SUMÁRIO

1. CARACTERÍSTICAS GERAIS

- Interpretada.
- Orientada a Objetos.
- Suporte a outros paradigmas: estrutural e funcional.
- Fácil integração com outras linguagens.
- Tipagem dinâmica: não há declaração de variáveis.
- Indentação (deslocamento das linhas de código) tem significado na sintaxe do programa.

2. A LINGUAGEM

2.1. Comentários

- Após o caractere '#' tudo até o final da linha é considerado como comentário e ignorado como código (exceto nos comentários funcionais).

```
# Exemplo de comentário.
```

O '#' também é utilizado em comentários funcionais, como na definindo da codificação do arquivo fonte, sendo colocado no início do código:

```
#-- coding: <enconding-name> --
```

Para trabalhar com caracteres em português nos comentários de um programa, usar:

```
#-- coding: latin-1 --
```

2.2. Indentação

- Indentação significa introduzir deslocamentos relativos entre linhas do código do programa. No Python (ao contrário das outras linguagens), a indentação é um elemento de sintaxe para definição dos blocos de código:
  - blocos seguem o símbolo ':';
  - blocos são delimitados pelo uso de indentação.

Exemplo.

```
a=6
print("O valor de a: ",end='') # end='' substitui a quebra de linha
if a==0:                       # por '' (caractere vazio).
   b=1             # Bloco verdade do teste (2 linhas)
   print("zero")
else:              # Bloco falsidade do teste (2 linhas)
   b=2
   print(a)
print("O valor de b:",b)
# Resultado do trecho acima:
#  O valor de a: 6
#  O valor de b: 2
```

- O deslocamento não precisa ser consistente em todo o arquivo, só no bloco de código (onde todos devem ser iguais), mas uma boa prática é ser consistente no projeto todo.

Cuidado ao misturar tabulação e espaços: pode ocasionar problemas de deslocamento num mesmo bloco.

2.3. Variáveis

- A tipagem no Python é dinâmica.
  - Uma variável não tem tipo fixo, ela tem o tipo do objeto que ela "contém" (aponta).
- Não precisam ser declaradas.
- Variáveis são criadas quando valores são atribuídos pela primeira vez.
  - Variáveis devem ser criadas antes de serem referenciadas (usadas).
- "Tudo" no python é como se fosse uma variável (referência).
  - Funções, classes, módulos etc.

2.4. Tipos de Dados

2.4.1. Variáveis mutáveis e imutáveis

• Em variáveis imutáveis, o objeto associado à variável não pode ser alterado, mas outro objeto pode ser associado à variável. São imutáveis: int, float, bool, tuple, str.

• Em variáveis mutáveis, o objeto associado à variável pode ser alterado, e também outro objeto pode ser associado à variável. São mutáveis: list (listas), dict (dicionários).

2.4.2. Variáveis Numéricas

- Imutáveis.

```
num_int = 13
num_int_long = 13L
num_real = 13.0
```

2.4.3. Strings

- Imutáveis.
- Criação.

```
texto1 = "Texto"
texto2 = "Outro texto"
texto3 = """Este texto
tem varias
linhas"""
texto4 = "Este texto"+chr(13)+"tem varias"+chr(13)+"linhas"""
```

- Acesso aos elementos pelo índice (posição da letra, começando em zero, 0).

```
texto = "abcdefgh"
print(texto[2])  # Imprime "c"
print("ABCDEFG"[5])  # Imprime "F"
```

- Principais Métodos (consultar documentação do Python):
  - split, count, index, join, lower, upper, replace.

2.4.4. Tuplas

- Imutáveis.
- Formadas por elementos de qualquer tipo.
- Delimitadas (opcionalmente) por parênteses: "(" e ")".
- Criação.

```
tupla=(a,) # Sem virgula entende como string.
tupla= 'a','b','c','d'
```

- Acesso a elementos pelo índice.

```
print(tupla[2])   # imprime 'c'
print(tupla[1:3]) # imprime "('b','c')"
```

- Vantagem: mais eficiente do que listas.

