material_de_referencia.py

Versão 3.0 – Fase Final do InterFatecs 2025

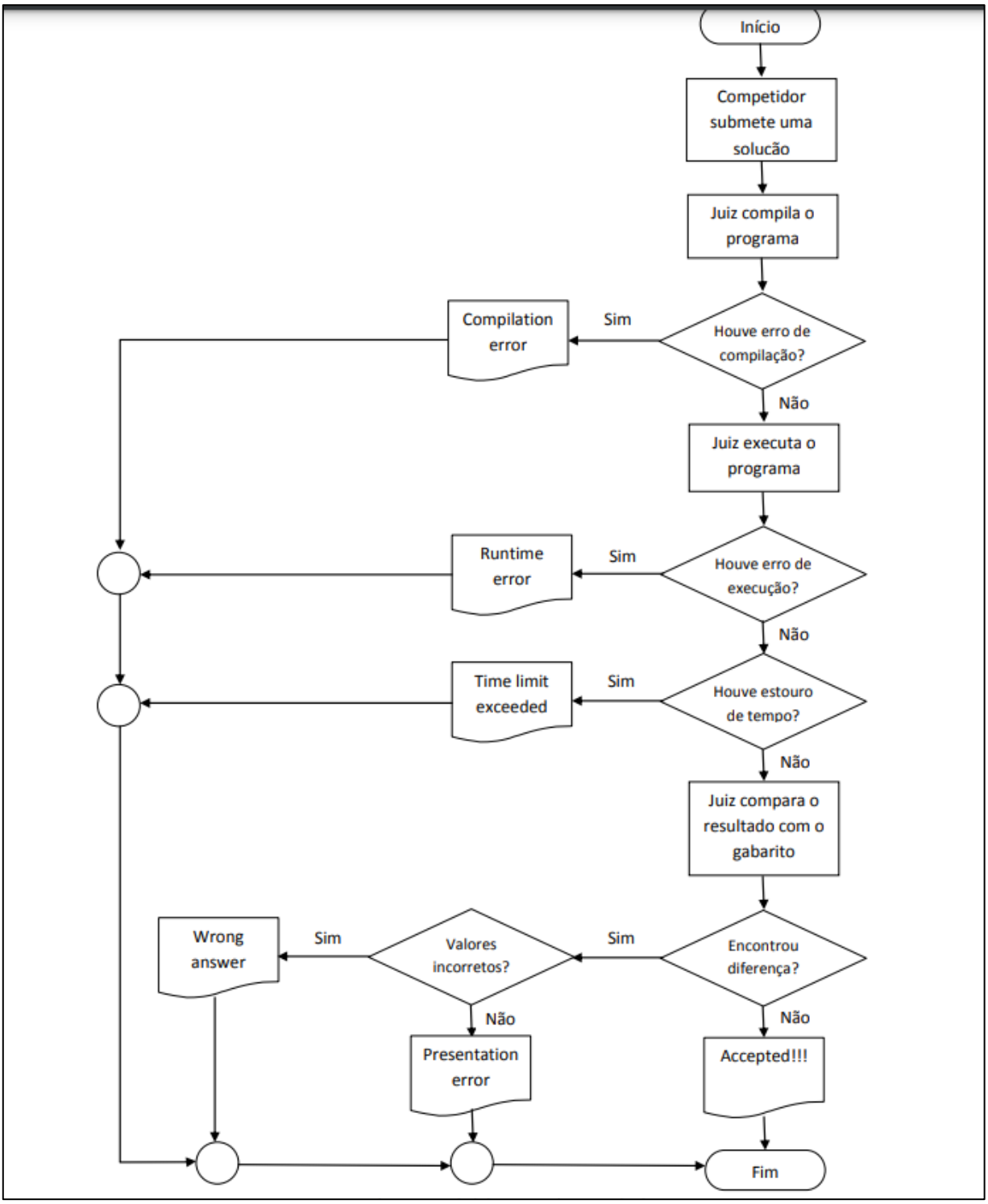
Código Trifásico – FATEC Itapira

Alexandre Alcindo
Gabriel Almir
Rodrigo Polastro

Técnico: Prof. Júnior Gonçalves

Agosto/2025

Fluxo de Correção no Interfatecs



Referência Básica Python

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict ...)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition> :
    <code>
else if <condition> :
    <code>
...
else:
    <code>

if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>

for <variable> in <list>:
    <code>

for <variable> in
range(start,stop,step):
    <code>

for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>)
    <code>
    return <data>
```

Modules

