**MODULO 1**

**Introdução ao Python e à programação informática.**

# **Lista de instruções (IL) comandos que o computador** reconhece, também chamado de **alfabeto da máquina.**

# ****Elementos da linguagem:****

# um alfabeto: um conjunto de símbolos

# um lexis: (ou seja, um dicionário) um conjunto de palavras que a linguagem oferece aos seus utilizadores

# uma sintaxe: um conjunto de regras (formais ou informais, escritas ou sentidas intuitivamente) utilizadas para determinar se uma determinada sequência de palavras forma uma frase válida

# semântica: um conjunto de regras que determinam se uma determinada frase faz sentido

# Linguagens de programação de alto nível: As linguagens que os humanos são capazes de escrever e a máquina de entender.

# Source code: um programa escrito numa linguagem de programação de alto.

# Machine code: programa escrito em código de máquina.

# ****Source file:**** o ficheiro que contém o source code.

**A programação precisa ser correta nos sentidos:**

* alfabeticamente - um programa precisa de ser escrito num guião reconhecível, tal como romano, cirílico, etc.
* lexicamente - cada linguagem de programação tem o seu dicionário e é preciso dominá-lo; felizmente, é muito mais simples e menor do que o dicionário de qualquer língua natural;
* sintaticamente - cada linguagem tem as suas regras, e estas devem ser obedecidas;
* semanticamente - o programa tem de fazer sentido.

**Como transformar um programa em linguagem de alto nível em linguagem de máquina:**

* COMPILAÇÃO – o compilador lê o código fazendo todas as análises sintáticas, traduz o programa uma vez ou sempre que o código for modificado e gera um ficheiro (arquivo) que contém o código de máquina.
* INTERPRETAÇÃO – o programa é traduzido pelo interpretador cada vez que é executado, sendo assim tanto quem desenvolveu quanto o utilizador final precisa do intérprete para o executar o programa. (Consome mais memória, é mais lento que a copilada, são de alto nível)

# 

**Scripts:** source programs codificados que utilizam linguagens interpretadas também chamadas de linguagens de scripting.

**PYTHON:** linguagem de programação interpretada, orientada a objetos, e de alto nível com semântica dinâmica, utilizada para programação de uso geral. Criado por [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum), nascido em 1956 em Haarlem, Holanda, o nome da linguagem vem de uma antiga série de comédia televisiva da BBC chamada Monty Python's Flying Circus.

**Objetivos do Python definidos por Guido van Rossum em 1999:**

* uma linguagem fácil e intuitiva, tão poderosa como a dos principais concorrentes;
* de open source, para que qualquer pessoa possa contribuir para o seu desenvolvimento;
* código que seja tão compreensível como o inglês simples;
* adequado para tarefas quotidianas, permitindo tempos de desenvolvimento curtos.

**Características:**

* é fácil de aprender - o tempo necessário para aprender Python é menor do que para muitas outras linguagens; isto significa que é possível iniciar a programação em si mais rapidamente;
* é fácil de ensinar - a carga de trabalho de ensino é menor do que a necessária para outras linguagens; isto significa que o professor pode colocar mais ênfase em técnicas de programação gerais (independentes da linguagem), não desperdiçando energia em truques exóticos, estranhas exceções e regras incompreensíveis;
* é fácil de usar para escrever novo software - é muitas vezes possível escrever código mais rapidamente quando se usa Python;
* é fácil de compreender - é também frequentemente mais fácil e rápido de compreender o código de outra pessoa se for escrito em Python;
* é fácil de obter, instalar e implementar - o Python é gratuito, aberto e multiplataforma; nem todas as linguagens se podem gabar disso.

**Linguagens com propriedades e predisposições comparáveis ao Python:**

* **Perl** - uma linguagem de scripting originalmente de autoria de Larry Wall;
* **Ruby** - uma linguagem de scripting originalmente escrita por Yukihiro Matsumoto.

**Versões do Python**

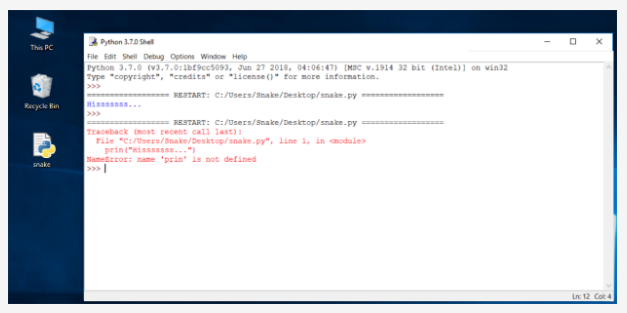
* **CPython:** a implementação padrão da linguagem de programação Python.
* **Cython:** traduz automaticamente o código Python em código "C" (complicado e falador, mas ágil).
* **Jython:** Python escrito em Java.
* **PyPy:** ferramenta para pessoas que desenvolvem Python. É um Python dentro de um Python. Representa um ambiente Python escrito em linguagem Python, chamado **RPython** (Restricted Python).
* **Criar arquivo 🡪** Abre a IDLE > File > New File

Na nova janela 🡪 File > Save as > Escolher pasta > Nomear

* **Escrever uma mensagem na tela** 🡪 print("MENSAGEM")
* **Salvar alterações** 🡪 File > Save > Run > Run Module

**Mensagem de erro:**

* o **traceback** (que é o caminho que o código percorre através de diferentes partes do programa - pode ignorá-lo por agora, uma vez que está vazio num código tão simples);
* a **localização do erro** (o nome do ficheiro contendo o erro, o número da linha e o nome do módulo); nota: o número pode ser enganador, uma vez que o Python normalmente mostra o local onde primeiro se notam os efeitos do erro, não necessariamente o erro em si;
* o **conteúdo da linha errada**; nota: a janela do editor IDLE não mostra os números das linhas, mas mostra a localização atual do cursor no canto inferior direito; use-a para localizar a linha errada num source code longo;
* o **nome do erro** e uma breve explicação.