2.4.5. Listas

- Mutáveis, mas menos eficientes do que tuplas.
- Formadas por elementos de qualquer tipo.
- Criação.

```
lista=[10 ,2 ,3 , "texto" ,20]
```

- Acesso a elementos pelo índice.

```
print(lista[2])    # imprime "3"
print(lista[0:4])  # imprime "[10, 2, 3, 'texto']"
```

- Principais Métodos (consultar documentação do Python):
  - append, count, index, insert, pop, remove, reverse, sort.

```
L=[] # Lista vazia.
print(L)
L.append(1); L.append(2);L.append(3); # Adiciona elementos.
print(L)
L.remove(1) # Remove o elemento 1.
print(L)
RESULTADO DO PROGRAMA
[]
[1, 2, 3]
[2, 3]
```

2.4.6. Dicionários

- Formados por pares de chave-valor.
- Delimitados por chaves. "{" e "}".

```
d={"chave": "valor", "linguagem": "python"}
```

- Principais Métodos (consultar documentação do Python).
  - copy, get, has_key, items, keys, update, values.
- Usando iteradores (otimizado para a instrução `for`: consultar item *2.8.2 Estrutura de Repetição*, pág. 10).

```
d={"chave": "valor", "linguagem": "python"}
# Imprime: "chave linguagem "
for c in d.keys():
   print(c,end=" ")
print()
# Imprime: "valor python"
for v in d.values():
   print(v,end=" ")
print()
# Imprime: "chave = valor; linguagem = python;"
for c, v in d.items ():
   print(c ,"=" ,v,end="; ")
RESULTADO DO PROGRAMA
chave linguagem
valor python
chave = valor; linguagem = python;
```

2.4.7. Vetores e Matrizes

Vetores e matrizes são implementados através de listas.

Um vetor é uma lista de elementos de mesmo tipo/classe. Abaixo, um exemplo de uso (para o uso do for: consultar item *2.8.2 Estrutura de Repetição*, pág. 10). Notar o uso de Vetor=n*[None]: isso permite criar um vetor de tamanho n e já inicializá-lo, no caso com None (tipo inexistente). Mas poderia ser Vetor=n*[0], que resultaria em [0, 0, 0, 0, 0]. Fazendo isso, pode-se atribuir um novo valor a qualquer elemento do vetor de forma mais fácil usando seu índice na estrutura, que em certos algoritmos é bem mais prático do que se usar o método append.

```
#-- coding: latin-1 --
Lista=[1,2,3,4,5] # Lista de literais inteiros.
n=len(Lista)   # Tamanho da lista e do vetor.
Vetor=n*[None] # Define um vetor de tamanho n vazio.
print("Lista:",Lista); print("Vetor:",Vetor)
for k in range(n):
   Vetor[k]=Lista[k]*3 # Não foi necessário usar append.
print("Vetor:",Vetor)  # Imprime vetor redefinido (lista é mutável).
RESULTADO DO PROGRAMA
Lista: [1, 2, 3, 4, 5]
Vetor: [None, None, None, None, None]
Vetor: [3, 6, 9, 12, 15]
```

Já uma matriz é tratada como uma lista de listas de elementos de mesmo tipo/classe. Abaixo, um exemplo de uso.

```
#-- coding: latin-1 --
m=2; n=3  # m linhas, n colunas.
Mat=[]
for i in range(m):
   Mat.append(n*[1]) # Adiciona as linhas com n elementos 1.
print(Mat) # Imprime na forma padrão.
```

```
Mat[1][1]=9 # Altera um elemento
Mat[0][2]=8 # Altera um elemento
for L in Mat: # Imprime na forma matricial.
   print(L) # Imprime uma linha da matriz.
```

RESULTADO DO PROGRAMA

```
[[1, 1, 1], [1, 1, 1]]
[1, 1, 8]
[1, 9, 1]
```

2.5. Operações

2.5.1. Matemáticas

Principais operadores, descrição e exemplos abaixo.

Operador	*Descrição*	*Exemplo*	*Resultado*
+	Soma	2 + 5	7
-	Subtração	7 - 3	4
*****	Multiplicação	2 * 5	10
/	Divisão	17 / 5	3.4
//	Divisão inteira	17 // 5	3
******	Potenciação	2**4	16
%	Módulo (resto da divisão)	17 % 5	2

Prioridade dos operadores e exemplos abaixo.

Prioridade	*Operador*	*Exemplo*	*Resultado*
1	(), funções	2 + 5 * 3	17
2	**	2 * 3**2	18
3	*, /, //, %	4 * 5 % 2	0
4	+, -	21 - 15 / 3	15

Nota. No caso de mesma prioridade, resolve-se da esquerda para a direita

2.5.2. Relacionais e Lógicas

Ambas as operações relacionais e lógicas resultam em verdadeiro (True) ou falso (False).

Principais operadores, descrição e exemplos abaixo.

