par ou Ímpar?

Funções – Retorno de valor

- Exemplo de reutilização da função épar em outra função:

```
def épar(x):  
    return (x%2 is 0)  
def par_ímpar(x):  
    if épar(x):  
        return “Par”  
    else:  
        return “Ímpar”  
print(par_ímpar(4))  
print(par_ímpar(5))
```

Funções – Pesquisa em uma lista

- Exemplo:

```
def pesquise(lista, valor):    #a função recebe dois parâmetros
    for x, e in enumerate(lista):
```

```
        if e is valor:
```

```
            return x                # retorna o valor da posição
```

```
    return None
```

```
#####
```

```
L = [10,20,25,30]
```

```
print(pesquise(L, 25))    # envia dois parâmetros para a função
```

```
print(pesquise(L, 27))
```


Funções – Cálculo da média

- Exemplo:

```
def média(L):
```

```
    total = 0
```

```
    for e in L:
```

```
        total += e
```

```
    return total/len(L)
```

```
#####
```

```
L = [10,20,25,30]
```

```
print(média(L))
```

Variáveis locais e globais

- Ao usar funções usa-se variáveis internas ou locais e variáveis externas ou globais. A diferença é a visibilidade ou escopo.
- Uma variável local a uma função existe apenas dentro dela, sendo normalmente inicializada a cada chamada. Assim, não se pode acessar o valor de uma variável local fora da função que a criou e, por isso, passam-se parâmetros e retornam-se valores nas funções, permitindo a troca de dados.
- Uma variável global é definida fora de uma função, podendo ser vista por todas as funções do módulo e por todos os módulos que importam o módulo que a definiu.

Variáveis locais e globais

- Exemplo:

```
EMPRESA="Universidade do Estado de SC"
```

```
def imprime_cabecalho():
```

```
    print(EMPRESA)
```

```
    print("-" * len(EMPRESA))
```

- A função `imprime_cabecalho` não recebe parâmetros nem retorna valores. Observe que a variável `EMPRESA` definida fora da função é uma variável global.

Variáveis locais e globais

- Execute o programa e analise seu resultado:

```
a = 5
```

```
def muda_e_imprime():
```

```
    a = 7
```

```
    print("A dentro da função:", a)
```

```
#####
```

```
print("A antes de mudar:", a)
```

```
muda_e_imprime()
```

```
print("A depois de mudar:", a)
```

Variáveis locais e globais

- Execute o programa que permite modificar uma variável global dentro de uma função, pela instrução **global**:

```
a = 5
```

```
def muda_e_imprime():
```

```
    global a
```

```
    a = 7
```

```
    print("A dentro da função:", a)
```

```
#####
```

```
print("A antes de mudar:", a)
```

```
muda_e_imprime()
```

```
print("A depois de mudar:", a)
```

Parâmetros Opcionais

- Nem sempre será necessário passar todos os parâmetros para uma função, preferindo utilizar um valor previamente escolhido como padrão.

- Exemplo 1:

```
def barra():  
    print("*" * 40)
```

- Exemplo 2:

```
def barra(n=40, caractere="*"):  
    print(caractere * n)
```

- Exemplo 3:

```
>>> barra(10)      # faz com que n seja 10  
*****
```

```
>>> barra(10, "-")
```

Parâmetros Opcionais – Exemplo

```
def soma(a, b, imprime=False)
```

```
    s = a + b
```

```
    if imprime:
```

```
        print(s)
```

```
    return s
```

Testar as seguintes instruções e escrever o resultado:

```
>>> soma(2,3)
```

```
>>> soma(3,4,True)
```

```
>>> soma(5,8,False)
```

Nomeando Parâmetros

- Ao especificar o nome dos parâmetros, pode-se passá-los em qualquer ordem.
- Exemplo com parâmetros obrigatórios e opcionais:

```
def retângulo(largura, altura, caractere="*"):
```

```
    linha = caractere*largura
```

```
    for i in range(altura):
```

```
        print(linha)
```

Testar as seguintes instruções e escrever o resultado:

```
>>> retângulo(3,4)
```

```
>>> retângulo(largura=3, altura=4)
```

```
>>> retângulo(altura=4, largura=3)
```

```
>>> retângulo(caractere="-", altura=4, largura=3)
```


Desempacotamento de Parâmetros

- Pode-se criar funções que recebem um número indeterminado de parâmetros utilizando listas de parâmetros.
- Exemplo:

```
def soma(*args):
```

```
    s = 0
```

```
    for x in args:
```

```
        s += x
```

```
    return s
```

```
#####
```

```
soma(1,2)
```

```
soma(2)
```

```
soma(5,6,7,8)
```

```
soma(9,10,20,30,40)
```

Nulo em Python = nenhum

Opcionalmente, pode-se atribuir nulo para uma variável, e, depois, atribuí-la a uma instância de objeto ou constante.

