

## SMAC03 – Grafos

# 1.1. Preparação do Ambiente e Dicas de Desenvolvimento



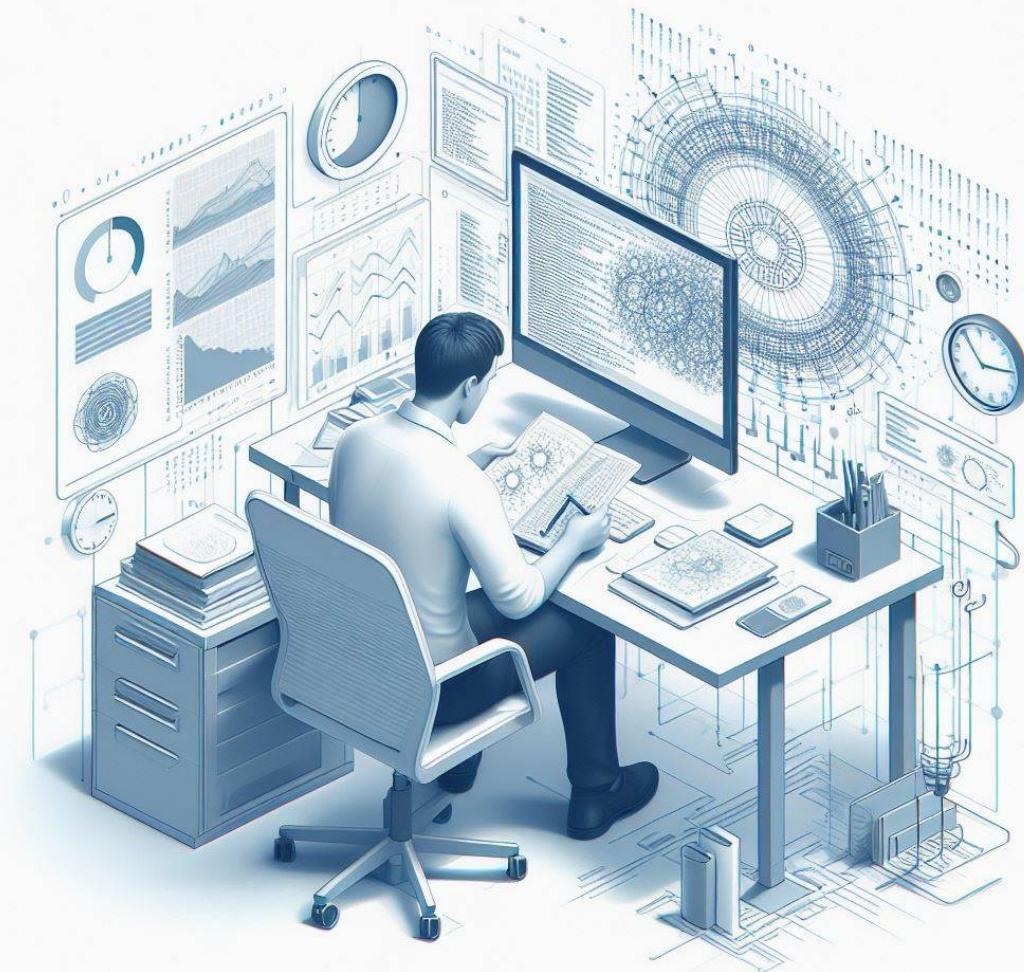
# OBJETIVO

Apresentar a linguagem e respectivo ambiente de desenvolvimento, comandos básicos da linguagem, bem como boas práticas de desenvolvimento de código.