```
import module
module.function()

from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()

f = open(<path>,'r')
for line in f:
    <code>
f.close()

f = open(<path>,'w')
f.write(<str>)
f.close()
```

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: **x,y** stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

Regex – Referência Rápida

Regex Cheat Sheet

Quantifiers	
`a|b`	Match either “a” or “b”
`?`	Match either “a” or “b”
`+`	One or more
`*`	Zero or more
`*?`	Zero or more, but stop after first match
`{N}`	Exactly N number of times (Where N is number)
`{N, M}`	From N to M number of times (Where N and M are numbers)

General Tokens	
`.`	Any character
`\n`	Newline character
`\t`	Tab character
`\s`	Any whitespace character (Including \t, \n, etc)
`\S`	Any non-whitespace character
`\w`	Any word character (Upper/lowercase letters, 0-9, _)
`\W`	Any non-word character
`\b`	Word boundary (Matches between characters)
`\B`	Non-word boundary
`^`	The start of a line
`$`	The end of a line
`\\`	The literal character “\”

CoderPad

For a full Regex guide:

https://bit.ly/regexblog

Pattern Collections	
`[A-Z]`	Match any uppercase character from “A” to “Z”
`[a-z]`	Match any lowercase character from “a” to “z”
`[0-9]`	Match any number
`[asdf]`	Match any character that’s either “a”, “s”, “d”, or “f”
`[^asdf]`	Match any character that’s not any of the following: “a”, “s”, “d”, or “f”

Flags	
`g`	Global, match more than once
`m`	Force $ and ^ to match each newline individually
`i`	Make the regex case-insensitive

Groups	
`( ... )`	Capture group (Matches any 3 characters)
`(?: ... )`	Non-capture group (Matches any 3 characters)
`(?<name> ... )`	Named capture group Group is called “name”

Named Back Reference	
`\k<name>`	Reference named capture group “name” in query

Lookahead and Lookbehind	
`(?!)`	Negative lookahead
`(?=)`	Positive lookahead
`(?<!)`	Negative lookbehind
`(?<=)`	Positive lookbehind

Exemplo de Validação de strings com Regex

```python
import re

placa = input()
padraoAAA_9999 = re.compile(r'[A-Z]{3}[0-9]{4}')
padraoAA_9999 = re.compile(r'[A-Z]{2}[0-9]{4}')
padraoMuitoAntiga = re.compile(r'[AP]{1}[0-9]{1,5}')
padraoNumerica = re.compile(r'[0-9]{1,7}')
padraoMercosul = re.compile(r'[A-Z]{3}[0-9]{1}[A-Z]{1}[0-9]{2}')
if padraoAAA_9999.match(placa) and len(placa) == 7:
    print('Placa AAA-9999')
elif padraoAA_9999.match(placa) and len(placa) == 6:
    print('Placa AA-9999')
elif padraoMuitoAntiga.match(placa) and len(placa) <= 6:
    print('Placa muito antiga')
elif padraoNumerica.match(placa) and len(placa) <= 7:
    print('Placa numerica')
elif padraoMercosul.match(placa) and len(placa) == 7:
    print('Placa mercosul')
else:
    print('Placa invalida')
```

Funções Embutidas

Funções embutidas

A	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
anext()			round()
any()	**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__()

chr (*i*)

Retorna o caractere que é apontado pelo inteiro *i* no código Unicode. Por exemplo, `chr(97)` retorna a string `'a'`, enquanto `chr(8364)` retorna a string `'€'`. É o inverso de *ord()*.

dir (*object*)

Sem argumentos, devolve a lista de nomes no escopo local atual. Com um argumento, tentará devolver uma lista de atributos válidos para esse objeto.

enumerate (*iterable*, *start=0*)

Devolve um objeto enumerado. *iterable* deve ser uma sequência, um *iterador* ou algum outro objeto que suporte a iteração. O método *__next__()* do iterador retornado por *enumerate()* devolve uma tupla contendo uma contagem (a partir de *start*, cujo padrão é 0) e os valores obtidos na iteração sobre *iterable*.

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

help (*request*)

Invoca o sistema de ajuda embutido. (Esta função é destinada para uso interativo.) Se nenhum argumento é passado, o sistema interativo de ajuda inicia no interpretador do console. Se o argumento é uma string, então a string é pesquisada como o nome de um módulo, função, classe, método, palavra-chave, ou tópico de documentação, e a página de ajuda é exibida no console. Se o argumento é qualquer outro tipo de objeto, uma página de ajuda para o objeto é gerada.

Note que se uma barra(/) aparecer na lista de parâmetros de uma função, quando invocando *help()*, significa que os parâmetros anteriores a barra são apenas posicionais. Para mais informações, veja a entrada no FAQ sobre parâmetros somente-posicionais.

ord (*c*)

Dada uma string que representa um caractere Unicode, retorna um número inteiro representando o ponto de código Unicode desse caractere. Por exemplo, `ord('a')` retorna o número inteiro `97` e `ord('€')` (sinal do Euro) retorna `8364`. Este é o inverso de *`chr()`*.

repr (*object*)

Retorna uma string contendo uma representação imprimível de um objeto.

round (*number*, *ndigits=None*)

Retorna *number* arredondado para *ndigits* precisão após o ponto decimal.

sum (*iterable*, */*, *start=0*)

Soma *start* e os itens de um *iterable* da esquerda para a direita e retornam o total. Os itens do *iterable* são normalmente números e o valor inicial não pode ser uma string.

quit (*code=None*)

exit (*code=None*)

Objetos que, quando impressos, imprimem uma mensagem como "Use quit() or Ctrl-D (i.e. EOF) to exit" e, quando chamados, levantam *`SystemExit`* com o código de saída especificado.

filter(função,iterável) = volta um ITERATOR EXPRESSION **(não uma lista)** com todos os elementos em "iterável" que retornarem True na função

OBS: a função tem que ser do tipo de retornar verdadeiro ou falso, e iterável pode ser uma lista , tupla, ou qualquer coisa que consiga se aplicar na função e seja iterável) [iterável = poder pegar 1 por 1 basicamente]

map (função, iteráveis) -> faz a mesma coisa do que o filter, porém pode a função pode realizar qualquer operação, não precisando ser booleana

int(str, base) - conversão entre diferentes bases