null in Javascript:

```
var null_variable = null;
```

null in PHP:

```
$null_variable = NULL;
```

null in Java:

```
SomeObject null_obj = null;
```

null em Python:

```
my_none_variable = None
```

Muitas vezes pode-se querer realizar uma ação que pode ou não funcionar. Usar *None* é uma maneira de verificar o estado da ação mais tarde. Exemplo:

```
database_connection = None
try:
    database = MyDatabase(db_host,
                          db_user, db_password,
                          db_database)
    database_connection =
        database.connect()
except DatabaseException:
    pass

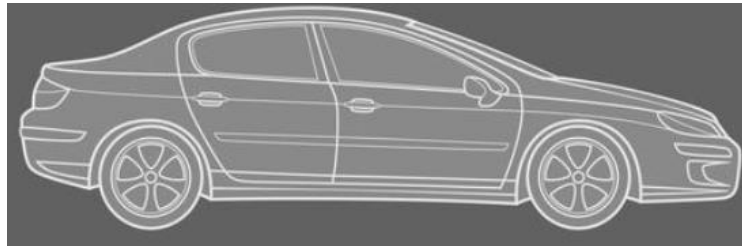
if database_connection is None:
    print('The database could not
        connect')
else:
    print('The database could
        connect')
```

Classe e Objeto – Conceito

- Classes são a definição de um novo tipo de dados que associa dados e operações em uma só estrutura (estrutura natural).
- Objeto é uma variável cujo tipo é uma classe, ou seja, um objeto é uma instância de uma classe. Como a representação de um objeto do mundo real, escrita em uma linguagem de programação.

Classe x Objeto

Classe:



Veículo

- modelo : string
- cor : string
- placa : string

Objetos:



Modelo: Gol

Cor: Verde

Placa: GOL 1983



Modelo: Fusca

Cor: Azul

Placa: KTU 1965

Método Construtor – Exemplo

Exemplo de um aparelho de TV. A TV tem uma marca e um tamanho de tela. Pode-se fazer com a TV, por exemplo, mudança de canal, ligá-la ou desligá-la.

class TV:

ligada = None *# Opcional, apenas facilita a identificação dos atributos*

canal = None *# globais em uma classe*

def __init__(self): *# método construtor; self é um objeto TV em si*

self.ligada = False *# é um valor de self, ou seja, do objeto TV*

self.canal = 2 *# ao especificar atributos do objeto, sempre usar self*

#####

tv = TV() *# cria-se um objeto tv utilizando a classe TV*

print(tv.ligada) *# ou seja, tv é uma instância de TV*

print(tv.canal)

tv_sala = TV() *# cria-se um objeto tv_sala*

tv_sala.ligada = **True**

tv_sala.canal = 4

print(tv.ligada)

print(tv.canal)

print(tv_sala.ligada)

print(tv_sala.canal)

Método Destruitor - `__del__`

- Destruítores são chamados automaticamente quando um objeto é destruído. É o oposto do construtor, que é chamado na criação.
- No Python, os destruidores não são necessários, porque o Python tem um coletor de lixo que manipula o gerenciamento de memória automaticamente.
- O método `__del__` () é conhecido como um método destrutivo no Python. É chamado quando todas as referências ao objeto foram excluídas, ou seja, quando um objeto é coletado como lixo.
- Exemplo 1:

```
class TestClass:  
    def __init__(self):  
        print ("construtor")  
    def __del__(self):  
        print ("destrutor")
```

```
obj = TestClass()  
del obj
```


Adicionando comportamento à classe

```
class TV:
    ligada = None
    canal = None
    def __init__(self):
        self.ligada = False
        self.canal = 2
    def muda_canal_para_baixo(self):
        self.canal -= 1
    def muda_canal_para_cima(self):
        self.canal += 1

#####
tv = TV()
tv.muda_canal_para_cima()
tv.muda_canal_para_cima()
print(tv.canal)
tv.muda_canal_para_baixo()
print(tv.canal)
```

Passagem de parâmetros

```
class TV:
```

```
    ligada = None
```

```
    canal = None
```

```
    cmin = None
```

```
    cmax = None
```

```
    def __init__(self, min, max):
```

```
        self.ligada = False
```

```
        self.canal = 2
```

```
        self.cmin = min
```

```
        self.cmax = max
```

```
    def muda_canal_para_baixo(self):
```

```
        if(self.canal-1 >= self.cmin):
```

```
            self.canal -= 1
```

```
    def muda_canal_para_cima(self):
```

```
        if (self.canal + 1 <= self.cmax):
```

```
            self.canal += 1
```

```
#####
```

```
tv = TV(1,99)
```

```
for x in range(0,120):
```

```
    tv.muda_canal_para_cima()
```

```
print(tv.canal)
```

```
for x in range(0,120):
```

```
    tv.muda_canal_para_baixo()
```

```
print(tv.canal)
```


Classe e Objeto – Self

- Tudo o que foi visto sobre funções, aplica-se para método. Porém, o método está associado a uma classe e atua sobre um objeto.
- O primeiro parâmetro do método é chamado **self**, e representa a instância sobre a qual o método atuará. É por meio de **self** que se tem acesso aos outros métodos de uma classe, preservando todos os atributos dos objetos.
- Não precisa passar o objeto como primeiro parâmetro ao invocar um método. Python faz automaticamente, porém faz-se necessário declarar **self** como o primeiro parâmetro de seus métodos.