## AGENDA

---

- Ambiente de Desenvolvimento
- Python Básico
- Organização do Código
- Modularização do Código
- Entrada e Saída de Dados
- Hibridização de Linguagens
- Estruturas de Dados
- Execuções via Script
- Medindo o Tempo
- Geração de números pseudoaleatórios
- Pythonicamente



**Linguagem:** Python 3.12.7 ou superior.

**download:** [www.python.org/downloads/](http://www.python.org/downloads/)

**OBS.** Atenção com a incompatibilidade de versões



---

**IDE:** JetBrains PyCharm Community 2024.2.5

**download:** [www.jetbrains.com/pycharm/download](http://www.jetbrains.com/pycharm/download)

---

## Bibliotecas Úteis:

- Numpy – Manipulação de matrizes.
- Matplotlib – Criação de gráficos.
- iGraph – Manipulação de grafos.
- pyCairo – Criação de gráficos.
- Pandas – Manipulação e análise de dados.



*NumPy*



*pandas*



*matplotlib*



*igraph*

**OBS.** Utilize **pip** para instalação de bibliotecas. No Windows por vezes é necessária a instalação via arquivo, os quais podem ser baixados em <https://www.cgohlke.com>.

# Entrada/Saída e Variáveis

**Entradas:** são lidas do teclado com `input()`.

**Saídas:** são escritas na tela com `print()`.

```
# first python program
name = input("Enter your name: ")
print("Hello, ", name, "!")
```

```
Enter your name: John
Hello, John !
```

Para **criar uma variável** basta dar-lhe um nome e um valor inicial utilizando o operador de atribuição (`=`).

```
a = 10 # creates variable "a" and assigns an int value (10)
b, c = 20, a # creates variables b and c, with values 20 and 10 (a)
print(a, b, c)
```

```
10 20 10
```

**Não é necessário especificar o tipo** pois ele é inferido.

Utilize `type()` para descobrir o tipo de uma variável.

O valor e o tipo de uma variável pode ser modificado:

```
a = 1
print(a)
a = "Now a String"
print(a)
```

```
1
Now a String
```

## Nomes de Variáveis

- Nomes de variáveis podem conter letras, o caractere “\_” e números, mas não podem ser iniciados por um número.
- *Case Sensitive*, diferencia maiúsculas e minúsculas. ex. “Nome” ≠ “nome”.
- **Palavras reservadas** da linguagem não podem ser utilizadas para nomear variáveis.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

# Tipos de Dados e Operações

- **Inteiro** (`int`): quantidades contáveis, sem parte fracionária. Ex.: 1, 5, -25, 1000.
- **Real** (`float`): Números com partes fracionárias. Ex.: 1.0, -2.8, 3.1415, 8.74.
- **Lógico** (`bool`): Possuem apenas dois valores: `True` e `False`.
- **Strings** (`str`): Expressam textos. Ex.: “Brasil”, “123456”. Seus valores são expressos entre **aspas duplas**, como nos exemplos.
- **Complexos** (`complex`): Escritos na forma  $x + yj$ , em que  $x$  é a parte real e  $y$  a parte imaginária, representando números complexos. Ex.  $5 + 6j$ ,  $1.2 + 5.7j$ .

## Operações Aritméticas Básicas

Operador	Descrição	Exemplo
+	Soma	$2 + 3 == 5$
-	Subtração	$5 - 2 == 3$
*	Multiplicação	$2 * 5 == 10$
/	Divisão	$15.0 / 2.0 == 7.5$
%	Resto da divisão inteira	$5 \% 2 == 1$
**	Expoente	$5 ** 2 == 25$
//	Divisão inteira	$5 // 2 == 2$

## Operações Lógicas Básicas

Operador	Descrição	Exemplo
==	Igual	$a == b$
!=	Diferente	$a != b$
>	Maior que	$a > b$
<	Menor que	$a < b$
>=	Maior ou igual a	$a >= b$
<=	Menor ou igual a	$a <= b$
and	“e” lógico	$a \text{ and } b$
or	“ou” lógico	$a \text{ or } b$
not	Negação lógica	$\text{not } a$

## Conversões de Tipos

Podem ser realizadas ao colocar o nome do tipo desejado e a informação ou a variável a ser convertida entre parênteses. Ex:

```
val = "100" # a string
val2 = 100 + val
print("Result = ", val2)

-----
TypeError                                 Traceback (most recent call last)
<ipython-input-7-2d4d191aac20> in <module>
      1 val = "100" # a string
----> 2 val2 = 100 + val
      3 print("Result = ", val2)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Não consegue somar  
valores de tipos int e str.