```
int("1010", 2)      # binário → 10
int("A", 16)        # hexadecimal → 10
int("12", 8)        # octal → 10
int("z", 36)        # base 36 → 35
```

Extra: Preencher f-string com caracteres

```
f"{'42':*>6}"   # '****42'
f"{'42':_>6}"   # '____42'
f"{'42':-^6}"   # '--42--'
```

Extra: Converter base de número no momento de printar

`b`	Binário	`f"{10:b}"` → `'1010'`
`o`	Octal	`f"{10:o}"` → `'12'`
`x`	Hexadecimal (min.)	`f"{255:x}"` → `'ff'`
`X`	Hexadecimal (mai.)	`f"{255:X}"` → `'FF'`
`d`	Decimal (padrão)	`f"{10:d}"` → `'10'`

Miscelânea de Funções Úteis

Operações em Listas

Operação	Resultado
`s[i] = x`	item *i* de *s* é substituído por *x*
`s[i:j] = t`	fatias de *s* de *i* até *j* são substituídas pelo conteúdo do iterável *t*
`del s[i:j]`	o mesmo que `s[i:j] = []`
`s[i:j:k] = t`	os elementos de `s[i:j:k]` são substituídos por aqueles de *t*
`del s[i:j:k]`	remove os elementos de `s[i:j:k]` desde a listas
`s.append(x)`	adiciona *x* no final da sequência (igual a `s[len(s):len(s)] = [x]`)
`s.clear()`	remove todos os itens de *s* (mesmo que `del s[:]`)
`s.copy()`	cria uma cópia rasa de *s* (mesmo que `s[:]`)
`s.extend(t)` ou `s += t`	estende *s* com o conteúdo de *t* (na maior parte do mesmo `s[len(s):len(s)] = t`)
`s *= n`	atualiza *s* com o seu conteúdo por *n* vezes
`s.insert(i, x)`	insere *x* dentro de *s* no índice dado por *i* (igual a `s[i:i] = [x]`)
`s.pop()` ou `s.pop(i)`	retorna o item em *i* e também remove-o de *s*
`s.remove(x)`	remove o primeiro item de *s* sendo `s[i]` igual a *x*
`s.reverse()`	inverte os itens de *s* in-place

Operação	Resultado	Notas
`x in s`	`True` caso um item de *s* seja igual a *x*, caso contrário `False`	(1)
`x not in s`	`False` caso um item de *s* for igual a *x*, caso contrário `True`	(1)
`s + t`	a concatenação de *s* e *t*	(6)(7)
`s * n` ou `n * s`	equivalente a adicionar *s* a si mesmo *n* vezes	(2)(7)
`s[i]`	*i*-ésimo item de *s*, origem 0	(3)
`s[i:j]`	fatia de *s* de *i* até *j*	(3)(4)
`s[i:j:k]`	fatia de *s* de *i* até *j* com passo *k*	(3)(5)
`len(s)`	comprimento de *s*	
`min(s)`	menor item de *s*	
`max(s)`	maior item de *s*	
`s.index(x[, i[, j]])`	índice da primeira ocorrência de *x* em *s* (no ou após o índice *i*, e antes do índice *j*)	(8)
`s.count(x)`	numero total de ocorrência de *x* em *s*	

Operações em Strings

`str.`**`zfill`**`(`*width*`)`

Retorna uma cópia da String deixada preenchida com dígitos ASCII `'0'` para fazer uma string de comprimento *width*. Um prefixo sinalizador principal (`'+'`/`'-'`) será tratado inserindo o preenchimento *após* o caractere de sinal em vez de antes. A String original será retornada se o *width* for menor ou igual a `len(s)`.

Por exemplo:

```
>>> "42".zfill(5)
'00042'
>>> "-42".zfill(5)
'-0042'
```

`str.`**`rsplit`**`(`*sep=None*, *maxsplit=-1*`)`

Retorna uma lista de palavras na string, usando *sep* como a string delimitadora. Se *maxsplit* é fornecido, no máximo *maxsplit* cortes são feitos, sendo estes mais *à direita*. Se *sep* não foi especificado ou `None` foi informado, qualquer string de espaço em branco é um separador. Exceto pelo fato de separar pela direita, `rsplit()` se comporta como `split()`, o qual é descrito em detalhes abaixo.

`str.`**`rstrip`**`(`[*chars*]`)`

Retorna uma cópia da string com caracteres no final removidos. O argumento *chars* é uma string que especifica o conjunto de caracteres para serem removidos. Se omitidos ou tiver o valor `None`, o argumento *chars* considera como padrão a remoção dos espaços em branco. O argumento *chars* não é um sufixo; ao invés disso, todas as combinações dos seus valores são removidos:

Rsplit como **Rstrip** (R = Right) tmb tem o L para Left -> Lsplit e Lstrip (tudo minúsculo, está em maiúsculo aqui apenas para melhor compreensão)

str.startswith e **str.endswith** (prefix[start:end]): retorna True se começar ou terminar com seu parâmetro e falso se não

str.swapcase() troca maiúsculas por minúsculas e vice versa -> TeStE -> tEsTe

str.ljust(length, char) e **str.rjust**(length, char) – Alinha a string à esquerda ou direita utilizando o char informado

Extra: Ferramenta "Sorted"

Sorted() -> é igual o .sort() de list porém **RETORNA UMA NOVA LISTA** e funciona com qualquer iterável. Tal como o sort(), também possui tem parâmetro **key** que especifica o que deverá ser considerado para a ordenação.

EXEMPLOS:

- Organizando com base em 'casefold' (a>Z)