Encapsulamento

- Métodos e atributos **fracamente privados** possuem um sublinhado (__) no início (à esquerda). Essa abordagem APENAS sinaliza que essas variáveis são privadas e outros programadores não devem usá-las em código externos, exceto se tratar de uma subclasse (herança). Porém, aos métodos fracamente privados não se impede que o código seja acessado de fora da classe. O efeito é que quando os módulos são importados as variáveis com sublinhado (__) não serão importadas.
- Exemplo de itens fracamente privados:

```
class ClasseFracamentePrivada():
    __atributoFracamentePrivado = None
    def __init__(self):
        self.__atributoFracamentePrivado = 100
    def __MetodoFracamentePrivado(self):
        print("método fracamente privado")
#####
p1 = ClasseFracamentePrivada()
p1.__MetodoFracamentePrivado()
print(p1.__atributoFracamentePrivado)
```

Encapsulamento

- Métodos e atributos **fortemente privados** possuem dois sublinhados (__) no início (à esquerda). Essa abordagem faz com que o nome do método não possa ser encontrado/acessado fora da classe - os escondendo. Para acessar um método fortemente privado basta colocar um sublinhado (_) na frente do nome classe em seguida escrever o nome da variável que o método poderá ser acessado.
- Exemplo de itens fortemente privados:

```
class ClasseFortementePrivada():
    __atributoFortementePrivado = None
    def __init__(self):
        self.__atributoFortementePrivado = 100
    def __MetodoFortementePrivado(self):
        print("método fortemente privado")
#####
p1 = ClasseFortementePrivada()
p1._ClasseFortementePrivada__MetodoFortementePrivado()
print(p1._ClasseFortementePrivada__atributoFortementePrivado)
```

Acessores e Modificadores

- Um atributo de classe promove o método de acesso e/ou método modificador. O método *get* é uma propriedade específica para acessar os atributos chamada de "*getter*". Já o método *set* é uma propriedade específica do método modificador chamado de "*setter*".
- Por convenção, adiciona-se a palavra *get* (obter, pegar) e *set* (por, colocar) antes do nome do atributo. Os métodos *get* e *set* promovem o acesso dos atributos. Um método que acessa uma instância mas não modifica a instância é chamado de acessor.
- Os métodos *set* promovem a modificação dos atributos. Um método que modifica uma instância é um modificador.
- O operador . (ponto) é utilizado para especificar o objeto em que o método também é invocado.

Acessores e Modificadores – Exemplo

```
class Cliente:
```

```
    __nome = None
```

```
    __telefone = None
```

```
    def __init__ (self, nome, telefone):
```

```
        self.__nome = nome
```

```
        self.__telefone = telefone
```

```
    def getnome (self):
```

```
        return self.__nome
```

```
    def setnome (self, nome):
```

```
        self.__nome = nome
```

```
    def gettelefone (self):
```

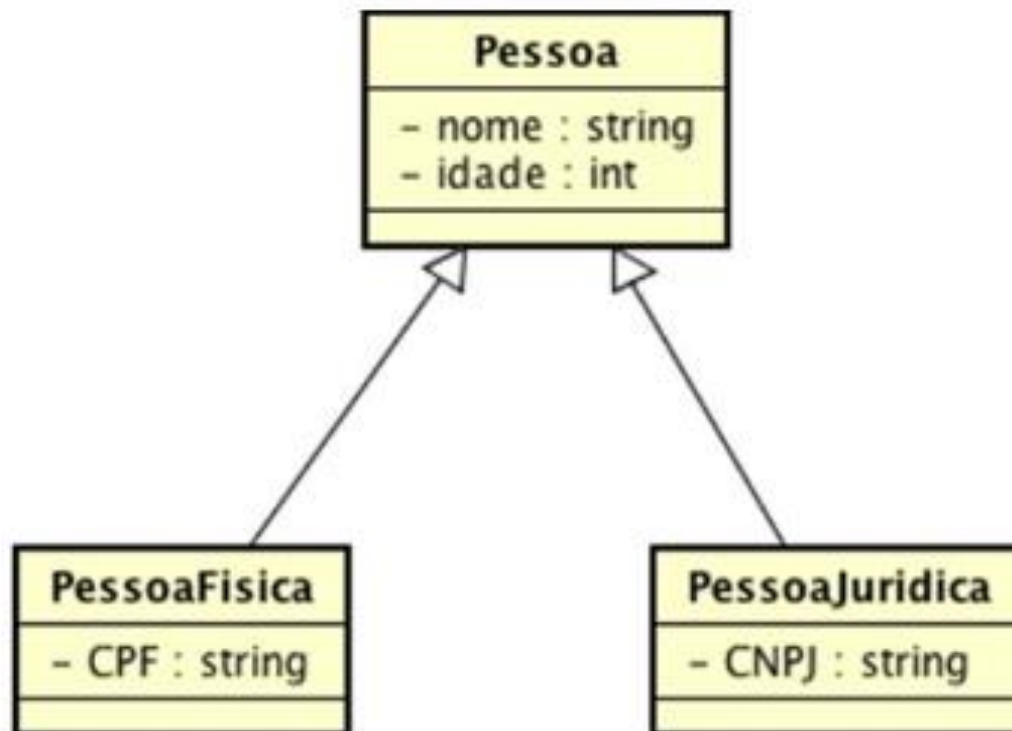
```
        return self.__telefone
```

```
    def settelefone (self, telefone):
```

```
        self.__telefone = telefone
```

Herança

- O mesmo ocorre entre Pessoa Física e Pessoa Jurídica, ou seja, possuem atributos semelhantes que podem ser estendidos para uma classe Pessoa.
- Exemplo:



class Pessoa:

__nome = None

__idade = None

def __init__(self, nome, idade):

self.__nome = nome

self.__idade = idade

def setnome(self, nome):

self.__nome = nome

def setidade(self, idade):

self.__idade = idade

def getnome(self):

return self.__nome

def getidade(self):

return self.__idade

Exercício:

- **instancie dois objetos (pf e pj);**
- **imprima os valores dos atributos dos objetos.**

from pessoa import Pessoa

class PessoaFisica(Pessoa):

__CPF = None

def __init__(self, CPF, nome, idade):

super().__init__(nome, idade)

self.__CPF = CPF

def getCPF(self):

return self.__CPF

def setCPF(self, CPF):

self.__CPF = CPF

from pessoa import Pessoa

class PessoaJuridica(Pessoa):

__CNPJ = None

def __init__(self, CNPJ, nome, idade):

super().__init__(nome, idade)

self.__CNPJ = CNPJ

def getCNPJ(self):

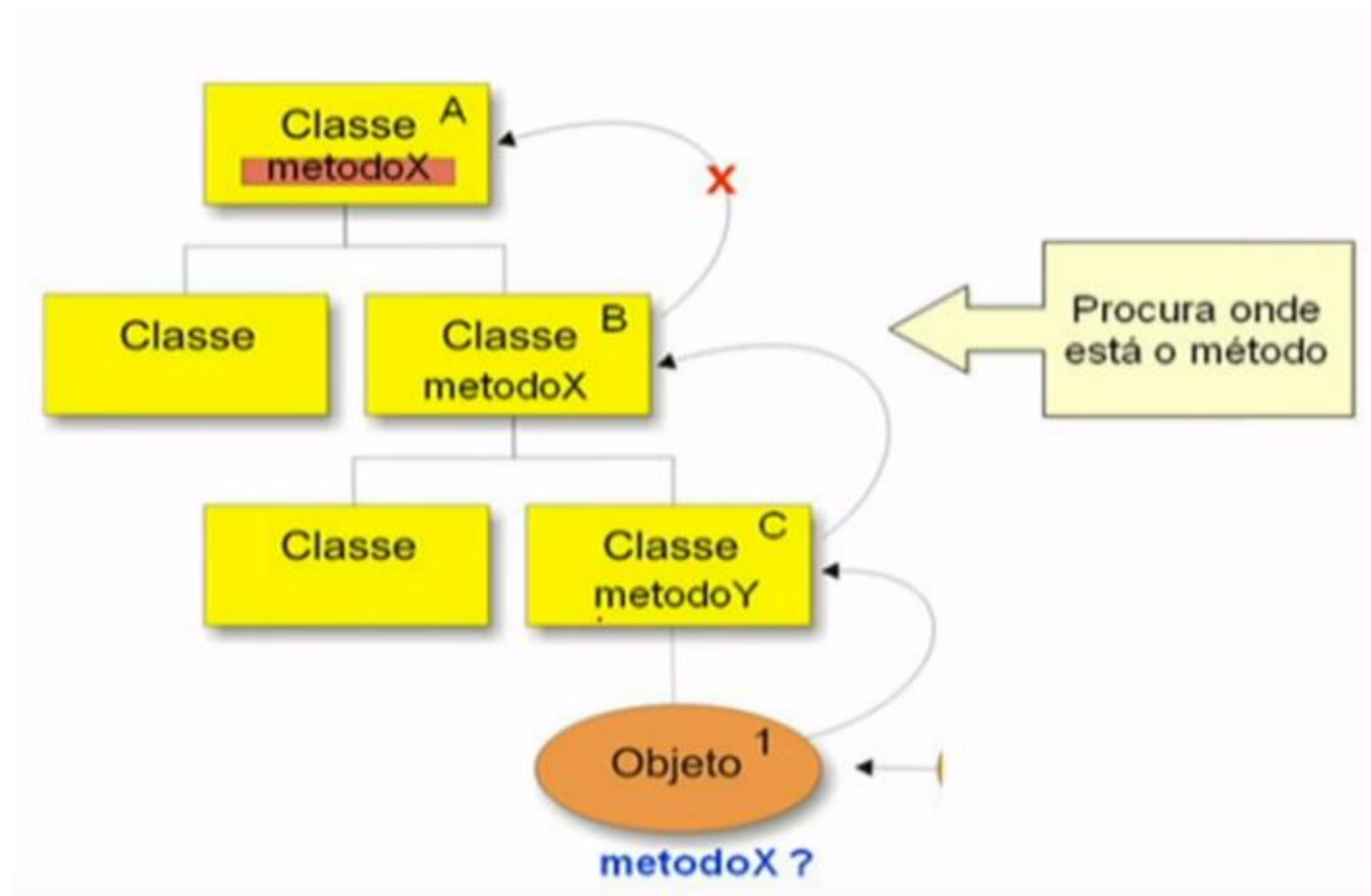
return self.__CNPJ

def setCNPJ(self, CNPJ):


self.__CNPJ = CNPJ

Herança e Sobrecarga de Método (Polimorfismo)

Qual método X é executado?