```
val = "100" # a string
val2 = 100 + int(val)
print("Result = ", val2)
```

Result = 200

A conversão é útil na entrada via teclado, cujos valores obtidos com `input()` são `strings`. Para realizar operações aritméticas é necessário convertê-los para os tipos adequados:

```
radius = float(input("Type the circle radius: ")) # gets str input and converts it to float
area = 3.14159265*(radius**2)
print("Circle area = ", area)

Type the circle radius: 10
Circle area = 314.159265
```

## Strings

São sequências de caracteres e possuem características particulares.

**Escape Sequence:** Para inserir caracteres que são ilegais em uma *String* use um caractere de “escape” (a barra invertida \). Ex. Inserir aspas duplas “”.

```
s = "python""  
print(s)  
  
File "<ipython-input-16-89604c9d16d2>", line 1  
    s = "python""  
          ^  
  
SyntaxError: EOL while scanning string literal
```



```
s = "python\""  
print(s)  
  
python"
```

**Raw String:** Para ignorar “escape sequences” de uma string, deve-se escrever a letra r antes dela. Útil se a string possui barras invertidas (ex. caminho de arquivo):

```
s = r"C:\users\JP\python\project\abc.txt"  
print(s)
```

```
C:\users\JP\python\project\abc.txt
```

## Strings

Operações: Concatenação e repetição.

```
s1 = "hello "
s2 = "world"
print(s1 + s2) # + concatenates

hello world
```

```
s1 = "hello "
print(s1 * 3) # * replicates

hello hello hello
```

Funções úteis:

```
1 s = "python is AWESOME!"
2 print(s.upper()) # to uppercase
3 print(s.lower()) # to Lowercase
```

```
PYTHON IS AWESOME!
python is awesome!
```

```
s.split()
```

```
['python', 'is', 'AWESOME!']
```

```
s.replace("AWESOME", "great")
```

```
'python is great!'
```

```
conective = "--"
phrase = connective.join(["Artificial", "Intelligence"])
print(phrase)
```

```
Artificial--Intelligence
```

```
print(phrase[3]) # prints 4th character in string
```

```
i
```

```
#string formatting
print("My name is %s, age %d" %("John", 37))
```

```
My name is John, age 37
```

# Listas

Para representar conjuntos de variáveis relacionadas através de um nome único e índices, o Python utiliza as Listas. Exemplos:

```
a = [] # creates empty list
b = [1,3,7,8,9,6,5,4,2,0] # creates list with pre-defined values
c = [0 for i in range(100)] # create 100-element list, filled with zeros
```

## Acesso a Elementos

```
print(b[7]) # gets 8th element (index starts with 0)
print(b[0:5]) # gets elements from 0 to 5
print(b[0:5:2]) # gets elements from 0 to 5 with step = 2
```

```
4
[1, 3, 7, 8, 9]
[1, 7, 9]
```

```
print(b[:3]) # List slice (up to 3)
print(b[3:]) # List slice (from 3 on)
print(b[::-2]) #List slice (from begin to end, step 2)
```

```
[1, 3, 7]
[8, 9, 6, 5, 4, 2, 0]
[1, 7, 9, 5, 2]
```

## Concatenação e Repetição

```
a = [1,2,3]
b = [4,5]
```

```
c = a + b # concat
print(c)
c = a * 3 #repeat
print(c)
```

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

# Listas

## Métodos úteis para acesso à lista

```
animals = ["cat", "horse", "dog"]
animals.append("bird") # inserts at the end
print(animals)
```