Operador	*Descrição*	*Exemplo*	*Resultado*
>	Maior	3 > 4	False
>=	Maior ou igual	5 >= 5	True
<	Menor	3 < 5	True
<=	Menor ou igual	2 <= 1	False

<table>
<tr><th>Operador</th><th>Descrição</th><th>Exemplo</th><th>Resultado</th></tr>
<tr><td>==</td><td>Igual</td><td>3 == 4</td><td>False</td></tr>
<tr><td>!=</td><td>Diferente</td><td>3 != 2</td><td>True</td></tr>
<tr><td>not</td><td>Não
<table><tr><th>A</th><th>Not A</th></tr><tr><td>True</td><td>False</td></tr><tr><td>False</td><td>True</td></tr></table></td><td>not True</td><td>False</td></tr>
<tr><td>and</td><td>E
<table><tr><th>A</th><th>B</th><th>A and B</th></tr><tr><td>False</td><td>False</td><td>False</td></tr><tr><td>False</td><td>True</td><td>False</td></tr><tr><td>True</td><td>False</td><td>False</td></tr><tr><td>True</td><td>True</td><td>True</td></tr></table></td><td>True and False</td><td>False</td></tr>
<tr><td>or</td><td>Ou
<table><tr><th>A</th><th>B</th><th>A or B</th></tr><tr><td>False</td><td>False</td><td>False</td></tr><tr><td>False</td><td>True</td><td>True</td></tr><tr><td>True</td><td>False</td><td>True</td></tr><tr><td>True</td><td>True</td><td>True</td></tr></table></td><td>True or False</td><td>True</td></tr>
</table>

Prioridade dos operadores e exemplos abaixo.

Prioridade	*Operador*	*Exemplo*	*Resultado*
0	Matemático	1 + 7 < 2 + 5	False
1	Relacional	not False and True	True
2	and	1<2 or 2<3 and 4<1	True
3	or	False or False or False or True	True
4	not	19%3 > 4 or 19%3==0	True

Notas.
(a) No caso de mesma prioridade, resolve-se da esquerda para a direita.
(b) Operadores relacionais são binários: não existe 1<2<3! Escreve-se 1<2 and 2<3.

2.5.3. Literais

A mais simples e usada é a concatenação de strings através do símbolo "+". Exemplos abaixo.

```
s1 = "Olá"
s2 = "Mundo"
n = 10
s3 = "Alunos"
s4 = s1 + ", " + s2 + "!"
s5= str(n)+ " " + s3 + "." # str converte número para string.
print(s4) # Imprime: "Olá, Mundo!"
print(s5) # Imprime: "10 Alunos."
```

2.6. Funções Embutidas

O interpretador do Python possui várias funções e tipos embutidos que sempre estão disponíveis. Abaixo funções em ordem alfabética (disponíveis em https://docs.python.org/pt-br/3/library/functions.html). Dessas funções alguma já forma utilizadas, como input(), int(), float(), print(), len() e range().

Funções embutidas

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	F	M	
ascii()	filter()	map()	S
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	G	N	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	H	object()	
callable()	hasattr()	oct()	T
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	Z
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

2.7. Entrada e Saída Básica

- A função input() lê dados na entrada padrão (teclado).

```
# Le string.
nome = input("Digite o seu nome: ")
# Le inteiro (int() converte string para int).
```

```
inteiro = int(input("Um numero inteiro: "))
# Le real (float() converte string para float).
real = float(input("Um numero real: "))
```

- A função print() imprime na saída padrão (tela).

```
L="----------------------------------------"
print(L)
Nome="Joao Silva"; P=70.7
print("Bom dia, "+Nome+".")
print("1.",Nome,", voce pesa",P,"quilos.")
print("2. {}, voce pesa {} quilos".format(Nome,P),".")
print("3. {}".format(Nome)+", voce pesa {} quilos".format(P)+".")
print(f"4. {Nome}, voce pesa {P} quilos.")
print("5. %s, voce pesa %.3f quilos."%(Nome,P))
print("6. %s, voce pesa %d quilos."%(Nome,P))
print(L)
RESULTADO
----------------------------------------
Bom dia, Joao Silva.
1. Joao Silva , voce pesa 70.7 quilos.
2. Joao Silva, voce pesa 70.7 quilos .
3. Joao Silva, voce pesa 70.7 quilos.
4. Joao Silva, voce pesa 70.7 quilos.
5. Joao Silva, voce pesa 70.700 quilos.
6. Joao Silva, voce pesa 70 quilos.
----------------------------------------
```

- Múltipla entrada de dados: método split().

```
#-- coding: latin-1 --
# Programa que soma dois Números.
s=input("Digite dois numeros inteiros: ")
data=s.split() # Divide s em dois strings colocados na lista data.
a=int(data[0]) # String 1 convertido para inteiro.
b=int(data[1]) # String 2 convertido para inteiro.
print(a,"+",b,"=",a+b)
```

RESULTADO DO PROGRAMA

Digite dois numeros inteiros: 3 5

3 + 5 = 8

2.8. Controle de Fluxo

2.8.1. Estrutura de Seleção

- *Estruturas com if.*

```
if expr:
   #instruções/bloco
```

```
if expr:
   #instruções/bloco
else:
   #instruções/bloco
```

```
if expr:
   #instruções/bloco
elif exp:
   #instruções/bloco
else:
   #instruções/bloco
```