Sobrecarga de Método (Polimorfismo) – Exemplo 1



O que será
impresso
na tela?


```
class Brinquedo:  
    def mover(self):  
        print('Mover brinquedo')
```

```
class Carro (Brinquedo):  
    def mover(self):  
        print('Correr')
```

```
class Aviao (Brinquedo):  
    def mover(self):  
        print('Voar')
```

```
a1 = Carro()  
a1.mover()
```

Sobrecarga de Método (Polimorfismo) – Exemplo 2



O que será
impresso
na tela?


```
class Brinquedo:  
    def mover(self):  
        print('Mover brinquedo')
```

```
class Carro (Brinquedo):  
    def mover(self):  
        print('Correr')
```

```
class Aviao (Brinquedo):  
    def mover(self):  
        print('Voar')
```

```
a1 = []  
a1.append(Carro())  
a1.append(Aviao())  
a1.append(Carro())  
for x in a1:  
    x.mover()
```

Sobrecarga de Método (Polimorfismo) – Exemplo 3



O que será
impresso
na tela?

```
class Brinquedo:  
    def mover(self):  
        print('Mover brinquedo')
```

```
class Carro (Brinquedo):  
    def Mover(self):  
        print('Correr')
```

```
class Aviao (Brinquedo):  
    def mover(self):  
        print('Voar')
```

```
a1 = []  
a1.append(Carro())  
a1.append(Aviao())  
a1.append(Carro())  
for x in a1:  
    x.mover()
```

Sobrecarga de Método – Exemplo

Considere a classe Pessoa e a classe Pais. Como Pais também são Pessoas, a classe Pais é derivada da classe Pessoa.

```
class Pessoa:
```

```
    _nome = None
```

```
    _sexo = None
```

```
    def __init__(self, nome, sexo):
```

```
        self._nome = nome
```

```
        self._sexo = sexo
```

```
    def __str__(self):
```

```
        return str(self._nome)
```

```
class Pais(Pessoa):
```

```
    _crianca = None
```

```
    def __init__(self, nome, sexo, crianca):
```

```
        super().__init__(nome, sexo)
```

```
        self._crianca = crianca
```

```
    def getCrianca(self):
```

```
        return self._crianca
```

```
    def __str__(self):
```

```
        return self._crianca
```

EXEMPLO:

Classe Pessoa: Crie uma classe que modele uma pessoa:

Atributos: nome, idade, peso e altura

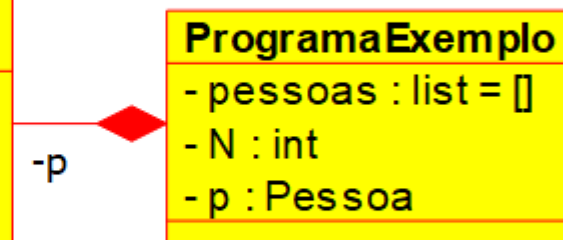
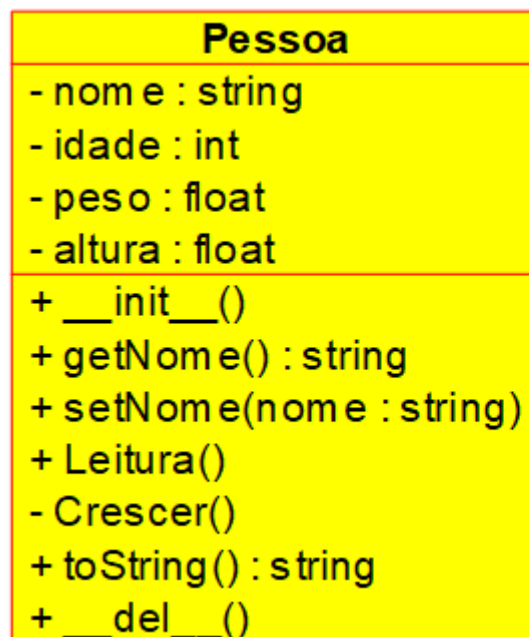
Métodos: Envelhecer, engordar, crescer, emagrecer. Obs: Por padrão, a cada ano que a pessoa envelhece, sendo a idade dela menor que 21 anos, ela deve crescer 0,5 cm ao ano. A pessoa engorda 0,5 kg por ano.

Faça uma projeção para X anos, informando o novo peso e/ou altura.

class Pessoa:

```
__nome = None
__idade = None
__peso = None
__altura = None

def __init__(self):
    self.__nome = None
    self.__idade = 0
    self.__peso = 0.0
    self.__altura = 0.0
def getNome(self):
    return self.__nome
def setNome(self, nome):
    self.__nome = nome
```



```
def getIdade(self):  
    return self.__idade  
def setIdade(self, idade):  
    self.__idade = idade  
def getPeso(self):  
    return self.__peso  
def setPeso(self, peso):  
    self.__peso = peso  
def getAltura(self):  
    return self.__altura  
def setAltura(self, altura):  
    self.__altura = altura
```

```
def Leitura(self):  
    self.__nome = input('\nEntre com o nome: ')  
    self.__idade = int(input('Idade: '))  
    self.__altura = float(input('Altura (em cm): '))  
    self.__peso = float(input('Peso: '))
```

```
def __Crescer(self):  
    self.__altura = self.__altura + 0.5
```

```
def Envelhecer(self, proj):  
    id = self.__idade  
    while id < 21 and id < proj:  
        self.__Crescer()  
        id += 1
```

```
def Engordar(self):  
    self.__peso += 0.5  
    self.__idade += 1
```

```
def toString(self):  
    Str=""  
    Str = Str + "\nNome: %s\nPeso: %.2f\nAltura: %.2f" % (self.__nome,  
                                                            self.__peso, self.__altura)  
    return Str
```

```
def __del__(self):  
    self.__nome = None  
    self.__idade = 0  
    self.__peso = 0.0  
    self.__altura = 0.0
```



```
#####
```

```
N=int(input('Entre com a quantidade de Pessoas: '))
```

```
    pessoas = []
```

```
    for i in range(0, N, 1):
```

```
        p = Pessoa()
```

```
        p.Leitura()
```

```
        proj = int(input('Deseja estimar a altura e peso para qual idade? '))
```

```
        p.Envelhecer(proj)
```

```
        while p.getIdade() <= proj:
```

```
            p.Engordar()
```

```
        pessoas.append(p)
```

```
for p in pessoas:
```

```
    print(p.toString())
```

```
for p in pessoas:
```

```
    del p
```