```
>>> sorted("This is a test string from Andrew".split(), key=str.casefold)
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
```

- Organizado por idade onde esse lambda apenas faz uma 'referenciação', esse 'student' é como se fosse o 'as', está apenas dando um parametro para cada item da lista 'student_tuples', e está chamando cada item o index 2 (terceiro item da tupla para organizar por idade)

```
>>> student_tuples = [
...     ('john', 'A', 15),
...     ('jane', 'B', 12),
...     ('dave', 'B', 10),
... ]
>>> sorted(student_tuples, key=lambda student: student[2])   # sort by age
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

- Para ordenar usando mais de um dado, passe os parâmetros da ordenação em uma tupla:

```
>>> sorted(student_tuples, key=lambda student: (student[1], student[2]))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```

- Organizar usando biblioteca itemgetter (para indexes) pode ser uma maneira de aumentar a eficiência

```
>>> from operator import itemgetter, attrgetter

>>> sorted(student_tuples, key=itemgetter(2))
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

As funções do módulo operator permite múltiplos níveis de ordenação. Por exemplo, ordenar por *grade* e então por *age*:

```
>>> sorted(student_tuples, key=itemgetter(1,2))
[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]
```

- Inverter usando parâmetro reverse (tanto sort quanto sorted)

```
>>> sorted(student_tuples, key=itemgetter(2), reverse=True)
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

Bibliotecas

Biblioteca Collections

Counter(string) -> volta um dicionário com quantas vezes aquela letra apareceu

- **Ex**: Counter('Teste') -> volta um dicionário {'T':1,'e':2,'s':1,'t':1}

Counter().most_common(inteiro) -> volta a quantidade de números que você colocar em inteiros os mais comuns.

- **Ex:** Counter('Teste').most_common(2) -> volta uma lista de tuplas dos que mais apareceram -> [('e',2),('T',1)] #obs> como todos valores depois do "e" só apareceram 1 vez, volta o que aparece primeiro
- **Extra**:> você pode usar index para pegar apenas as letras ou as quantidades que apareceram

defaultdict(funcao) – Dicionário que utiliza o valor padrão retornado pelo parâmetro "funcao" para as chaves que ainda não foram inicializadas. Evita ter que verificar se uma chave já foi inicializada antes de acessá-la.

Biblioteca Itertools

product(lista1,lista2) -> volta todas combinações de lista possíveis -> ex l1 = 1,2 e l2 = 3,4, product retorna -> (1,3),(1,4),(2,3),(2,4) #>para exibir colocar list(product(l1,l2))
a = [1,2,3,4]

combinations(a,2) -> todas combinações com 2 valores de a, não repete, ou seja, 3,1 == 1,3, então não é uma combinação e sim uma variação de uma mesma combinação

permutations(a,2) -> a mesma coisa que combinations porém sem a restrição de variação de uma combinação

accumulate(a) -> o numero atual do index recebe ele + anterior, a saída de a ficaria = [1,3,6,10] onde 3 = a[0]+a[1]///, e ac[2] = 6 pq é == a a[2]+a[1]+a[0]

Biblioteca Copy

deepCopy = copiar objetos complexos (objetos com objetos dentro ou arrays com múltiplas dimensões)

Biblioteca String

Listas de caracteres úteis

```
# Letras minúsculas ASCII (a-z)
ascii_lowercase = 'abcdefghijklmnopqrstuvwxyz'

# Letras maiúsculas ASCII (A-Z)
ascii_uppercase = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

# Todas as letras ASCII (a-z + A-Z)
ascii_letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

# Caracteres ASCII imprimíveis:
# inclui dígitos, letras, pontuação e alguns caracteres de espaço especiais
printable = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;⟸⟹?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

Biblioteca Functools

```
from functools import reduce
reduce(lambda a, b: a+b, [1, 2, 3], 0)  # 6
```

Biblioteca Math

🔢 Aritmética e potências

- `math.ceil(x)` → Arredonda para cima.
- `math.floor(x)` → Arredonda para baixo.
- `math.trunc(x)` → Trunca (remove parte decimal).
- `math.fabs(x)` → Valor absoluto como `float`.
- `math.factorial(n)` → Fatorial de `n` (inteiro).
- `math.prod(iterable, *, start=1)` → Produto de todos os elementos.
- `math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)` → Compara floats com tolerância.