```
['cat', 'horse', 'dog', 'bird']
```

```
animals.remove("cat") # removes item from list
print(animals)
```

```
['horse', 'dog', 'bird']
```

```
animals.insert(2, "fish") # inserts at index 2
print(animals)
```

```
['horse', 'dog', 'fish', 'bird']
```

```
animals.sort() #sorts the list
print(animals)
```

```
['bird', 'dog', 'fish', 'horse']
```

```
animals.reverse()
print(animals)
```

```
['horse', 'fish', 'dog', 'bird']
```

## Matrizes são representadas como listas

```
#representing a matrix
# [[0,1]
#  [2,3]
#  [4,5]]
mat = [[0,1], [2,3], [4,5]]
print(mat) # whole matrix
print(mat[2][1]) # getting specific element
mat[1][0] = 100 # changing element value
print(mat)
```

```
[[0, 1], [2, 3], [4, 5]]
```

```
5
```

```
[[0, 1], [100, 3], [4, 5]]
```

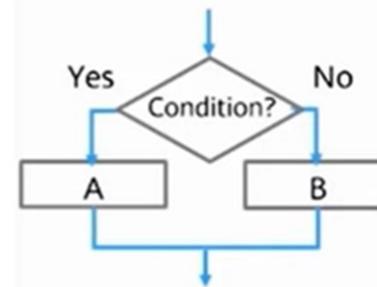
# Estruturas Condicionais

```
if(condicao):
    comando1
    comando2
    ...
    comandon
```

```
if (condicao):
    comando1
    comando2
else:
    comando3
    comando4
```

Duas coisas chamam atenção:  
**Dois pontos ( : )**  
**Indentação**

Fique atento!



Exemplo:

```
p1 = float(input("Entre com nota 1: "))
p2 = float(input("Entre com nota 2: "))
media = (p1+p2) / 2

if(media >= 6):
    print("Aprovado! Media = ", media)
else:
    print("Reprovado! Media = ", media)
```

## Aninhamento

```
ang = float(input("Entre com o angulo: "))

if (ang < 0 or ang > 360): # verificacao
    print("Angulo invalido")
elif (ang > 0 and ang <= 90):
    print("Quadrante 1")
elif (ang <= 180):
    print("Quadrante 2")
elif (ang <= 270):
    print("Quadrante 3")
else:
    print("Quadrante 4")
```

## Estruturas de Repetição

```
print("Utilizando for-range:")
for i in range(0,10,2): # para i de 0 a 10, com passo 2, faça:
    print(i)
```

A função `range` pode receber 1, 2 ou 3 parâmetros:

`range(y)`: y = valor final (exclusivo). O valor inicial = 0, passo = 1.

`range(x, y)`: x = valor inicial (inclusivo), y = valor final (exclusivo), passo = 1.

`range(x, y, z)`: Para passos diferentes de 1. z = valor do passo.

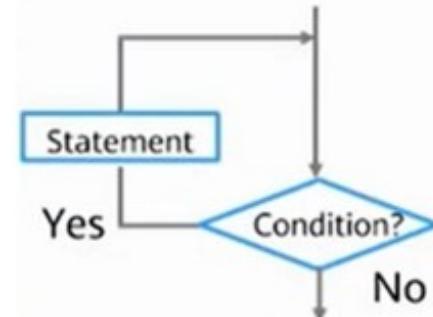
---

```
print("O mesmo com while:")
i = 0    # inicializacao do contador
while(i < 10):  # condicao
    print(i)
    i += 2      # incremento
```

## Estruturas de Repetição