Exercícios

Para os exercícios a seguir, incluir no mínimo:

- Construtores e destrutores;
- *Getters* e *setters*;
- Método para leitura;
- Método *toString* para impressão (quando necessário);
- Encapsulamento;
- Herança (quando necessário);
- Diagrama de classe.

Exercícios Revisão

1. Crie a classe Imovel, que possui um endereço e um preço.

- a) crie uma classe Novo, que herda Imovel e possui um adicional no preço. Crie métodos de acesso e impressão deste valor adicional.
- b) crie uma classe Velho, que herda Imovel e possui um desconto no preço. Crie métodos de acesso e impressão para este desconto.

Exercícios Revisão

2. Fazer um programa que mostre o seguinte menu:

Menu de Opções:

- 1) Ler dois valores inteiros
- 2) Somar
- 3) Subtrair
- 4) Dividir
- 5) Multiplicar
- 6) Sair

Digite a opção:

A única forma de encerrar a execução dar-se-á mediante a opção 6.

Exercícios Revisão

3. Fazer um programa que proporcione ao usuário um menu para as seguintes operações:

- Leia o valor de n e os n valores de uma lista A com valores numéricos;
- Imprime os valores ordenados de forma crescente;
- Determine e imprima para cada número que se repete no conjunto a quantidade de vezes que ele aparece repetido;
- Elimine os elementos repetidos formando uma nova lista B , imprimindo este novo conjunto;
- Determine e imprima a média dos valores da lista A .

Obs: impede-se o uso de *sort* e *reverse*.

Exercícios Revisão

4. Fazer um programa que leia uma série de *strings* (utilize *flag* para encerrar a leitura), em seguida exibir um menu com a opção de imprimir somente as *strings* que iniciarem com uma determinada letra (definida pelo usuário em tempo de execução), e a outra opção para imprimir todas as *strings* em ordem crescente. Bônus: adicione uma opção para imprimir todas as *strings* em ordem decrescente.

Obs: impede-se o uso de *sort* e *reverse*.

Exercícios Revisão

5. Fazer um programa que:

- a) leia o número de inscrição, altura e peso das moças inscritas em um concurso de beleza (utilize *flag* adequada para encerrar a leitura);
- b) calcule e imprima as moças aprovadas para o concurso, ou seja, as moças com IMC inferior a 18.

IMC = índice de massa corporal (peso / altura²).