- *Exemplos.*

```
if x==1:
   y=3
   z=4
```

```
if d==1:
   a=1
   b=2
else:
   a=2
   b=4
```

```
if x<10:
   y=1
elif x<100:
   y=2
else:
   y=3
```

- Os seguintes valores são considerados falsos.
  - None.
  - False.
  - Valor 0 de vários tipos: 0, 0.0, 0L, 0j.
  - Seqüências vazias: "", (), [].
  - Mapeamentos vazios.
  - Instâncias de objetos que definam __nonzero__() que retorne valor False ou 0.
  - Instância de objetos que definem __len__() retornando 0.

2.8.2. Estrutura de Repetição

- *Estrutura*

```
for var in seq:
   #instruções/bloco
```

```
while exp:
   #instruções/bloco
```

- *Exemplos.*

```
for num in range(200):
   print(num)
```

```
from time import time
start = time()
while time()-start<3.0:
   print ("esperando 3 segundos...")
```

- Python fornece a cláusula else para os laços.
  - Será executada quando a condição do laço for falsa.

```
# Exemplo A
lista=["inicio", "meio", "fim"]
for elemento in lista:
   if elemento=="parada":
      break # Sai do for.
   print(elemento)
else:
   print("Laco chegou ao fim.")
RESULTADO DO PROGRAMA
inicio
meio
fim
Laco chegou ao fim.
```

```
# Exemplo B
lista=["inicio", "parada", "fim"]
for elemento in lista:
   if elemento=="parada":
      break # Sai do for.
   print(elemento)
else:
   print("Laco chegou ao fim.")
RESULTADO DO PROGRAMA
inicio
Laco chegou ao fim.
```

  - Nos exemplos A e B acima, a mensagem "Laço chegou ao fim" é impressa ao término da repetição.

2.9. Módulos e Funções de Biblioteca

- Módulos contêm funções definidas em arquivos separados, como bibliotecas.
- Itens são importados utilizando from ou import.

```
from module import function
function()
```

```
import module
module.function()
```

- Módulos são "namespaces". São como "escopo" de definição de objetos. Dessa forma, um mesmo identificador pode ser utilizado em módulos diferentes, evitando conflitos de nome.
  - Podem ser utilizados para organizar nomes de variáveis.
    `mod1.umValor=mod1.umValor-mod2.umValor`
- Exemplo.

```
#-- coding: latin-1 --
# Programa que calcula a hipotenusa de um triângulo retângulo dados os catetos.
from math import sqrt
s=input("Digite os dois catetos: ")
data=s.split()
a=int(data[0])
b=int(data[1])
print("Hipotenusa:",sqrt(a**2+b**2))
```

```
RESULTADO DO PROGRAMA
Digite os dois catetos: 3 4
Hipotenusa: 5.0
```

2.10. Funções do Usuário

2.10.1. Escopo: Regra LGB

- Referências buscam 3 escopos: local (L), global (G), built-in (embutida, B).
- Atribuições criam ou modificam nomes locais por default.
- Pode-se forçar argumentos a serem globais utilizando global.
- Exemplo.

```
x = 99
def func(y):
   z = x+y    # x não é atribuído, então é a global.
   return z
func(1)   # Imprime 100.
```

2.10.2. Argumentos Default

- É possível definir argumentos defaults que podem ser passados opcionalmente (exemplo tirado do console Python).

```
def func(a, b, c=10, d=100):
     print(a, b, c, d)
```

```
>>> func(1,2) [enter]
1 2 10 100
>>> func(1,2,3,4) [enter]
1 2 3 4
```

2.10.3. Número variável de argumentos

- Argumentos podem ser passados para a função na forma de uma lista.

```
def arg_sem_nome (*args):
   for arg in args :
      print("arg :",arg)
arg_sem_nome(1,2,123)
RESULTADO DO PROGRAMA
arg : 1
arg : 2
arg : 123
```

- Argumentos podem ser passados para a função na forma de um dicionário, o nome do argumento é a chave.

```
def arg_com_nome (** kargs):
   for nome, valor in kargs.items():
      print(nome,"=",valor)
arg_com_nome(a=1,b=2,teste=123)
RESULTADO DO PROGRAMA
a = 1
b = 2
teste = 123
```

2.10.4. Múltiplos valores de retorno

- Uma função pode retornar mais de um valor. Exemplo abaixo.

```
def aumenta1(a,b,c):
   return a+1,b+1,c+1
x=2; y=5; z=4
print(x,y,z)
x,y,z=aumenta1(x,y,z)
print(x,y,z)
RESULTADO DO PROGRAMA
 2 5 4
 3 6 5
```