```
# não se esqueça dos dois pontos!
while (condicao):
    comando1 # o bloco a repetir é definido pela indentação
    comando2
    ...
    comandons
```

Se necessário, também podem estar aninhadas.

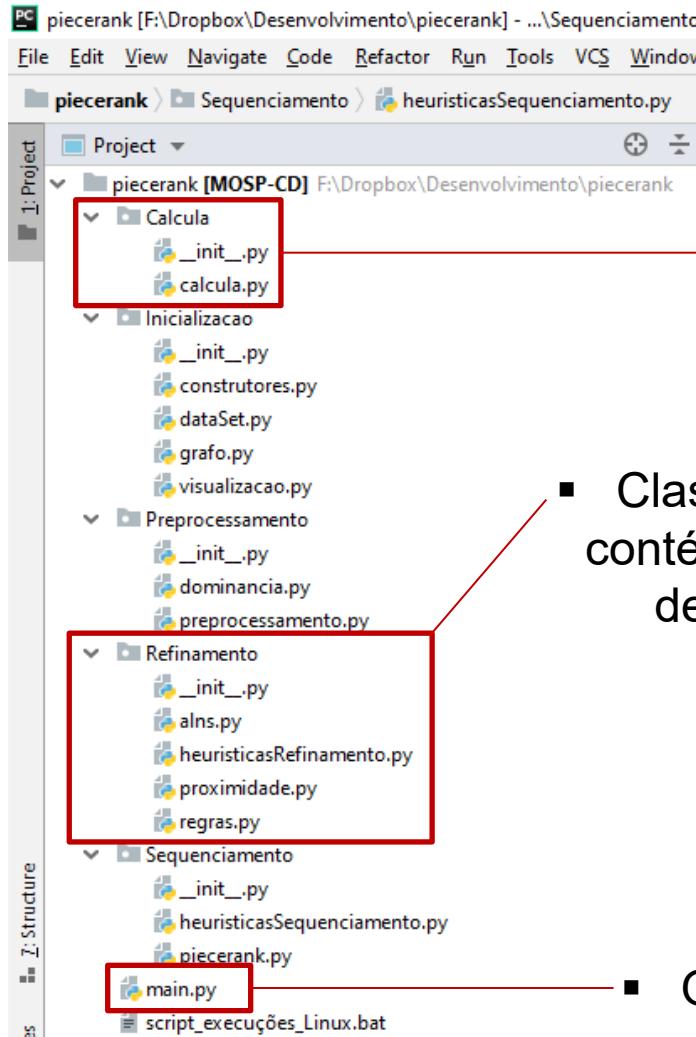


Exemplo:

```
soma = nota = cont = 0
while (nota >= 0):    # repete enquanto não digitar valor negativo
    nota = float(input("Entre com a nota: "))
    if (nota >= 0):
        soma = soma + nota
        cont += 1
media = soma/cont # ultima não conta
print("Media da turma = ", media)
```

```
Entre com a nota: 10
Entre com a nota: 8
Entre com a nota: 9
Entre com a nota: -1
Media da turma = 9.0
```

Organize o código em pastas e arquivos de modo a agilizar o desenvolvimento e a localização de itens a serem alterados.

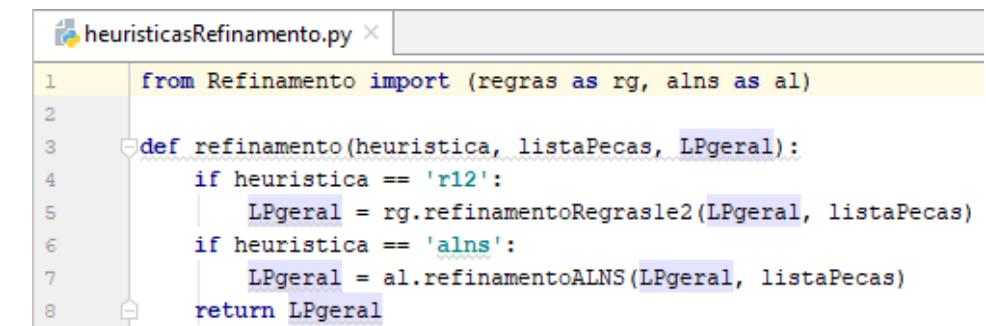


### Exemplos:

- A classe “Calcula” contém o arquivo “calcula.py” que contém apenas funções que realizam cálculos.

- Classe “Refinamento” contém apenas métodos de refinamento de soluções.

O arquivo “heurísticasRefinamento.py” recebe como parâmetro a função a ser executada.



```
from Refinamento import (regras as rg, alns as al)

def refinamento(heuristica, listaPecas, LPgeral):
    if heuristica == 'r12':
        LPgeral = rg.refinamentoRegrasLe2(LPgeral, listaPecas)
    if heuristica == 'alns':
        LPgeral = al.refinamentoALNS(LPgeral, listaPecas)
    return LPgeral
```

- O arquivo “main.py” contém o core da aplicação.

# MODULARIZAÇÃO DO CÓDIGO

Crie funções modularizadas de modo a possibilitar seu acionamento ou alterações de forma independente, sem comprometer o funcionamento de toda a aplicação.

Seja criterioso com os **comentários**

```
''' Função que constroi uma lista de padroes com respectivas peças que ele contém (idPadrao : [peças] qtdPeças)
Entrada: Matriz = Matriz Padrao x Peças; l = qtd. linhas matriz; c = qtd. colunas matriz
Saída: lista Padrões = idPadrao : [pecas, qtdPecas] '''
def constroiListaPadroes(matriz, l, c):
    listaPadroes = {}
    for i in range(l):...
return listaPadroes
```

Entrada da função

Saída da função

Altere o que for preciso na função respeitando sempre que possível sua entrada e saída para um menor impacto na quantidade de alterações em outras funções.

Padronize os arquivos de entrada (*datasets*) de modo a utilizar uma única função para entrada dos dados. Para facilitar o uso de scripts de execução, coloque todos os arquivos em um mesmo local alterando o nome e identificador da instância.

6	7						
1	1	1	0	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	1	0	1	1
0	0	0	1	0	1	1	1
1	0	1	0	0	0	0	0
0	0	1	0	1	0	1	1

Nome do *dataset* (*inst*) e  
identificador da instância (*id*)

Modo de acesso: **r** = read, **b** = binary

```
def criaMatPadraoPeca(inst, id):
    caminho = 'F:/Dropbox/Desenvolvimento/InstanciasPadroesxPecas/' + inst + id + '.txt'
    with open(caminho, 'rb') as f:
        nrows, ncols = [int(field) for field in f.readline().split()]
        data = np.genfromtxt(f, dtype="int32", max_rows=nrows)
    return data
```

Os dados são armazenados  
em uma matriz (data).

Lê um arquivo .txt f, cria uma  
matriz de inteiros (32bits)

Crie uma função que possibilite salvar em arquivo o resultado da execução do método. Salve os resultados de modo a facilitar seu planilhamento e obtenção de dados estatísticos para discussão.

“resultado” armazena o nome da instância, tempo e valor da função objetivo obtidas pelo método

```
resultado = str(inst) + ';' + str("%.4f" % tempo) + ';' + str("%.3f" % F0)  
ds.salvaResultadoResumido(resultado)
```

Precisão de 4 casas decimais

Local de criação e nome do arquivo.

```
def salvaResultado(resultado):  
    arquivo = open('D:/Dropbox/Desenvolvimento/Resultados/Resultados-MetodoX.csv', 'a+')
```

Modo de Escrita:  
**a** = Append (acrescenta)  
**w** = Write (apaga e escreve)

Fecha o arquivo após a escrita

Python permite a integração com C possibilitando o aproveitamento de código ou casos em que a função (ex. função objetivo) tem exigências de desempenho.

Comando para criação da biblioteca de vínculo dinâmico a partir do código escrito em C:

```
gcc -shared -o mcnh.so -fPIC mcnh.c
```

Biblioteca para integração com C

```
import numpy as np
import ctypes

def mcnh(matriz):
    padroes, pecas = np.shape(matriz)
    matriz = np.transpose(matriz)
    lib = ctypes.cdll.LoadLibrary("./Sequenciamento/HNCM/mcnh.so")
    _mcnh = lib.mcnh

    solucao = np.empty((padroes), dtype=np.int32) # Sequencia de padroes
    _mcnh(ctypes.c_void_p(matriz.ctypes.data), ctypes.c_int(padroes), ctypes.c_int(pecas), ctypes.c_void_p(solucao.ctypes.data))
    LP = solucao.tolist()
    return LP
```

Localização da biblioteca de vínculo dinâmico

Deve-se associar na entrada, o tipo de dados suportado em C

Associação dos tipos de dados