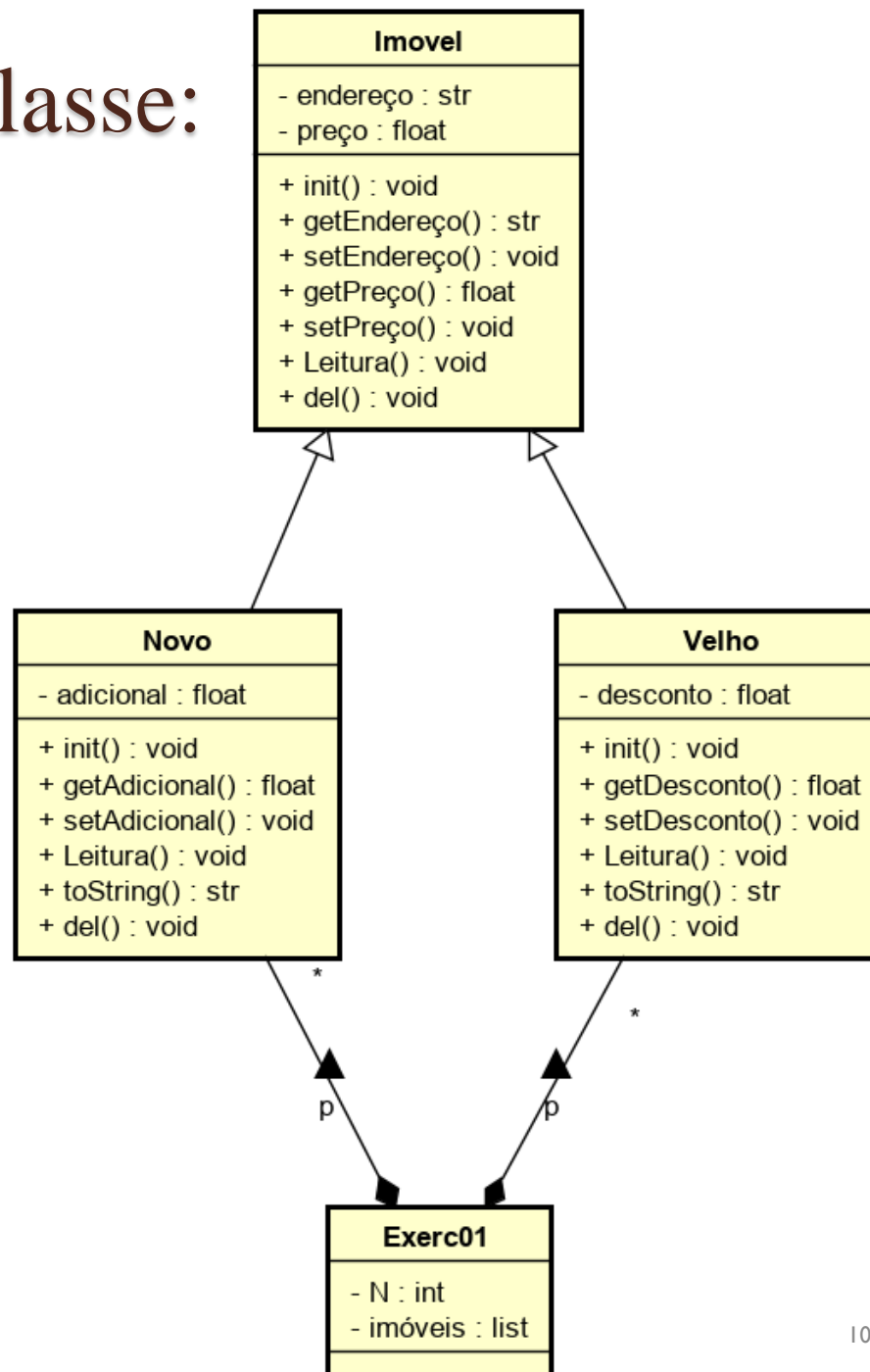
Exercícios Revisão

6. Fazer um programa para uma universidade. Os atributos são: número de matrícula, sexo, nota1 e nota2. Fazer um programa que:
- a) Leia as informações de cada aluno (utilizar *flag* para encerrar a leitura);
 - b) Imprima a relação dos alunos aprovados (média ≥ 7);
 - c) Imprima a relação dos alunos em exame;
 - d) Determine e imprima o melhor aluno (pode considerar que não haverá empate), e o aluno com menor média.