**MODULO 2**

**Função** é uma parte separada do código do computador capaz de:

* **causar um qualquer efeito** (por exemplo, enviar texto para o terminal, criar um ficheiro, desenhar uma imagem, reproduzir um som, etc.);
* **avaliar um valor** (por exemplo, a raiz quadrada de um valor ou o comprimento de um dado texto) e **devolvê-lo como o resultado da função**; é isto que faz as funções Python serem os parentes dos conceitos matemáticos.

**Tipos de função:**

* **Incorporada:**  podem vir do próprio Python, como a função print;
* **Módulos:** provenientes de um ou mais dos add-ons de Python que requerer instalação separada
* **Escrita pelo próprio programador**, colocando tantas funções quantas quiser e precisar dentro do seu programa para o tornar mais simples, mais claro e mais elegante.

**Instrução** é um comando para executar uma tarefa específica quando executada, por exemplo, para imprimir uma determinada mensagem no ecrã. Os programas de computador são coleções de instruções.

**print()** é uma função incorporada o Python que imprime/faz output de uma mensagem especificada para a janela do ecrã/consola.

* **Impressão multi-linha:** pode usar **aspas triplas** para imprimir strings em várias linhas a fim de tornar o texto mais fácil de ler, ou criar um desenho especial baseado em texto. Ex:

print(

"""

MENSAGEM

""")

* **print(‘’2”) != print(2) 🡪** na memória do computador, estes dois valores são armazenados de formas completamente diferentes - a string existe apenas como uma string - uma série de letras. O número é convertido em representação mecânica (um conjunto de bits).

**Funções incorporadas:** estão sempre disponíveis e não têm de ser importadas. O Python 3.8 vem com 69 funções incorporadas.  [Biblioteca Padrão Python](https://docs.python.org/3/library/functions.html).

I**nvocação de função** ou **chamada de função**: o processo para chamar uma função. É necessário usar o nome da função seguido de parêntesis.

**Argumentos:** Deve separar os argumentos com uma vírgula, por exemplo, print("Hello,", "world!").

* **Tipos de argumentos:** string, números, caráteres, valores lógicos, objetos.

**Argumentos posicionais** são aqueles cujo significado é ditado pela sua posição, por exemplo, o segundo argumento é apresentado após o primeiro, o terceiro é apresentado após o segundo, etc.

**Argumentos de keyword** são aqueles cujo significado não é ditado pela sua localização, mas por uma palavra especial (keyword) utilizada para os identificar. Os loops end e sep podem ser usados para formatar o output da função print() .

* O parâmetro sep especifica o separador entre os argumentos de output.





* O parâmetro end especifica o que imprimir no final da declaração print.

 PS: junta na mesma linha

**Caractere de escape**: um caractere especial que anuncia que o próximo caractere tem um significado diferente, por exemplo \n

**Caractere de newline:** a barra invertida (\) que incita o console a iniciar uma **nova linha de output**. Uma função “vazia” print() também faz output de uma linha vazia para o ecrã.

* **LITERAIS**

são notações para representar alguns valores fixos em código.

**Type:** característica do valor numérico que determina o seu tipo, intervalo e aplicação.

**Tipos de literais**:

* **Sintaxe para strings:** são delimitadas com **aspas**, por exemplo, "I am a string" (aspas duplas), ou 'I am a string, too' (aspas simples). Quase tudo o que colocar dentro das aspas será tomado literalmente, não como código, mas como **dados**.

*Codificar uma apóstrofe ou uma aspa dentro de uma string:* pode usar o caratere de escape:

print("I like \"Monty Python\"")

print('I\'m Monty Python.')

print('I like "Monty Python"')

* **Inteiros:** (ou simplesmente **int**s): números escritos sem um componente fracionário, por exemplo, 256, ou -1 (inteiros negativos).
* **Floating-point:** (ou simplesmente **float**s): números que contêm (ou são capazes de conter) um componente fracionário, por exemplo 1.27. Neste tipo utiliza-se ponto e não virgula. Pode se escrever o valor 0.4 como .4; o valor de 4.0 como 4.. No float também pode utilizar a letra e. Por exemplo, 300000000. Pode-se abreviar para 3 x 108 e em Python, 3E8 ou 3e8.
* **Valores booleanos:** são os dois objetos constantes True e False usado para representar valores de verdade (em contextos numéricos 1 é True, enquanto 0 é False.Não se pode mudar nada - é preciso tomar estes símbolos tal como eles são, incluindo **case-sensitivity.**
* **None:** chamado NoneType objeto, e é utilizado para representar **a ausência de um valor.**

**Representação de números negativos:** codificados adicionando um sinal de menos. Ex: -2

**Representação de valor octal:** 0O ou 0o prefixo (zero-o). Isto significa que o número deve conter apenas dígitos retirados do intervalo [0-7]. 0o123 é um número octal com um valor (decimal) igual a 83.

**Representação de valor hexadecimal:** 0x ou 0X (zero-x). 0x123 é um número hexadecimal com um valor (decimal) igual a 291. Ex: print(0x123)

* **OPERADORES**

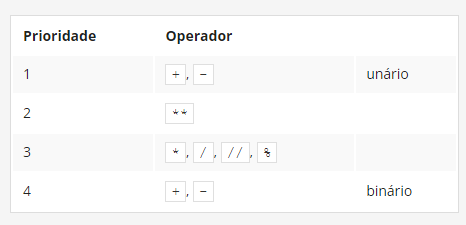
**Expressão** é uma combinação de valores (ou variáveis, operadores, chamadas a funções - em breve aprenderá sobre elas) que avalia a um valor, por exemplo, 1 + 2.

**Operadores** são símbolos especiais ou keywords capazes de operar sobre os valores e realizar operações (matemáticas), por exemplo, o \* operador multiplica dois valores: x \* y.

**Operadores aritméticos em Python:**

* + (adição)
* - (subtração),
* (multiplicação),
* / (divisão clássica - devolve sempre um float),
* % (módulo - divide o operando esquerdo pelo operando direito e devolve o resto da operação, por exemplo, 5 % 2 = 1),
* \*\* (exponenciação - operando esquerdo elevado à potência do operando direito, por exemplo, 2 \*\* 3 = 2 \* 2 \* 2 = 8),
* // (divisão por piso/inteiro - também pode ser chamada **floor division.** Devolve um número resultante da divisão, mas arredondado para baixo para o número inteiro mais próximo, por exemplo 3 // 2.0 = 1.0)
* Operador **unário** é um operador com apenas um operando, por exemplo, -1, ou +3.
* Operador **binário** é um operador com dois operandos, por exemplo, 4 + 5, ou 12 % 5.

**Hierarquia de prioridades**: alguns operadores atuam antes de outros

****

* **Subexpressões** **entre parêntesis** são sempre calculadas em primeiro lugar, por exemplo, 15 - 1 \* (5 \* (1 + 2)) = 0.
* **O operador de** **exponenciação (23 =** 2 \*\* 3.**)** utiliza ligação do lado direito, por exemplo 2 \*\* 2 \*\* 3 = 256. (Primeiro 2\*\*3 e depois 2\*\*8)
* **VARIÁVEIS**

Local nomeado, reservado para armazenar valores na memória. O Python é uma **linguagem dinamicamente dactilografada**, o que significa que não precisa de *declarar* o tipo de variáveis nela contidas. (2.1.4.3). Cada variável deve ter um nome exclusivo - um identificador que deve ser:

* uma sequência não vazia de carateres,
* começar com o underscore (\_), ou uma letra,
* não pode ser uma keyword de Python.
* O primeiro caratere pode ser seguido por underscores, letras e dígitos.
* Os identificadores em Python diferenciam entre maiúsculas e minúsculas (são case-sensitive).

**Inicializar uma variável:**

* atribuindo **o nome da variável desejada, depois o sinal de igual (=) e o valor que se pretende colocar na variável.** Ex: var = 1.
* Também pode utilizar operadores de atribuição composta (operadores de atalho) para modificar valores atribuídos a variáveis, por exemplo, var += 1, ou var /= 5 \* 2. (2.1.4.8)
* Pode atribuir novos valores a variáveis já existentes, utilizando o operador de atribuição ou um dos operadores compostos, por exemplo: (2.1.4.5)

var = 2

print(var)

var = 3

print(var)

var += 1

print(var)

**Imprimindo variável**: com a função print() pode combinar texto e variáveis utilizando o operador + para fazer output de strings e variáveis, por exemplo:

var = "007"

print("Agent " + var)

**Função round (**nome variável, número de casas decimais**)**: arredonda o resultado em output para o número de casas decimais especificadas nos parêntesis e devolve um float.

print (miles, "miles is", **round(**miles\_to\_kilometers, 3**)**, "kilometers")

Console imprime: 7.38 miles is 11.882 kilometers

* **COMENTÁRIOS**

Em Python, um comentário é um pedaço de texto que começa com # e estende-se até ao fim da linha. Os comentários podem ser importantes quando você ou outra pessoa estiver a ler o seu próprio código após algum tempo pois deixam informações adicionais que tornam os programas mais fáceis de compreender.

Se quiser colocar um comentário que abranja várias linhas, precisa de colocar # à frente de todas elas.

Sempre que possível e justificado utilize nomes de variáveis legíveis e significativos em código (**nomes self-commenting),** por exemplo, se estiver a utilizar duas variáveis para armazenar um usar nomes das variáveis length e width podem ser uma escolha melhor do que myvar1 e myvar2.

**Atalho para comentar ou descomentar rapidamente várias linhas de código:** selecione a(s) linha(s) que deseja modificar e use o seguinte atalho de teclado: **CTRL** + **/** (Windows) ou **CMD** + **/** (Mac OS).

* **FUNÇÃO INPUT()**

A função input() obtém dados do console. Quando a função input() é chamada, o fluxo do programa é interrompido, o símbolo de prompt continua a piscar até o utilizador ter introduzido um input e/ou premido a tecla *Enter*. O resultado da função é uma string. Isto significa que **não deve usá-la como um argumento de qualquer operação aritmética**

**string prompt:** parâmetro opcional que permite-lhe escrever uma mensagem antes do input do utilizador. Ex: name = input("Enter your name: ")

**Type casting:** permite fazer operações aritméticas com os valores informados no input

* a **função int()** toma um argumento (por exemplo, uma string: int(string)) e tenta convertê-lo num inteiro; se falhar, todo o programa também falhará.
* a **função float()** toma um argumento (por exemplo, uma string: float(string)) e tenta convertê-lo num float. Ex: anything = float(input("Enter a number: "))

**Operador de concatenação: +** junta-se strings umas às outras. Ex:

print(num\_1 + num\_2) # the program returns 1221

**Operador de multiplicação:** \* replica strings. Um número menor ou igual a zero produz uma **string vazia.** Ex:

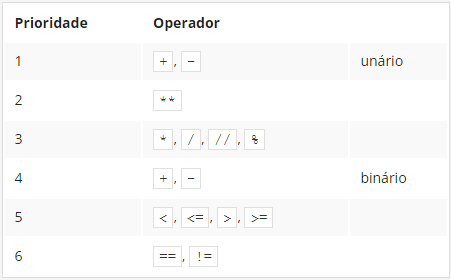
print(my\_input \* 3) # Expected output: hellohellohello

**Conversão de tipo str():** usado para **converter um número numa string. Ex:** print (str(number))

**MÓDULO 3**

**Operadores de comparação:**

* == igual a: compara os valores de dois operandos. Se forem iguais, o resultado da comparação é True. Se eles não forem iguais, o resultado da comparação é False.
* != não igual a: também compara os valores de dois operandos, mas se eles forem iguais, o resultado da comparação é False. Se eles não forem iguais, o resultado da comparação é True.
* > maior que: retorna True ou False
* >= maior que ou igual a
* <= menor que ou igual a



**Nesting: If’s aninhados**

**Cascade** (cascata): a forma de reunir as declarações subsequentes if-elif-else

* **ESTRUTURA CONDICIONAL**

**IF:** quando a condição é cumprida, *if* executa as suas declarações **apenas uma vez;**

if expressão:

comando

if expressão:

comando

if expressão:

comando

**PS**: Cada if declaração é testada separadamente

**IF- Else**

if expressão:

comando

else:

comando

if expressão:

comando

if expressão:

comando

else:

comando

Cada if é testado separadamente. O corpo de else é executado se o último if for falso.

**IF- Elif -Else**

if expressão:

comando

elif expressão2:

comando

else:

comando

Se algum dos if-elif-else contiver apenas uma instrução pode ser codificada na mesma linha, por exemplo:

if number1 > number2: larger\_number = number1

else: larger\_number = number2

**Função max() min():** encontra o maior ou menor número de todos.

largest\_number = max(number1, number2, number3)

print("The largest number is:", largest\_number)

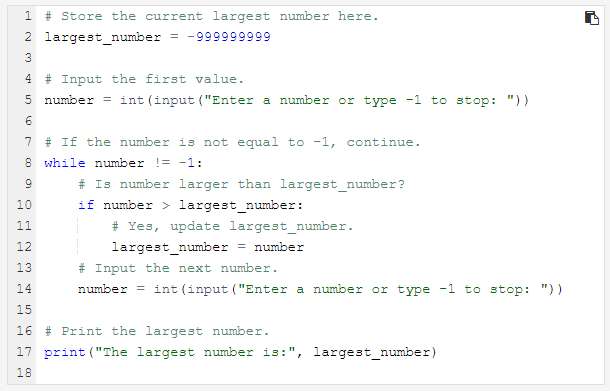
* **WHILE**

**Executa a condição e depois testa.** Executa uma declaração ou um conjunto de declarações, desde que uma condição booleana especificada seja verdadeira. Se for falsa logo que é testada pela primeira vez, o corpo não é executado nem uma vez.

while expressão condicional:

comando

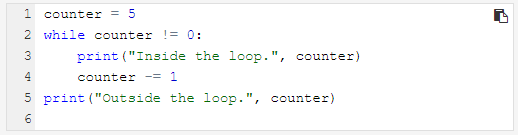
Ex:



Certas expressões podem ser simplificadas sem alterar o comportamento do programa.



**Usar uma variável de contagem para sair do looping**



**Loop infinito** (**endless loop):** é uma sequência de instruções num programa que se repete indefinidamente (loop interminável).

* **FOR**

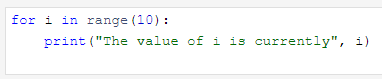
Executa um conjunto de declarações várias vezes; é usado para iterar sobre uma sequência (por exemplo, uma lista, um dicionário, um tuple, ou um conjunto) ou outros objetos que são iteráveis (por exemplo, strings).

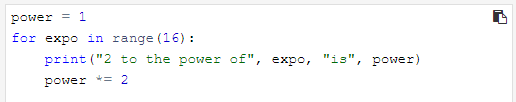
for i in range(n):

comando

* variável i após o *for* é a variável de controle do loop; conta as voltas do loop automaticamente;
* a keyword *in* introduz um elemento de sintaxe que descreve a gama de valores possíveis que estão a ser atribuídos à variável de controlo;
* a função range() é responsável por gerar todos os valores desejados da variável de controle de 0 até n-1;

Ex:

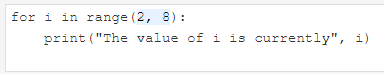




**Função range():** gera uma sequência de números. Aceita números inteiros e devolve objetos de range. A sintaxe de range() parece como se segue: range(start, stop, step), onde:

* start é um parâmetro opcional que especifica o número inicial da sequência (0 por padrão)
* stop é um parâmetro opcional que especifica o fim da sequência gerada (não está incluído),
* e step é um parâmetro opcional que especifica a diferença entre os números na sequência (1 por padrão.)

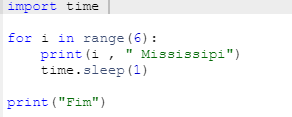
Ex: 2 argumentos que devem estar ordenados em ordem crescente.



**Ex: 3 argumentos, sendo o terceiro um incremento:**

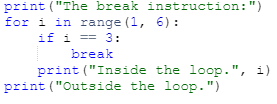


**Função sleep(): usado para suspender a execução de uma instrução por x tempo determinado no parâmetro.**

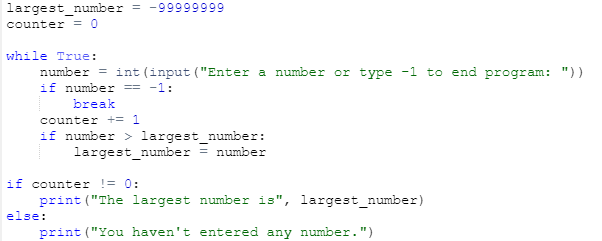


**Declarações usadas para alterar o fluxo de um loop:**

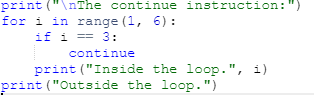
* **break** – usado quando é desnecessário continuar o loop como um todo. Sai imediatamente do loop, e termina incondicionalmente a operação do loop; o programa começa a executar a instrução mais próxima após o corpo do loop;



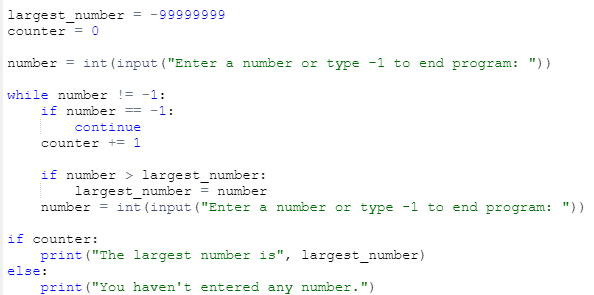
Ex:



* **continue** – usado quando é necessário iniciar a próxima volta do loop sem completar a execução da volta atual. Comporta-se como se o programa tivesse subitamente chegado ao fim do corpo; inicia-se a volta seguinte e a expressão da condição é testada imediatamente.



Ex:

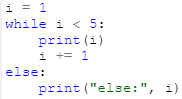


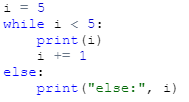
**Doces sintáticos**: funções que não melhoram o poder expressivo da linguagem, mas simplificam o trabalho do programador.

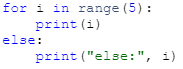
**Função Upper():** Converte todas as letras para maiúsculo.



**Else dentro dos loops:** executa-se após o loop terminar a sua execução, desde que não tenha sido terminado por break, por exemplo.

 Vai imprimir de 1 a 4 e no final a mensagem no else que é 5

Vai imprimir diretamente o else onde o valor de i é 5

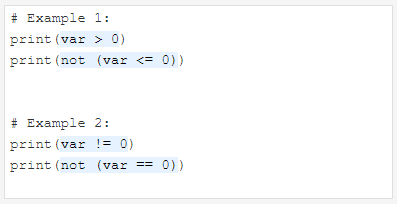
Vai imprimir de 1 a 4 e depois o else onde o valor de i é 4.

* **OPERAÇÕES LÓGiCAS**

Analisam o valor inteiro final:

* **and:** **conjunção** - **operador binário com uma prioridade que é inferior à expressa pelos operadores de comparação**. Apenas verifica se um dos valores é O (falso) ou 1 (verdadeiro)
* **or :** **disjunção - operador binário com uma prioridade inferior a**
* **not : operador unário que executa uma negação lógica** e a sua **prioridade é muito alta: a mesma que o unário**+**e**–.

**Condições** **equivalentes em pares**:



* **OPERADORES BITWISE**

Permitem **manipular bits únicos de dados.** Analisam **cada bit separadamente**. Abrangem as operações:

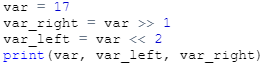
* **&** (e comercial) - conjunção bitwise;
* **|** (barra) - disjunção bitwise;
* **~** (til) - negação bitwise;
* **^** (acento circunflexo) - bitwise exclusive ou (xor).

**shifting** (deslocamento): operação relacionada a bits individuais, é aplicado apenas a valores **inteiros**.

* << o shifting de um valor um bit para a esquerda corresponde assim a multiplicá-lo por dois;
* >>o shifting um bit para a direita é o mesmo que dividí-lo por dois (o bit mais à direita é perdido).

**value << bits |value >> bits 🡪** O argumento à esquerda destes operadores é um valor inteiro cujos bits são deslocados. O argumento à direita determina o tamanho do shifting.

Ex:



17 >> 1 → 17 // 2 (**17** divido por baixo por **2 à potência de 1**) → 8 (shifting para a direita por um bit é o mesmo que a divisão inteira por dois)

17 << 2 → 17 \* 4 **17** multiplicado por **2 à potência de 2**) → 68 (shifting para a esquerda por dois bits é o mesmo que a multiplicação de inteiros por quatro)



**LISTAS**

**Lista é uma coleção dinâmica de elementos, mas cada elemento é um valor independente.** Os elementos dentro de uma lista **podem ter tipos diferentes.** Alguns deles podem ser inteiros, outros floats e outros ainda podem ser listas. **Começa e termina com colchetes** e é preenchido com valores separados por vírgulas.



**Escalares:**  variáveis que são capazes de armazenar apenas um determinado valor de cada vez.

* Atribuir um valor ao elemento 0 🡪
* Copiar o valor de um elemento para outro 🡪
* Imprimir um elemento 🡪 
* Imprimir a lista 🡪 
* Verificar o tamanho da lista 🡪 
* Remover um elemento da lista 🡪 

Um elemento com um index igual a -1 é **o último na lista**

**MÉTODO X FUNÇÃO**

Uma função não pertence a nenhum dado, é propriedade de todo o código - recebe dados, pode criar novos dados e produz um resultado (return). Um método é propriedade dos dados para os quais trabalha e não possui return. Recebe e cria dados além de ser capaz de alterar o estado de uma entidade selecionada.

A função toma um argumento, faz algo, e devolve um resultado (return).