📏 Potência e raízes

- `math.sqrt(x)` → Raiz quadrada.
- `math.pow(x, y)` → Potência (float).
- `math.exp(x)` → e^x.
- `math.log(x, base)` → Logaritmo (base opcional).
- `math.log10(x)` → Log base 10.
- `math.log2(x)` → Log base 2.

📊 Outros utilitários

- `math.comb(n, k)` → Combinações (n choose k).
- `math.perm(n, k)` → Permutações.
- `math.gcd(a, b, *rest)` → Máximo divisor comum.
- `math.lcm(a, b, *rest)` → Mínimo múltiplo comum.
- `math.copysign(mag, sign)` → Copia sinal.
- `math.dist(p, q)` → Distância entre dois pontos (iteráveis).
- `math.fsum(iterable)` → Soma precisa de floats.
- Constantes úteis:
  `math.pi`, `math.e`, `math.tau`, `math.inf`, `math.nan`.

Extra - Graus <-> Radianos com um valor de pi arbitrário

```python
# Definindo o valor de pi
PI = 3.141592653589793

# Converter graus para radianos
def graus_para_radianos(graus):
    return graus * (PI / 180)

# Converter radianos para graus
def radianos_para_graus(radianos):
    return radianos * (180 / PI)
```

Algoritmos Úteis

PRIMOS - Verificar se um número é primo

```python
from math import sqrt, ceil
def isPrime(number):
    if number == 2:
        return True

    if number == 1 or number % 2 == 0:
        return False

    for divisor in range(3, ceil(sqrt(number))+1, 2):
        if number % divisor == 0:
            return False

    return True
```

PRIMOS - Gerar lista dos N primeiros primos/ Encontrar enésimo primo

```python
# retorna uma lista com os primeiros N primos informados
def first_n_primes(numberOfPrimes):
    if numberOfPrimes < 6:
        limit = 15
    else:
        limit = int(numberOfPrimes * (math.log(numberOfPrimes) + math.log(math.log(numberOfPrimes)))) + 10

    sieve = [True] * (limit + 1)
    sieve[0] = sieve[1] = False

    for i in range(2, int(math.sqrt(limit)) + 1):
        if sieve[i]:
            for j in range(i * i, limit + 1, i):
                sieve[j] = False

    primes = [i for i, is_p in enumerate(sieve) if is_p]
    #OBS: se quiser retornar somente o enésimo primo, utilize:
    # → return primes[n-1]
    return primes[:numberOfPrimes]
```

PRIMOS - Gerar lista de primos até um número N (inclui o N se N for primo)

```python
# retorna a lista de primos até o limite informado (INCLUI O LIMITE NO INTERVALO)
def primes_up_to_n(limit):
    sieve = [True] * (limit + 1)
    sieve[0] = sieve[1] = False

    for i in range(2, int(math.sqrt(limit)) + 1):
        if sieve[i]:
            for j in range(i * i, limit + 1, i):
                sieve[j] = False

    primes = [i for i, is_p in enumerate(sieve) if is_p]
    return primes
```

GRAFOS – Verificar se grafo é conexo com DFS

Grafo Conexo = Grafo “conectado”; Quer dizer que é possível viajar entre quaisquer dois pontos.

```python
def dfs(grafo, node, visitados):
    if node in visitados:
        return
    visitados.add(node)
    for vizinho in grafo[node]:
        dfs(grafo, vizinho, visitados)

def eh_conexo(grafo):
    nodes = set(grafo.keys()) #assumindo que grafo é um dicionário de lista de adjacencias
    for inicio in nodes:
        visitados = set()
        dfs(grafo, inicio, visitados)

    return visitados == nodes
```

GRAFOS – Encontrar menor caminho entre dois pontos em GRID com BFS

BFS Passo 1 - Função para retornar posições vizinhas de um nó

```python
def posicoesVizinhas(posicao, qtdLinhasCenario, qtdColunasCenario):
    i, j = posicao[0], posicao[1]
    posicoesVizinhasPossiveis = [
        (i,j+1),
        (i,j-1),
        (i+1,j),
        (i-1,j),
    ]
    posicoesVizinhasValidas = []
    for posicao in posicoesVizinhasPossiveis:
        if 0 <= posicao[0] <= qtdLinhasCenario-1 and 0 <= posicao[1] <= qtdColunasCenario-1:
            posicoesVizinhasValidas.append(posicao)
    return posicoesVizinhasValidas
```

BFS Passo 2 - Definições iniciais

```python
CENARIO = [
    ['I', '.', '.', '.', '.', '.', '.'],
    ['.', '.', '.', '.', '#', '#', '#'],
    ['.', '.', '#', '#', '.', '.', ''],
    ['.', '.', '.', '.', '.', '#', 'F'],
]
QTD_LINHAS_CENARIO = 4
QTD_COLUNAS_CENARIO = 7
POSICAO_INICIAL = (0, 0)
POSICAO_FINAL = (3, 6)
BLOQUEIO = '#'

proximasPosicoes = list()         # ⇒ fila de próximas posições (tuplas)
posicoesAnteriores = dict()       # ⇒ (iAtual, jAtual) → (iAnterior, jAnterior)
proximasPosicoes.append((0,0))    # ⇒ inicia fila com posição inicial
posicoesAnteriores[(0,0)] = None # ⇒ posicao inicial não possui posição anterior
```