2.11. Orientação a Objetos

2.11.1. Declaração e uso de Classe

```
### Criação da classe ###
class Class:
   contador = 0  # Atributo de classe (estático).
   # Construtor da classe: sempre chamado de __init__.
   def __init__(self, atributo1=0, atributo2=0):
      # Se atributo não definido, usa zero.
```

```
        self.atributo1 = atributo1  # Atributo de objeto.
        self.atributo2 = atributo2  # Atributo de objeto.
    def imprimeAtr(self):  # Método de objeto (com self).
        Class.contador += 1  # Acesso a atributo de classe (prefixo Class).
        # "self.contador += 1" poderia ser utilizado, mas pode induzir
        # confusão na distinção entre atributos de objetos e de classe:
        # não recomendado.
        print(self.atributo1, end=', ')
        print(self.atributo2, end=' ')
    @classmethod # Decorador para indicar que o método é de classe.
    def imprimeCont(cls):  # Método de classe (cls como parâmetro).
        print(f'(impressão no. {cls.contador}).')
### Principal (cliente da classe) ###
a1 = Class(1, 2)  # Cria objeto.
a2 = Class(5 / 2, 2 / 5)
a3 = Class('Pedro', 'Maria')
a4 = Class()  # Cria objeto sem inicialização: usa valores default.
a1.imprimeAtr()  # Evoca método de objeto.
Class.imprimeCont()  # Evoca método de classe.
a2.imprimeAtr()
Class.imprimeCont()
a3.imprimeAtr()
Class.imprimeCont()
a4.imprimeAtr()
RESULTADO DO PROGRAMA
1, 2 (impressão no. 1).
2.5, 0.4 (impressão no. 2).
Pedro, Maria (impressão no. 3)
0, 0 (impressão no. 4).
```

2.11.2. O método __str__

```
### Import para usar funções que manipulam data/tempo. ###
from datetime import datetime, date
### Criação da classe ###
class Pessoa:
    def __init__ (self,nome,nascimento):
        self._nome=nome
        self._nascimento=datetime.strptime(nascimento,'%d/%m/%Y').date()
    def idade(self):
        delta=date.today()-self._nascimento
        return delta.days/365
    def __str__(self): # Define como imprimir utilizando print(objPessoa),
        return "%s, %d anos."%(self._nome, self.idade())
### Principal (cliente da classe Pessoa) ###
P1=Pessoa('Joao Pedro','21/09/1990') # Cria objeto.
P2=Pessoa('Maria Rita','12/03/2003') # Cria objeto.
print(P1) # Impressão conforme definido em __str__().
print(P2)
```

```
RESULTADO DO PROGRAMA
Joao Pedro, 33 anos.
Maria Rita, 21 anos.
```

2.11.3. Atributos de Objeto

- Atributos de objeto (ou atributos de instância) são definidos dentro dos métodos da classe, geralmente dentro do método "__init__", que é o método construtor do objeto.
- Esses atributos pertencem a instâncias individuais da classe, em vez de à própria classe, e podem ter valores diferentes para diferentes instâncias/objetos.
- Para associar os atributos às instâncias da classe, ou objetos, é obrigatório o uso do prefixo "self.", conforme feito nos dois exemplos anteriores.
- Exemplo.

```
#-- coding: latin-1 --
class Cachorro:
   def __init__(self, nome, idade):
      self.nome=nome
      self.idade=idade
# Criando instâncias da classe Cachorro
cachorro1 = Cachorro('Bilu',3)
cachorro2 = Cachorro('Rex',5)
# Acessando os atributos de cada instância
print(cachorro1.nome,',',cachorro1.idade,'anos .')
print(cachorro2.nome,',',cachorro2.idade,'anos .')
RESULTADO DO PROGRAMA
Bilu , 3 anos .
Rex , 5 anos .
```

2.11.4. Atributos de Classe

- São atributos que pertence à classe, e não às instâncias (objetos).
- Equivalentes aos atributos com modificador "static" em classes do Java.
- São compartilhados entre todas as instâncias da classe.
- Os valores iniciais são atribuídos na definição de classe.
- Úteis para casos como Jogos, onde uma imagem deve ser compartilhada por todos os personagens idênticos, economizando memória, ou para implementar contadores (um exemplo foi fornecido no item *2.11.1 Declaração e uso de Classe*).
- Exemplo.

```
class C:
   L=[0] # A lista L é atributo de classe (criada fora de método).
c1=C()
c2=C()
c1.L.append(1)
c2.L.append(2)
# A lista L é acessível a todos os objetos, além da classe.
print(c1.L)  # imprime[0, 1, 2]
```

```
print(c2.L)  # imprime[0, 1, 2]
print(C.L)   # imprime[0, 1, 2].
RESULTADO DO PROGRAMA
[0, 1, 2]
[0, 1, 2]
[0, 1, 2].
```

2.11.5. Métodos de objeto

- São métodos que operam em instâncias específicas de uma classe.
- Recebem automaticamente a instância atual como o primeiro argumento, denominado como self.
- Exemplo.

```
# Declaração da classe.
class Carro:
    def __init__(self, modelo): # Construtor é método de objeto.
        self.modelo = modelo
    def mostrar_modelo(self): # Método de objeto (uso de self).
        print(f"Modelo: {self.modelo}")
# Uso da classe.
meu_carro1 = Carro("Fusca")
meu_carro2 = Carro("Kombi")
meu_carro1.mostrar_modelo()
meu_carro2.mostrar_modelo()
RESULTADO DO PROGRAMA
Modelo: Fusca
Modelo: Kombi
```

2.11.6. Métodos de Classe

- Métodos de classe em Python são métodos que estão ligados à classe em si e não a instâncias específicas da classe.
- Eles são definidos usando o decorador @classmethod antes de sua declaração
- Recebem a própria classe como primeiro parâmetro, denominada de cls.
- Exemplo.

```
# Declaração da classe.
class Cachorro:
   numero_de_cachorros = 0
   def __init__(self, nome, idade):
      self.nome = nome
      self.idade = idade
      Cachorro.numero_de_cachorros += 1
   @classmethod
   def obter_numero_de_cachorros(cls):
      return cls.numero_de_cachorros
# Criando instâncias da classe Cachorro
cachorro1 = Cachorro("Zeca", 3)
```

```
cachorro2 = Cachorro("Rex", 5)
# Usando o método de classe para obter o número de cachorros
print('Número de cachorros:',Cachorro.obter_numero_de_cachorros())
RESULTADO DO PROGRAMA
Número de cachorros: 2
```

2.11.7. Modificação de acesso: público, protegido e privado.

- Utiliza-se uma nomenclatura especial para definir atributos/métodos públicos, protegidos ou privados, mas são formas menos rígidas e "burláveis": o Python não tem modificadores de acesso formais definidos na linguagem, como no C++ ou Java (public, protected e private).
- A nomenclatura para controle de acesso é a abaixo.
  - Privados: nomes que se iniciam com __ ("underscore" duplo).
  - Protegidos: nomes que se iniciam com _ ("underscore" simples).
  - Públicos: os outros nomes possíveis sem "underscore" na frente.
- Exemplo para atributos/métodos de classe.

```
### Classe p/ exemplo de controle de acesso ###
class exemploCA:
    atributo_de_classe_publico = "Valor público (cls)."
    _atributo_de_classe_protegido = "Valor protegido (cls)."
    __atributo_de_classe_privado = "Valor privado (cls)."
    def metodo_classe_publico():
      print("Método de classe público evocado.");
    def _metodo_classe_protegido():
      print("Método de classe protegido evocado.");
    def __metodo_classe_privado():
      print("Método de classe privado evocado.");
### Teste de exemploCA ###
# Imprime valor público.
print('1:',exemploCA.atributo_de_classe_publico)
# Imprime valor protegido.
print('2:',exemploCA._atributo_de_classe_protegido)
# Imprime valor privado colocando o prefixo _exemploCA ("logramento").
print('3:',exemploCA._exemploCA__atributo_de_classe_privado)
# Evoca método de classe público.
exemploCA.metodo_classe_publico()
# Evoca método de classe protegido.
exemploCA._metodo_classe_protegido()
# Evoca método de classe privado colocando o prefixo _exemploCA.
exemploCA._exemploCA__metodo_classe_privado()
# Não imprime valor privado (proteção parcial, burlável conforme acima):
# mensagem de erro.
print('4:',exemploCA.__atributo_de_classe_privado)
# Não evoca método de classe privado (mas burlável conforme acima):
# mensagem de erro.
exemploCA.__metodo_classe_privado()
```

RESULTADO DO PROGRAMA

```
1: Valor público (cls).
2: Valor protegido (cls).
3: Valor privado (cls).
Método de classe público evocado.
Método de classe protegido evocado.
Método de classe privado evocado.
Traceback (most recent call last):
  File "PEDA_Python_04a.py", line 28, in <module>
    print('4:',exemploCA.__atributo_de_classe_privado)
AttributeError: type object 'exemploCA' has no attribute
'__atributo_de_classe_privado'.
Traceback (most recent call last):
  File "PEDA_Python_04a.py", line 30, in <module>
    exemploCA.__metodo_classe_privado()
AttributeError: type object 'exemploCA' has no attribute
'__metodo_classe_privado'.
```