Um método típico de invocação é geralmente semelhante a este:

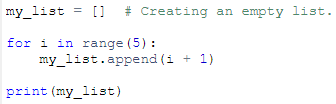


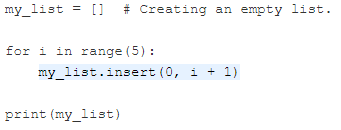
o nome do método é precedido do nome dos dados que possuem o método. Em seguida, adiciona-se um ponto, seguido do nome do método, e um par de parêntesis que encerra os argumentos.

O método comportar-se-á como uma função, mas pode fazer algo mais - pode alterar o estado interno dos dados a partir dos quais foi invocado.

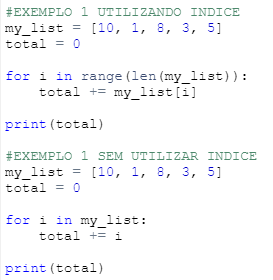
* Adicionar um valor no final da lista 🡪
* Adicionar um valor em x lugar da lista 🡪 

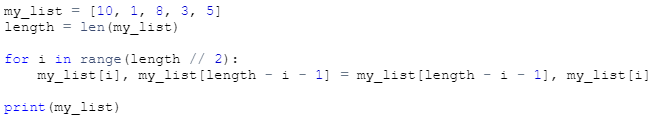
Location: mostra a localização necessária do elemento a ser inserido; todos os elementos existentes à direita do novo elemento são deslocados para a direita. Value: é o elemento a ser inserido.

(imprime uma lista de 1 a 5)

(imprime uma lista em ordem inversa)

EXEMPLO DE CÓDIGO PARA SOMAR VALORES EM UMA LISTA



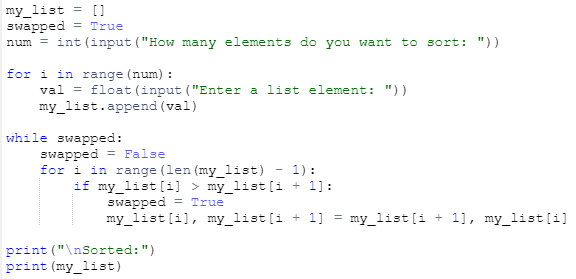
* Inverter a ordem dos elementos 🡪

**BUBLE SORT**

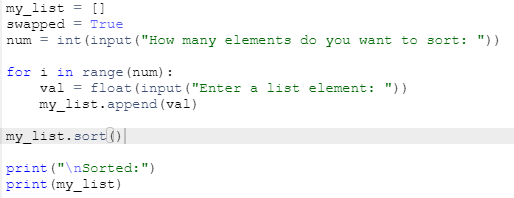
Digamos que uma lista pode ser ordenada de duas maneiras:

* crescente (ou mais precisamente - não decrescente) - se em cada par de elementos adjacentes, o primeiro elemento não for maior do que o segundo;
* decrescente (ou mais precisamente - não crescente) - se em cada par de elementos adjacentes, o primeiro elemento não for inferior ao segundo.

Compara se o elemento em evidência com seus sucessores. Se estiver na ordem desejada passa a próxima iteração, se não é feito a troca entre os elementos.



* Método integrado no Python para ordenar lista 🡪

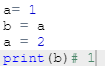


* Método para ordem decrescente da lista 🡪

**OPERAÇÔES COM LISTAS**

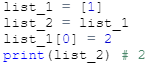
As listas (e muitas outras entidades complexas de Python) são armazenadas de formas diferentes das variáveis ordinárias (escalares):

* o nome de uma variável ordinária é o nome do seu conteúdo;

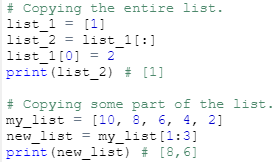


* o nome de uma lista é o nome de um local de memória onde a lista é armazenada.

A atribuição: list\_2 = list\_1 copia o nome do array, não o seu conteúdo. Com efeito, os dois nomes (list\_1 e list\_2) identificam o mesmo local na memória do computador. Modificar um deles afeta o outro, e vice-versa.



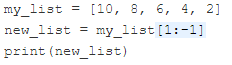
**Slice**: é um elemento da sintaxe Python que lhe permite fazer uma cópia completamente nova de uma lista ou partes de uma lista. Copia o conteúdo da lista, não o seu nome.



**faz uma nova lista (alvo), retirando elementos dos índices desde o início até**end - 1. Assim mostrará os índices iguais a 1 e 2 (mas não 3).

Uma das formas mais gerais da slice tem o seguinte aspeto: my\_list[start:end]

* start é o index do primeiro elemento incluído no slice;
* end é o index do primeiro elemento não incluído no slice.

Exemplo com índice negativo 🡪 

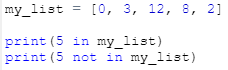
Se omitir o start no seu slice, assume-se que pretende obter um slice começando pelo elemento com index 0. Ex: my\_list[:end]. Da mesma forma, se omitir o final, mostrara do start até o último elemento. Omitindo os dois terá uma cópia completa da lista.

* A instrução del permite **apagar slices: 🡪** 
* del para apagar toda a lista 🡪

A instrução del **eliminará a**

**lista em si, não o seu conteúdo.**

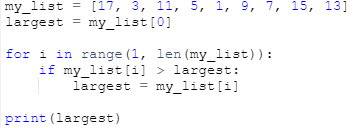
**Operador in:** verifica se um está atualmente armazenado dentro da lista - o operador devolve True neste caso.