BFS Passo 3 - Busca em largura utilizando fila de próximas posições

```python
def buscaEmLargura():
    while len(proximasPosicoes) > 0:
        posicaoAtual = proximasPosicoes.pop(0)
        if posicaoAtual == POSICAO_FINAL:
            menorCaminho = getMenorCaminho(POSICAO_INICIAL, POSICAO_FINAL, posicoesAnteriores)
            return menorCaminho

        for proximaPosicao in posicoesVizinhas(posicaoAtual, QTD_LINHAS_CENARIO, QTD_COLUNAS_CENARIO):
            if proximaPosicao in posicoesAnteriores:
                continue

            if CENARIO[proximaPosicao[0]][proximaPosicao[1]] == BLOQUEIO:
                continue

            posicoesAnteriores[proximaPosicao] = posicaoAtual
            proximasPosicoes.append(proximaPosicao)

menorCaminho = buscaEmLargura()
print(menorCaminho)
# ⇒ [(0, 1), (1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4), (2, 4), (2, 5), (2, 6), (3, 6)]
```

BFS Passo 4 – Função para retornar menor caminho

```python
def getMenorCaminho(posicaoInicial, posicaoFinal, posicoesAnteriores):
    caminhoPercorrido = []
    posicao = posicaoFinal
    while posicao != posicaoInicial:
        caminhoPercorrido.append(posicao)
        posicao = posicoesAnteriores[posicao]

    return caminhoPercorrido[::-1]
```

GRAFOS – Union Find (União de conjuntos disjuntos)

```
QTD_ELEMENTOS = 999
pai = [i for i in range(QTD_ELEMENTOS)]
peso = [0] * QTD_ELEMENTOS
tamanhoConjunto = [1] * QTD_ELEMENTOS
qtdComponentes = QTD_ELEMENTOS

def find(a):
    if pai[a] == a:
        return a
    patriarca = find(pai[a])
    pai[a] = patriarca
    return patriarca

def union(a, b):
    global qtdComponentes
    paiA = find(a)
    paiB = find(b)

    if paiA == paiB:
        return
    qtdComponentes -= 1

    if peso[paiA] < peso[paiB]:
        pai[paiA] = paiB
        tamanhoConjunto[paiB] += tamanhoConjunto[paiA]
    elif peso[paiB] < peso[paiA]:
        pai[paiB] = paiA
        tamanhoConjunto[paiA] += tamanhoConjunto[paiB]
    else:
        pai[paiA] = paiB
        peso[paiB] += 1
        tamanhoConjunto[paiB] += tamanhoConjunto[paiA]
```

GRAFOS – Árvore Geradora Mínima (MST) com algoritmo de Kruskal

A ideia é percorrer a lista de arestas ordenada pelo tamanho de forma crescente e unir os pontos dessas arestas se eles já não estiverem na mesma componente. Utiliza Union-Find para fazer a união e a verificação se dois pontos estão no mesmo conjunto.

```
qtdEstacoes, qtdLigacoes = [int(_) for _ in input().split()]
pai = [i for i in range(qtdEstacoes)]
peso = [0 for _ in range(qtdEstacoes)]

for _ in range(qtdLigacoes):
    estacao1, estacao2, distancia = [int(_) for _ in input().split()]
    ligacoes.append((estacao1-1, estacao2-1, distancia))

ligacoes.sort(key=lambda l: l[2]) #ordena arestas pelo seu tamanho de forma crescente

tamanhoArvore = 0
qtdElementosArvore = 0
for ligacao in ligacoes:
    a, b, distancia = ligacao
    if find(a) != find(b): #verifica se pontos já não estão conectados
        union(a, b)
        qtdElementosArvore += 1
        tamanhoArvore += distancia
        if qtdElementosArvore == qtdEstacoes-1:
            break
print(tamanhoArvore)
```

GRAFOS – Menor caminho entre dois pontos com Dijkstra

```python
import heapq

def dijkstra(grafo, inicio):
    min_heap = [(0, inicio)] # min_heap guarda pares (distância_acumulada, nó)
    distancias = {node: float('inf') for node in grafo} # começam como infinito
    distancias[inicio] = 0  # A distância até o ponto de partida é 0
    anteriores = {node: None for node in grafo}  # armazenar o anteiror no menor caminho

    while min_heap:
        # Pegamos o nó com a menor distância acumulada conhecida
        distancia_atual, atual = heapq.heappop(min_heap)

        # Se essa distância não for a melhor encontrada para esse nó, ignoramos
        # Isso evita processar caminhos piores que já conhecemos
        if distancia_atual > distancias[atual]:
            continue

        for vizinho, peso in grafo[atual].items():
            # Calculamos a distância até ele passando pelo nó atual
            distancia = distancia_atual + peso

            # Se essa nova distância for menor que a distância conhecida
            if distancia < distancias[vizinho]:
                # 1 - Atualizamos a distância para esse vizinho
                distancias[vizinho] = distancia
                # 2 - Armazenamos o antrior do vizinho
                anteriores[vizinho] = atual
                # 3 - Colocamos o vizinho na fila para explorar depois
                heapq.heappush(min_heap, (distancia, vizinho))

    return (distancias, anteriores)
```

```python
grafo = {
    'A': { 'B': 1, 'C': 4},
    'B': { 'A': 1, 'C': 2, 'D': 5},
    'C': { 'A': 4, 'B': 2, 'D': 1},
    'D': { 'B': 5, 'C': 1},
}

def reconstruir_caminho(inicio, fim, anteriores):
    caminho = []
    atual = fim
    while atual is not None:
        caminho.append(atual)
        atual = anteriores[atual]
    caminho.reverse()
    # Só retorna o caminho se ele começar no start (senão significa que não há caminho)
    return caminho if caminho[0] == inicio else []

distancias, anteriores = dijkstra(grafo, 'A')
print(distancias) # ⇒ {'A': 0, 'B': 1, 'C': 3, 'D': 4}
print(anteriores) # ⇒ {'A': None, 'B': 'A', 'C': 'B', 'D': 'C'}
```

GRAFOS – Verificar loop em grafo NÃO DIRECIONADO

```python
from collections import defaultdict
vizinhos = defaultdict(list)

def temCiclo(node, anterior):
    if node in visitado:
        return True

    visitado.add(node)

    for vizinho in vizinhos[node]:
        if vizinho != anterior:
            if temCiclo(vizinho, node):
                return True
    return False

def grafoBidirecionalPossuiCiclo():
    modulosIniciais = list(vizinhos) #evitar "dictionary changed size during iteration"
    for modulo in modulosIniciais:
        if modulo not in visitado:
            if temCiclo(modulo, None):
                return True
    return False

qtdVizinhos = int(input())
visitado = set()
for _ in range(qtdVizinhos):
    a, b = input().split()
    vizinhos[a].append(b)
    vizinhos[b].append(a)
```

GRAFOS – Verificar loop em grafo DIRECIONADO

```python
from collections import defaultdict
vizinhos = defaultdict(list)
BRANCO = 0 # nao visitado
CINZA  = 1 # visitado na stack atual
PRETO  = 2 # visitado e fora da stack atual

def temCiclo(node):
    #para grafos direcionaddos, só é ciclo se for repetido na stack atual
    if cor[node] == CINZA:
        return True

    if cor[node] == PRETO:
        return False

    cor[node] = CINZA

    for vizinho in vizinhos[node]:
        if temCiclo(vizinho):
            return True
    cor[node] = PRETO
    return False

def grafoDirecionadoPossuiCiclo():
    modulosIniciais = list(vizinhos) #evitar "dictionary changed size during iteration"
    for modulo in modulosIniciais:
        if cor[modulo] == BRANCO:
            if temCiclo(modulo):
                return True
    return False

qtdVizinhos = int(input())
cor = defaultdict(lambda: BRANCO)
for _ in range(vizinhos):
    a, b = input().split()
    vizinhos[a].append(b)
```

KNAPSACK – 1/0 – Com Backtracking

```python
# KNAPSACK I/O → pega maior valor otimizando pelo peso, apenas 1 único item de cada
# ===== ENTRADA =====
n = int(input("Número de itens: "))
capacidade = int(input("Capacidade da mochila: "))
pesos = list(map(int, input("Pesos: ").split()))
valores = list(map(int, input("Valores: ").split()))

dp = [[0 for _ in range(capacidade + 1)] for _ in range(n + 1)]

# ===== PREENCHIMENTO DA TABELA DP =====
for i in range(1, n + 1):
    for w in range(capacidade + 1):
        if pesos[i - 1] ⩽ w:
            dp[i][w] = max(dp[i - 1][w], valores[i - 1] + dp[i - 1][w - pesos[i - 1]])
        else:
            dp[i][w] = dp[i - 1][w]

# ===== BACKTRACKING PARA ENCONTRAR OS ITENS ESCOLHIDOS =====
w = capacidade
itens_escolhidos = []

for i in range(n, 0, -1):
    if dp[i][w] =/= dp[i - 1][w]:
        itens_escolhidos.append(i - 1)  # salva o índice do item
        w -= pesos[i - 1]               # reduz a capacidade

# ===== SAÍDA =====
print("\nMelhor valor possível:", dp[n][capacidade])
print("Itens escolhidos (índices):", list(reversed(itens_escolhidos)))
print("Pesos escolhidos:", [pesos[i] for i in reversed(itens_escolhidos)])
print("Valores escolhidos:", [valores[i] for i in reversed(itens_escolhidos)])
```