- Exemplo para atributos/métodos de objeto.

```
### Classe p/ exemplo de controle de acesso em objetos ###
class exemploCA:
    def __init__(self):
       self.atributo_de_objeto_publico = "Valor público (obj)."
       self._atributo_de_objeto_protegido = "Valor protegido (obj)."
       self.__atributo_de_objeto_privado = "Valor privado (obj)."
    def metodo_objeto_publico(self):
      print("Método de objeto público evocado.");
    def _metodo_objeto_protegido(self):
      print("Método de objeto protegido evocado.");
    def __metodo_objeto_privado(self):
      print("Método de objeto privado evocado.");
### Teste de exemploCA para objeto ###
# Cria objeto.
e=exemploCA()
# Imprime valor público.
print('1:',e.atributo_de_objeto_publico)
# Imprime valor protegido.
print('2:',e._atributo_de_objeto_protegido)
# Imprime valor privado colocando o prefixo _exemploCA ("logramento").
print('3:',e._exemploCA__atributo_de_objeto_privado)
# Evoca método de objeto público.
e.metodo_objeto_publico()
# Evoca método de objeto protegido.
e._metodo_objeto_protegido()
# Evoca método de classe privado colocando o prefixo _exemploCA.
e._exemploCA__metodo_objeto_privado()
# Não imprime valor privado (proteção parcial, burlável conforme acima).
print('4:',e.__atributo_de_objeto_privado)
```

```
# Não evoca método de classe privado (mas burlável conforme acima).
e.__metodo_objeto_privado()
```

RESULTADO DO PROGRAMA

```
1: Valor público (obj).
2: Valor protegido (obj).
3: Valor privado (obj).
Método de objeto público evocado.
Método de objeto protegido evocado.
Método de objeto privado evocado.
Traceback (most recent call last):
  File "PEDA_Python_04b.py", line 30, in <module>
    print('4:',e.__atributo_de_objeto_privado)
AttributeError: 'exemploCA' object has no attribute
'__atributo_de_objeto_privado'. Did you mean:
'_atributo_de_objeto_protegido'?
Traceback (most recent call last):
  File "PEDA_Python_04b.py", line 32, in <module>
    e.__metodo_objeto_privado()
AttributeError: 'exemploCA' object has no attribute
'__metodo_objeto_privado'. Did you mean: '_metodo_objeto_protegido'?
```

2.12. Tratamento de Exceções.

- Exemplo de tratamento simplificado de exceções ("erros").

```
### Função de divisão com tratamento de exceções ###
def dividir(a,b):
   try:
      resultado=a/b
   except ZeroDivisionError:
      print("Erro: Divisão por zero não é permitida.")
      return None
   except TypeError:
      print("Erro: Os argumentos devem ser números.")
      return None
   else:
      print("Divisão realizada com sucesso!")
      return resultado
   finally: # Impressão padrão ao final da operação.
      print("Operação de divisão finalizada.")
### Teste sa função com diferentes entradas ###
print("----------------------------------------")
# Divisão normal.
print(dividir(10,2))
print("----------------------------------------")
# Divisão por zero.
print(dividir(10,0))
print("----------------------------------------")
# Argumento inválido.
```

```
print(dividir(10,"a"))
print("----------------------------------------")
```

```
RESULTADO DO PROGRAMA
----------------------------------------
Divisão realizada com sucesso!
Operação de divisão finalizada.
5.0
----------------------------------------
Erro: Divisão por zero não é permitida.
Operação de divisão finalizada.
None
----------------------------------------
Erro: Os argumentos devem ser números.
Operação de divisão finalizada.
None
----------------------------------------
```

- Exemplo de tratamento de exceções com exceção criada pelo usuário.

```
### Uma classe de exceção do usuário ###
class DivisaoPorValorNegativoError(Exception):
   """Exceção levantada quando o divisor é um valor negativo."""
   def __init__(self, divisor,
                mensagem="O divisor não pode ser um valor negativo."):
      self.divisor = divisor
      self.mensagem = mensagem
      super().__init__(self.mensagem)
### Função de divisão por positivos com tratamento de exceções ###
def dividir(a, b):
   try:
      if b<0:
         raise DivisaoPorValorNegativoError(b)
      resultado = a/b
   except ZeroDivisionError:
      print("Erro: Divisão por zero não é permitida.")
      return None
   except DivisaoPorValorNegativoError as e:
      print(f"Erro: {e.mensagem} Valor fornecido: {e.divisor}")
      return None
   except TypeError:
      print("Erro: Os argumentos devem ser números.")
      return None
   else:
      print("Divisão realizada com sucesso!")
      return resultado
   finally: # Impressão padrão ao final da operação.
      print("Operação de divisão finalizada.")
### Teste sa função com diferentes entradas ###
print("----------------------------------------")
# Divisão normal.
```

```
print(dividir(10, 2))
print("----------------------------------------")
# Divisão por negativo.
print(dividir(10, -5))
print("----------------------------------------")
# Divisão por zero.
print(dividir(10, 0))
print("----------------------------------------")
# Argumento inválido.
print(dividir(10, "a"))
print("----------------------------------------")
```

```
RESULTADO DO PROGRAMA
---------------------------------------
Divisão realizada com sucesso!
Operação de divisão finalizada.
5.0
---------------------------------------
Erro: O divisor não pode ser um valor negativo. Valor fornecido: -5
Operação de divisão finalizada.
None
---------------------------------------
Erro: Divisão por zero não é permitida.
Operação de divisão finalizada.
None
---------------------------------------
Erro: Os argumentos devem ser números.
Operação de divisão finalizada.
None
---------------------------------------
```