**Operador not in:** verifica se um dado elemento está ausente numa lista - o operador devolve True neste caso.

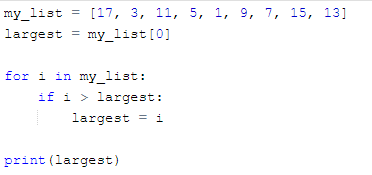
**EXEMPLOS UTILIZANDO LISTAS:**

* Encontrar o maior valor na lista.

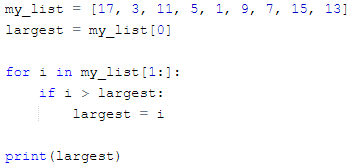
**Modo 1**



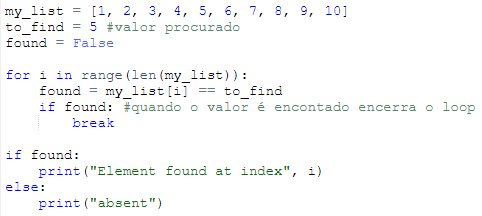
**Modo 2**



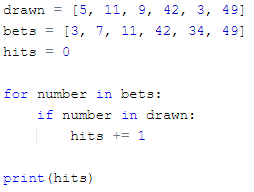
**Modo 3**



* 2- Procurar um número na lista



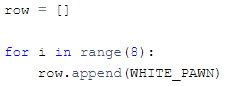
* 3- Saber quantos números acertou na mega



**LISTAS EM APLICAÇOES AVANÇADAS**

**Compreensão de lista:** permite-lhe criar novas listas a partir de listas existentes de uma forma concisa e elegante.

é igual a

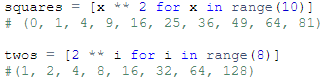
Modo comum 🡪

Modo com compreensão 🡪

A parte do código colocada dentro dos colchetes especifica:

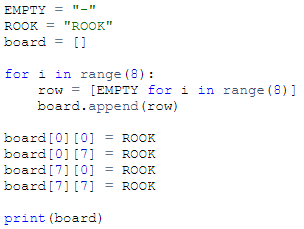
* os dados a utilizar para preencher a lista (WHITE\_PAWN)
* a cláusula que especifica quantas vezes os dados ocorrem dentro da lista (for i in range(8))

Outros exemplos:



**MATRIZ**

* a parte interior do loop cria uma linha composta por oito elementos (cada um deles igual a EMPTY) e anexa-o à lista board ;
* a parte externa repete-o oito vezes;
* no total, a lista board consiste em 64 elementos (todos iguais a EMPTY)



Representando de forma contraída:

[[EMPTY COLUNAS] LINHAS]

* Para adicionar um valor 🡪

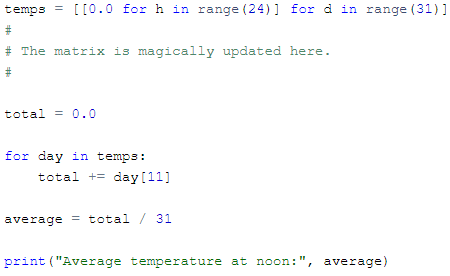
[linha][coluna]

**MATRIZ BIDIMENSIONAL**

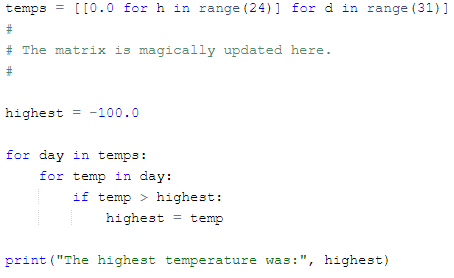
**Encontrar um elemento de uma lista bidimensional:** é preciso utilizar duas *coordenadas*:

* uma vertical (número da linha)
* e uma horizontal (número da coluna).

Exemplo: as filas registarão as leituras de hora em hora (por isso a fila terá 24 elementos ou campos) e que cada uma das filas será atribuída a um dia do mês (vamos supor que cada mês tem 31 dias, por isso precisa de 31 filas ou linhas). h é para hora, d para dia:



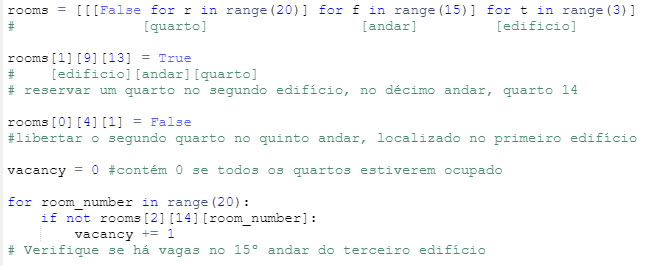
Para determinar a temperatura média mensal ao meio-dia. Some todas as 31 leituras registadas ao meio-dia e divide a soma por 31.



Encontrando a temperatura mais alta durante todo o mês

**MATRIZ TRIDIMENSIONAL**

Exemplo: Imagine um hotel. É um enorme hotel constituído por três edifícios, com 15 andares cada um. Há 20 quartos em cada andar. Para tal, é necessário um array que possa recolher e processar informações sobre os quartos ocupados/livres.



**MÓDULO 4**

**FUNÇÕES**

**Decomposição**: Dividir o código (ou mais precisamente: o problema) em peças bem isoladas, e codificar cada uma delas sob a forma de uma função. Isto simplifica consideravelmente o trabalho do programa, porque cada peça de código pode ser codificada separadamente, e testada separadamente. Se vai dividir o trabalho entre vários programadores**, decomponha o problema para permitir que o produto seja implementado como um conjunto de funções escritas em separado, embaladas em diferentes módulos.**

Em geral, as funções vêm de pelo menos três lugares:

* do próprio Python chamadas de **funções integradas**;
* dos **módulos pré-instalados** de Python
* **escritas pelo programador**

- Quando se **invoca** uma função, o Python lembra-se do local onde aconteceu e *salta* para a função invocada; o corpo da função é então executado e ao **c**hegar ao fim da função força o Python a regressar ao local diretamente após o ponto de invocação.