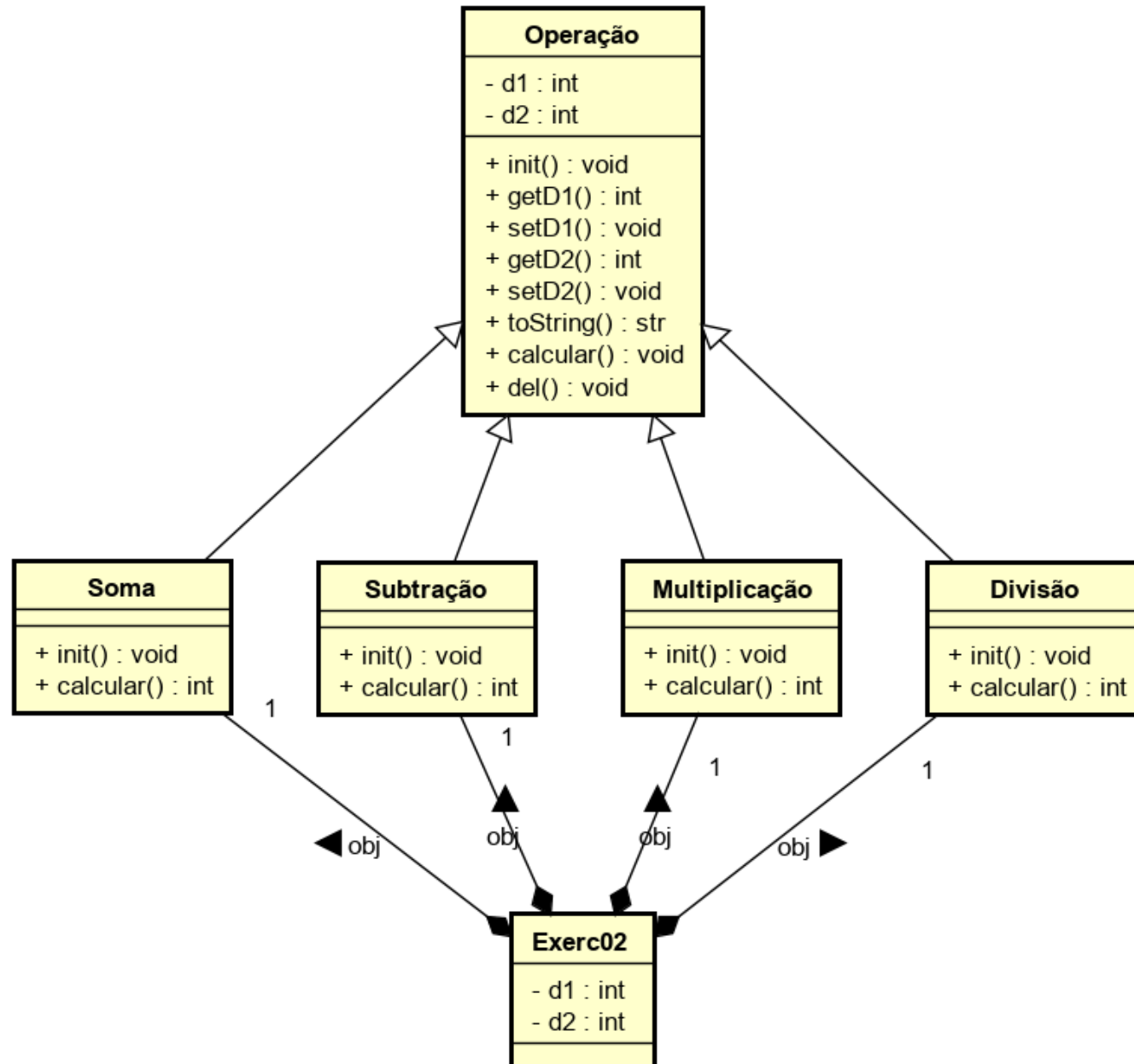
Exercícios Revisão – Bônus

7. Fazer um programa para corrigir provas de múltipla escolha. Cada prova tem 5 questões e cada questão vale 2 pontos. O primeiro conjunto de dados a ser lido será o gabarito para a correção da prova. Os outros dados serão, os números de matrícula dos alunos e suas respectivas respostas (utiliza *flag* para encerrar a leitura). O algoritmo deverá calcular e imprimir:
- a) para cada aluno, o seu número e a sua nota;
 - b) a porcentagem de aprovação, sabendo-se que a nota mínima de aprovação é 7;
 - c) a nota que teve maior frequência absoluta, ou seja, a nota que apareceu mais vezes entre os alunos.

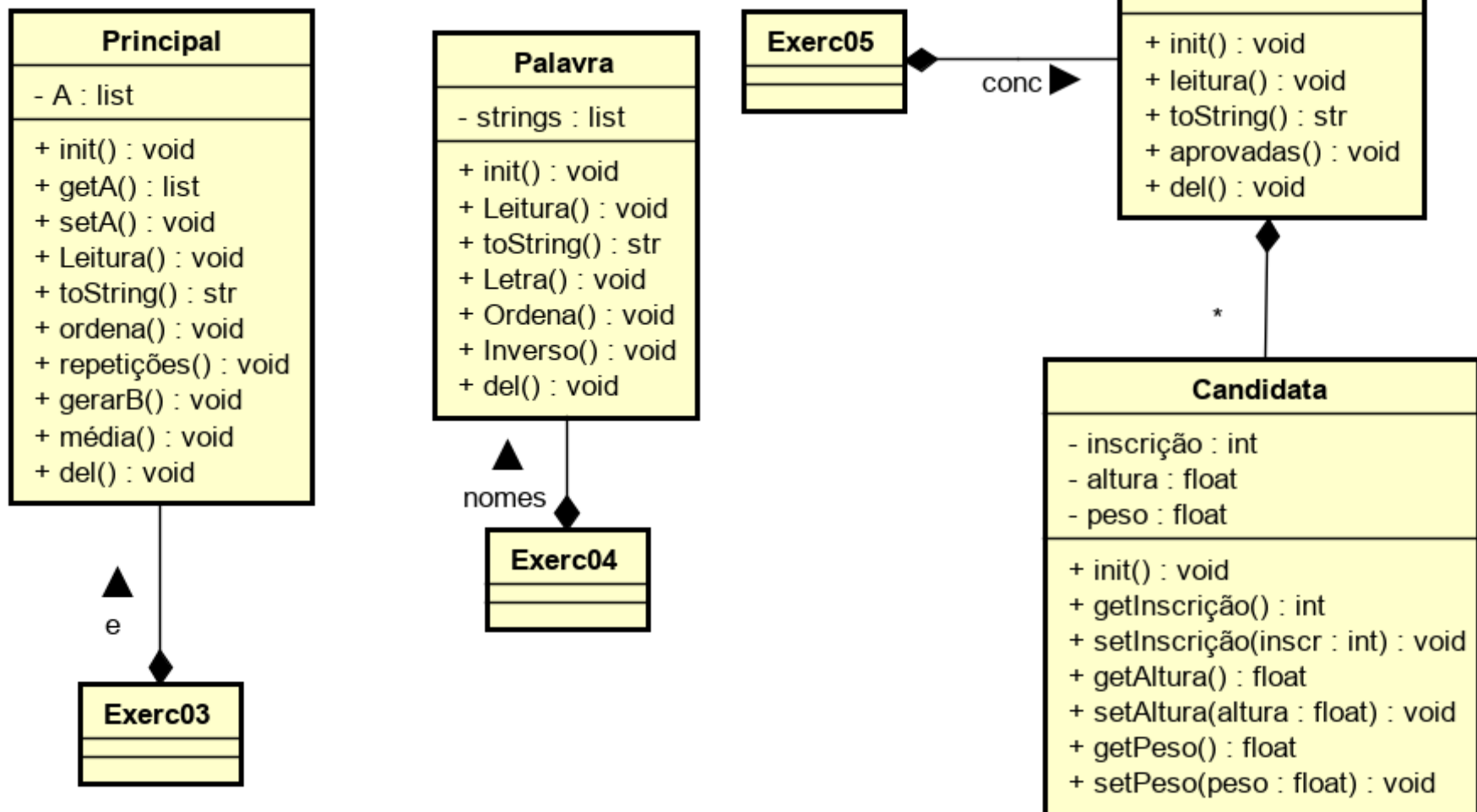
Diagramas de classe:



Diagramas de classe:



Diagramas de classe:



Diagramas de classe:

