# Material de Referência.py

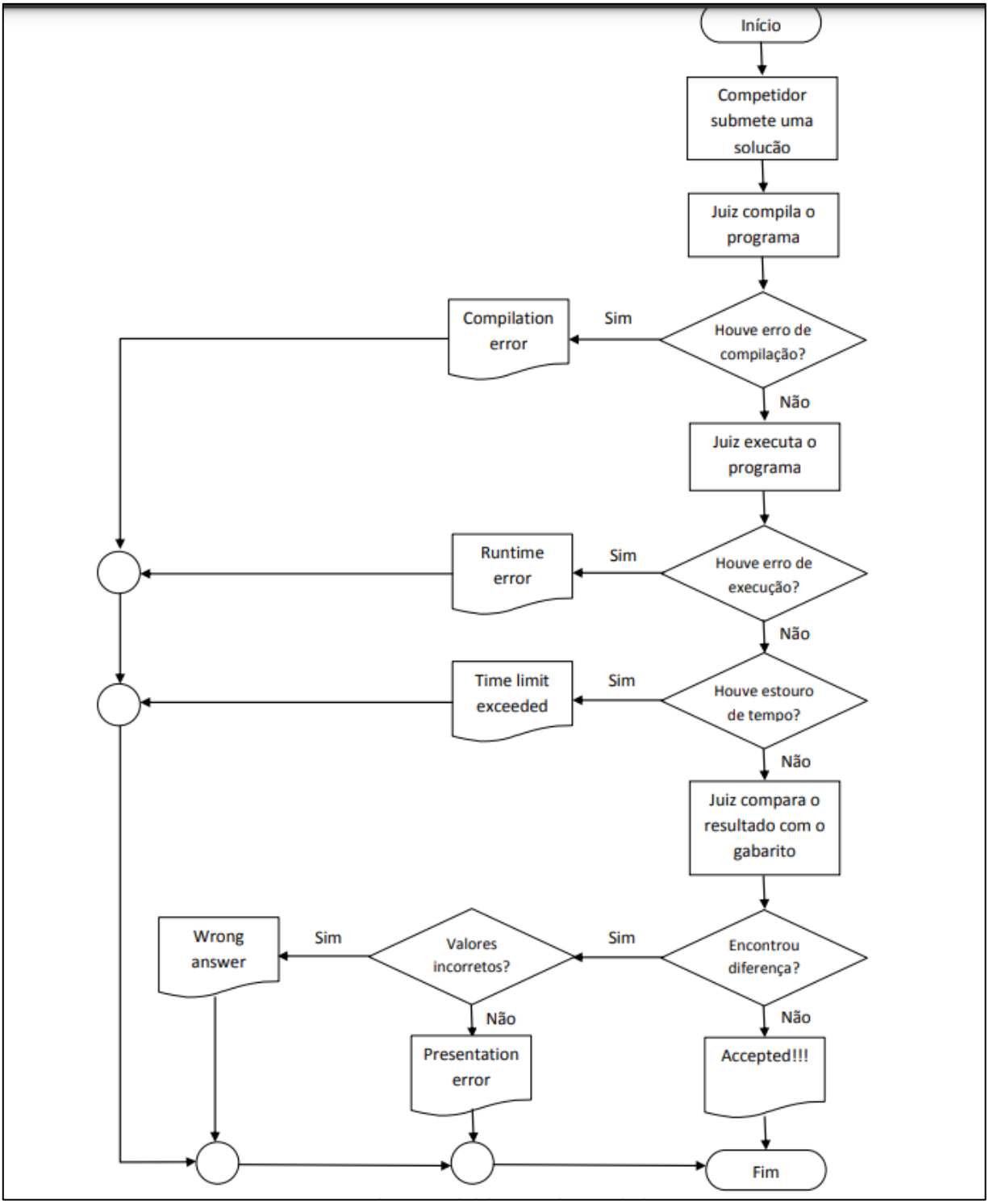
## Código Trifásico – FATEC Itapira

Alexandre Alcindo
Gabriel Almir
Rodrigo Polastro

Técnico: Prof. Júnior Gonçalves

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.

# 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 uma lista com todos os iteráveis 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 básicamente]

**map** (função, iteráveis) -> faz a mesma coisa do que o filter porem pode receber qualquer valor além de verdadeiro ou falso

**math.isqrt(n)** -> retorna inteiro da raiz (sqrt() normal retorna float da raiz)

## 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 funciona com qualquer iterável (sort LEVEMENTE mais eficiente que sorted), quanto sort e sorted tem parâmetro key que especifica em base oque deve ser organizado, ex:

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)]
```

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 de teste:

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 so apareceram 1 vez, volta o que aparece primeiro ##extra> você pode usar index para pegar apenas as letras ou as quantidades q apareceram

## 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]

# Algoritmos Úteis

## Remover acentos de string

```python
import unicodedata

def remove_acentos(texto):
    return ''.join(c for c in unicodedata.normalize('NFD', texto) if unicodedata.category(c) != 'Mn')
```

## Copiar Objetos Complexos:

Quando precisar copiar objetos complexos (objetos com objetos dentro ou arrays com múltiplas dimensões), utilize a função “deepcopy” da biblioteca “copy”. Caso contrário, apenas criará um ponteiro para o objeto original

```python
import copy

nestedObject = {
    "a": {'primeiro': 1},
    "b": {'segundo': 2},
    "c": {'terceiro': 3}
}

# => esse objeto é apenas um ponteiro para o objeto original
shallowCopy = copy.copy(nestedObject)
shallowCopy['a']['primeiro'] = 'duplicado'
# => {'a': {'primeiro': 'duplicado'}, 'b': {'segundo': 2}, 'c': {'terceiro': 3}}
# => {'a': {'primeiro': 'duplicado'}, 'b': {'segundo': 2}, 'c': {'terceiro': 3}}

# => somente usando uma cópia profunda um objeto realmente novo criado
deepCopy = copy.deepcopy(nestedObject)
deepCopy['a']['primeiro'] = 'sem duplicação'
# => {'a': {'primeiro': 'duplicado'}, 'b': {'segundo': 2}, 'c': {'terceiro': 3}}
# => {'a': {'primeiro': 'sem duplicação'}, 'b': {'segundo': 2}, 'c': {'terceiro': 3}}
```

## Verificar se um número pertence à sequência fibonacci

```python
'''_____________________________________________________

Fibonacci
_____________________________________________________'''

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

**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)), 2):
        if number % divisor == 0:
            return False

    return True
```

**Encontrar o enésimo primo (otimizado)**

```python
import math

def prime(n):
    if n < 6:
        limit = 15
    else:
        limit = int(n * (math.log(n) + math.log(math.log(n)))) + 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]

    return primes[n - 1]
```

## Busca em profundidade em um grafo (Depth First Search – DFS)

```
pontosVisitados = []
pontosCidade = None
ruasCidade = None

def dfs(numPonto):
    global pontosVisitados
    if pontosVisitados[numPonto] == True:
        return

    pontosVisitados[numPonto] = True
    for numConexao, haConexao in enumerate(conexoes[numPonto]):
        if haConexao is True:
            dfs(numConexao)

def testaCaminhos():
    for pontoInicial in range(pontosCidade):
        global pontosVisitados
        pontosVisitados = [False] * pontosCidade
        dfs(pontoInicial)
        if False in pontosVisitados:
            print("N")
            return

    print("S")
```

## Busca em largura em um grafo (Breadh First Search – BFS) – APLICADO EM GRID

1 – Função para retornar posições vizinhas de um nó

```
def posicoesVizinhas(posicao):
    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] <= linhasCenario-1 and 0 <= posicao[1] <= colunasCenario-1:
            posicoesVizinhasValidas.append(posicao)

    return posicoesVizinhasValidas
```

2 – Loop de busca utilizando “fila”

```
while len(proximasPosicoes) > 0:
    posicaoAtual = proximasPosicoes.pop(0)
    if posicaoAtual == POSICAO_CHAVE:
        imprimeTamanhoMenorCaminho()
        quit()

    for proximaPosicao in posicoesVizinhas(posicaoAtual):
        if proximaPosicao in posicoesAnteriores:
            continue

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

3 – Calcular e printar tamanho do menor caminho entre dois pontos

```
def imprimeTamanhoMenorCaminho():
    qtdMovimentos = 0
    caminhoPercorrido = []
    posicao = POSICAO_CHAVE
    while posicao != POSICAO_INICIAL:
        qtdMovimentos+=1
        caminhoPercorrido.append(posicao)
        posicao = posicoesAnteriores[posicao]

    print(len(caminhoPercorrido))
```

**Validar entradas com padrões de regex (compile e match)**

```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')
```