```
void mcnh(const int *indata, int padroes, int pecas, int *solucao)
```



# ESTRUTURA DE DADOS

Analise a estrutura mais adequada para manipulação dos dados de modo a facilitar a sua utilização com o melhor desempenho possível.

Utilize a biblioteca [numpy](#) que possui uma grande diversidade de comandos para operações sobre matrizes.

A estrutura de [dicionários](#) funciona como uma tabela *Hash*, sendo eficiente para armazenar informações de objetos.

Define listaNos como um [dicionário](#)

```
listaNos = {}  
nosAdjacentes = []  
listaNos[no] = [grauNo, nosAdjacentes, round(centralidade, 4)]
```

```
0 : 4 [1, 2, 4, 6] 0.7308  
1 : 3 [0, 2, 3] 0.5881  
2 : 5 [0, 1, 3, 4, 6] 0.8822  
3 : 4 [1, 2, 5, 6] 0.7256  
4 : 3 [0, 2, 6] 0.6006  
5 : 2 [3, 6] 0.4349  
6 : 5 [0, 2, 3, 4, 5] 0.8698
```

nosAdjacentes é uma [lista](#) de inteiros

**Exemplos de Comandos:**

- > listaNos[0][0] retorna 4
- > listaNos[0][1] retorna [1, 2, 4, 6]
- > listaNos[0][1][2] retorna 4

A biblioteca [iGraph](#) permite a manipulação de grafos e possui uma grande diversidade de comandos e funções para operações sobre grafos.

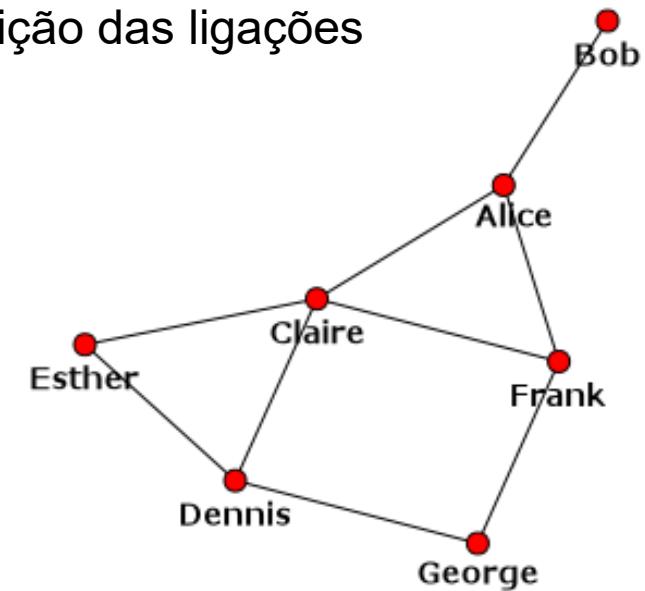
Criação de um grafo `g` que representa uma rede social a partir da definição das ligações

```
>>> g = Graph([(0,1), (0,2), (2,3), (3,4), (4,2), (2,5), (5,0), (6,3), (5,6)])
```

Definição dos [atributos](#) dos nós (vertices, vs).

O mesmo é possível para as arestas (edges, es)

```
>>> g.vs["name"] = ["Alice", "Bob", "Claire", "Dennis", "Esther", "Frank", "George"]
>>> g.vs["age"] = [25, 31, 18, 47, 22, 23, 50]
>>> g.vs["gender"] = ["f", "m", "f", "m", "f", "m", "m"]
```



A biblioteca disponibiliza vários recursos, que vão desde [medidas](#) (ex. grau, distâncias) até [métodos](#) (ex. agrupamento, detecção de comunidades).



A execução do algoritmo via *script* permite automatizar os experimentos, que além de facilitar os testes é útil nos casos em que existe uma grande quantidade de instâncias.

Em main() define-se os **parâmetros de entrada** do método, que podem ser desde valores necessários para **execução do método**, como valores utilizados nas análises.