KNAPSACK – Bounded

```python
# bounded knapsack → maior valor com determinado numeros de cada item
def bounded_knapsack(pesos, valores, quantidades, capacidade):
    n = len(pesos)
    # dp[i][w] = melhor valor usando itens 0..i-1 com capacidade w
    dp = [[0] * (capacidade + 1) for _ in range(n + 1)]

    # Para armazenar quantas unidades do item i foram usadas para dp[i][w]
    escolha = [[0] * (capacidade + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        peso = pesos[i-1]
        valor = valores[i-1]
        max_q = quantidades[i-1]
        for w in range(capacidade + 1):
            dp[i][w] = dp[i-1][w]  # sem pegar o item i
            escolha[i][w] = 0
            # tentar pegar k unidades do item i
            for k in range(1, max_q + 1):
                if k * peso ⩽ w:
                    val = dp[i-1][w - k*peso] + k*valor
                    if val > dp[i][w]:
                        dp[i][w] = val
                        escolha[i][w] = k  # quantidade escolhida do item i
    # Backtracking para recuperar os itens usados
    w = capacidade
    itens_usados = [0] * n
    for i in range(n, 0, -1):
        k = escolha[i][w]
        itens_usados[i-1] = k
        w -= k * pesos[i-1]
    print("Valor máximo:", dp[n][capacidade])
    print("Itens usados (quantidades):", itens_usados)
```

KNAPSACK – Unbounded

```python
# KNAPSACK UNBOUDED → pega o maior valor que cabe naquela capacidade podendo pegar infinitos numeros
def knapsack_unbounded():
    capacidadeBolsa = 20
    peso_itens = [2, 3, 5, 7]
    valores_itens = [3, 5, 10, 13]
    dp = [0 for _ in range(capacidadeBolsa + 1)]
    escolha = [-1 for _ in range(capacidadeBolsa + 1)]  # pra guardar qual item foi escolhido em cada capacidade

    # Preenchendo dp e escolha
    for j in range(1, capacidadeBolsa + 1):
        for i in range(len(peso_itens)):
            if j - peso_itens[i] >= 0:
                val = valores_itens[i] + dp[j - peso_itens[i]]
                if val > dp[j]:
                    dp[j] = val
                    escolha[j] = i  # guarda o índice do item que melhorou o valor

    print("Valor máximo:", dp[capacidadeBolsa])

    # Backtracking para descobrir os itens usados
    j = capacidadeBolsa
    itens_usados = []
    while j > 0 and escolha[j] != -1:
        i = escolha[j]
        itens_usados.append(i)
        j -= peso_itens[i]

    print("Itens usados (índices):", itens_usados)
    print("Pesos dos itens usados:", [peso_itens[i] for i in itens_usados])
    print("Valores dos itens usados:", [valores_itens[i] for i in itens_usados])
```

KNAPSACK – Subset Sum

```python
# subset → caso especial do knapsack, onde só importa o peso,
#        → "É possível encher a mochila exatamente com peso W?"
def subset_sum_backtrack(nums, target, index=0, current_sum=0, subset=[]):
    # Se soma atual iguala o target, achamos uma solução
    if current_sum == target:
        print("Subset encontrado:", subset)
        return True
    # Se soma passar do target ou acabarem os números, volta
    if current_sum > target or index == len(nums):
        return False

    # Tenta incluir o número atual
    if subset_sum_backtrack(nums, target, index + 1, current_sum + nums[index], subset + [nums[index]]):
        return True

    # Tenta não incluir o número atual
    if subset_sum_backtrack(nums, target, index + 1, current_sum, subset):
        return True

    return False
```

Gera subconjuntos de elementos de array

```python
# gera subconjuntos de combinações de caracteres.
# ExemploS:
# - combinations('ABC', 1) → [A, B, C]
# - combinations('WXYZ', 2) → [WX, WY, WZ ... YZ]
def combinations(entries, length):
    if length == 0:
        return ['']

    if len(entries) == 0:
        return []

    entries = sorted(entries)
    result = []

    for i in range(len(entries)):
        current = entries[i]
        remaining = entries[i + 1:]
        for comb in combinations(remaining, length - 1):
            result.append(current + comb)

    return result
```

Maior subsequência contígua (Kadane)

```python
def kadane(arr):
    max_soma = atual = arr[0]
    start = end = temp_start = 0

    for i in range(1, len(arr)):
        if arr[i] > atual + arr[i]:
            atual = arr[i]
            temp_start = i
        else:
            atual += arr[i]

        if atual > max_soma:
            max_soma = atual
            start = temp_start
            end = i

    return max_soma, arr[start:end + 1]
```

Maior sequência de elementos iguais

```
def encontraMaiorSequenciaIgual(lista):
    indicesMaiorSequencia = list()
    tamanhoMaiorSequencia = -inf

    pecaAnterior = lista[0]
    posicoesSequencia = [0]
    for j in range(1, len(lista)):
        pecaAtual = lista[j]

        if pecaAtual == pecaAnterior:
            posicoesSequencia.append(j)
            if len(posicoesSequencia) > tamanhoMaiorSequencia:
                tamanhoMaiorSequencia = len(posicoesSequencia)
                indicesMaiorSequencia = posicoesSequencia
        else:
            pecaAnterior = pecaAtual
            posicoesSequencia = [j]
    return indicesMaiorSequencia
```

Verificar se um número pertence à sequência Fibonacci

```
def isInFibonacci(n):
    a, b = 1, 1
    while a < n: a, b = b, a+b
    return a == n
```

Verificar se um número é fatorial

```
def isFactorial(n):
    f = 1
    i = 2
    while f < n:
        f *= i
        i += 1
    return f == n
```