3. EXERCÍCIOS

Programação Imperativa

1) Escreva um programa que leia três números distintos e imprima o maior e o menor valor (identificando-os)

2) Escreva um programa que imprima todos os números inteiros do intervalo fechado de 1 a 10 e a sua soma.

3) Escreva um programa que imprima todos os números inteiros de 10 a 1 (em ordem decrescente) e a suma média.

4) Escreva um programa que leia um número (NUM) e então imprima os múltiplos de 3 e 5, ao mesmo tempo, no intervalo fechado de 1 a NUM.

5) Escreva um programa que leia 10 números e imprima a soma dos positivos e o total (quantidade) de números negativos.

6) Escreva um programa que leia um vetor com 10 números e imprima o elemento de maior valor e sua posição na estrutura. Se houver duplicidade, vale o primeiro valor encontrando.

7) Escreva um programa que leia uma matriz m x n, m e n também fornecidos, e imprima o elemento de menor valor e sua posição na estrutura. Se houver duplicidade, vale o primeiro valor encontrando na varredura por linhas.

Programação Orientada a Objetos

8) Escrever a classe Retangulo de lados a e b em python. Os métodos deverão ser: construtor, área, perímetro e __str__, esse último tal que realize a impressão na forma "a=?; b=?; per=?; area=?". Utilizar um cliente para teste que crie um objeto retângulo com valores para seus lados e imprima seu perímetro e sua área. Tal programa de teste pode ser o ao lado.

```
#main
r=Retangulo(2,3)
print(r)
# Resultado do print(r):
#  a=2; b=3; per=10; area=6
```

9) Escrever a classe Complexo de parte real a e parte imaginária b. Os métodos deverão ser: construtor (a, b padrão nulos), soma, subtração e multiplicação, os três últimos implementados como operadores (métodos __add__, __sub__ e __mul__, respectivamente). Definir também o método __str__. Escrever cliente para teste que execute essas três operações sobre dois números complexos dados, c1 e c2. Parte desse código pode ser o abaixo.

```
class Complexo:
   def __init__(self,a=0,b=0):
      self._a=a
      self._b=b
   def __add__(self,c):
      s=Complexo()
      s._a=c._a+self._a
      s._b=c._b+self._b
      return s
<completar>
```

```
#main
c1=Complexo(2,3)
c2=Complexo(1,-1)
c=c1+c2
print("c=",c)
```

10) Incluir na classe anterior a divisão (__truediv__) entre números complexos. O método correspondente deve tratar a exceção devido à divisão por zero. Testar o programa. Lembrar que a divisão complexa é dada pela expressão abaixo.

$$Z = \frac{Z_1}{Z_2} = \frac{Z_1 \cdot \bar{Z}_2}{Z_2 \cdot \bar{Z}_2} = \frac{(a_1 + ib_1)(a_2 - ib_2)}{(a_2 + ib_2)(a_2 - ib_2)} \rightarrow z = \frac{a_1a_2 + b_1b_2}{(a_2)^2 + (b_2)^2} + i\frac{b_1a_1 - a_1b_2}{(a_2)^2 + (b_2)^2}.$$

11) Escreva em Python uma classe Conta que contenha o nome do cliente, o número da conta, o saldo, seu salário mensal e o limite de saque. Estes valores deverão ser informados no construtor, sendo que o limite não poderá ser maior que o valor do salário mensal do cliente (corrigir automaticamente). Faça um método deposito e um método retira. O método retira irá devolver true ou false dependendo se o cliente pode retirar (saque deve ser menor que o menor entre limite e saldo). Faça um método saldo que retorne o saldo do cliente, e outro que imprima todos os dados do cliente. Crie um programa de teste que use todos os métodos da classe.