```
'''Chamada à função main()
    Parâmetros: [1]Dataset, [2]qtdInstancias, [3]melhorSol, [4]executarPreprocessamento, [5]metRefinamento'''
if __name__ == '__main__':
    main(str(sys.argv[1]), int(sys.argv[2]), float(sys.argv[3]), int(sys.argv[4]), str(sys.argv[5]))
```

Em um arquivo .bat são definidos os scripts de execução para cada instância do *dataset*.

```
F:\Dropbox\Desenvolvimento\piecerank>type completa_Artigo.bat
python main.py Random-30-30-2- 5 10.80 0 semRef
python main.py Random-30-30-4- 5 17.40 0 semRef
python main.py Random-30-30-6- 5 21.80 0 semRef
python main.py Random-30-30-8- 5 25.40 0 semRef
```

Trechos de código poderão ser medidos com uma função de tempo a partir de formato pré-configurados.

```
import time # Biblioteca

t_inicio = time.time() # Tempo de início da medição

    << TRECHO DE CÓDIGO A SER MEDIIDO >>

t_total = time.time() - t_inicio # Tempo final da medição

print (str("%.4f" % t_total)) # Precisão de 4 casas
```

**Mais informações sobre a biblioteca e configurações:**

<https://docs.python.org/3/library/time.html>



# GERAÇÃO DE NÚMEROS PSEUDOALEATÓRIOS

Muitos métodos necessitam que sejam gerados números pseudoaleatórios para o seu funcionamento. A seguir alguns comandos para geração destes números de diversas formas.

```
from random import *
# Definir semente para permitir reprodução do experimento
seed(valor_semente)

# Real entre [0, 1) - Ex. 0.0853, 0.9561
("%.4f" % random())

# Escolhe um elemento de uma lista
choice(lista)

# Inteiro entre 0 e 9 - Ex. 3, 7
randrange(0, 9)

# Real entre 1 e 9 - Ex. 4.683, 5.461
("%.3f" % uniform(0, 9))

# Obtém uma nova lista com os k elementos embaralhados.
sample(lista, k=len(lista)) # Para todos os elementos use k = len(lista)
```

**Mais informações sobre a biblioteca e configurações:**

<https://docs.python.org/3/library/random.html>

Um código mais enxuto e melhores desempenhos são obtidos quando se escreve o código de forma “*Pythonica*”.

**Exemplo:** As funções a seguir dobram o valor de cada número par do vetor:

Implementação em C

```
int[] arr = { 1, 2, 3, 4, 5, 6 };
int length = sizeof(arr) / sizeof(arr[0]);
for (int i = 0; i < length; i++) {
    if (arr[i] % 2 == 0) {
        arr[i] *= 2
    }
}
```

Tradução direta em Python

```
arr = [1, 2, 3, 4, 5, 6]
length = len(arr)
for i in range(0, length):
    if arr[i] % 2 == 0:
        arr[i] *= 2
```

Forma Pythonica com *list comprehension*

```
arr = [1, 2, 3, 4, 5, 6]
arr = [x * 2 if x % 2 == 0 else x for x in arr]
```

## Dicas de leituras:

*Effective Python*

<https://hacktec.gitbooks.io/effective-python/content/en/>

*Meus truques preferidos em Python – Parte I*

<https://leportella.com/pt-br/2018/05/07/pytricks-l.html>

*Python Performance Tips*

<https://wiki.python.org/moin/PythonSpeed/PerformanceTips>

*Map vs List comprehension in Python*

<https://dev.to/lyfolos/map-vs-list-comprehension-in-python-2ljj>

# Perguntas? Sugestões?