**- Não se deve ter uma função e uma variável com o mesmo nome.**

* Criar função 🡪

Exemplo:



* Chamar a função 🡪

Python lê o código de cima para baixo, por isso só se pode invocar uma função que já foi criada antes.

**FUNÇÃO COM PARAMETRO**

* **os parâmetros existem apenas dentro das funções em que foram definidos**, e o único lugar onde o parâmetro pode ser definido é um espaço entre um par de parêntesis na declaração def ;
* **atribuir um valor ao parâmetro é feito no momento da invocação da função**, especificando o argumento correspondente.

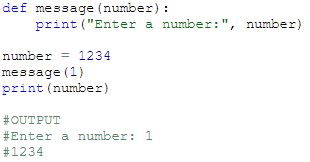


- É válido, e possível, ter uma **variável com o mesmo nome que o parâmetro de uma função**.

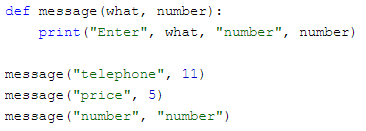
**Sombreamento**:

* parâmetro x sombreia qualquer variável do mesmo nome, mas...
* ... apenas dentro da função que define o parâmetro.

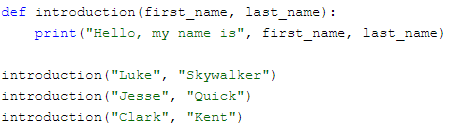
O parâmetro chamado number é uma entidade completamente diferente da variável chamada number.



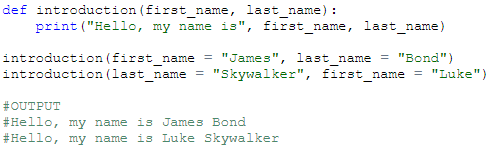
- Pode-se definir quantos parâmetros quiser para uma função



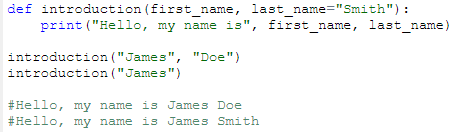
**Passagem de parâmetro posicional**: o primeiro argumento é atribuído ao primeiro parâmetro de função, e assim sucessivamente. Os argumentos passados desta forma são denominados **argumentos posicionais**. A ordem de passagem dos argumentos importa



**Keyword argument passing**: passar argumentos, onde **o significado do argumento é ditado pelo seu nome**, não pela sua posição. A posição não importa aqui - o valor de cada argumento conhece o seu destino com base no nome utilizado.

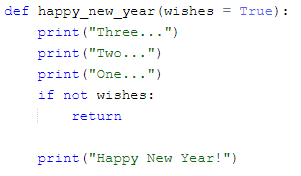


**Smith / Argumentos pré definidos:** quando os valores de um determinado parâmetro são utilizados com mais frequência do que outros podem ter os seus **valores padrão (predefinidos)**. Vaso não seja atribuído um argumento a aquele parâmetro, será usado o definido anteriormente.

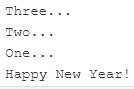


FUNÇÃO COM RETURN

## **return sem uma expressão:** Quando utilizada dentro de uma função, provoca a **terminação imediata da execução da função.** Se está não tem o objetivo de retornar um valor **utilizar a instrução**return**não é obrigatório, mas pode**-se utilizá-la para **terminar as atividades da função** antes de chegar à última linha da função.



* Se invocada sem argumentos 🡪 happy\_new\_year()



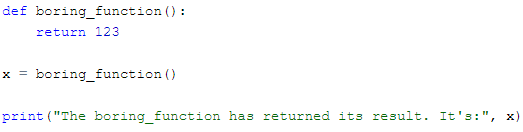
* Invocada com argumento 🡪 happy\_new\_year(False)



**return com uma expressão:** Quando utilizado assim como a opção anterior provoca a terminação imediata da execução da funçãoe além disso, a função avaliará o valor da expressão e devolverá um resultado que pode ser livremente utilizado aqui, por exemplo, para ser atribuído a uma variável ou ser ignorado e perdido.

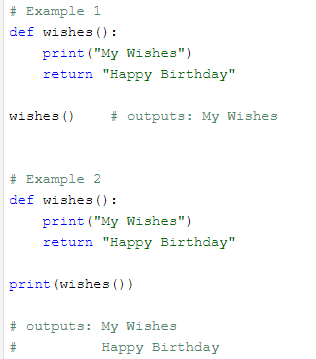


Exemplo:

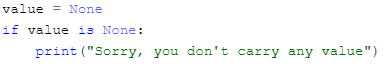


OUTPUT: 

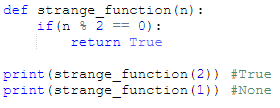
Exemplo de return de função atribuído a variável:



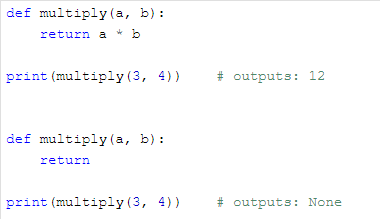
**None:** valor nulo. Pode ser utilizado quando o atribui a uma variável (ou o devolve como o resultado de uma função) e quando o compara com uma variável para diagnosticar o seu estado interno. Se uma função não devolver um determinado valor usando uma return , supõe-se que ela **devolve implicitamente**None.



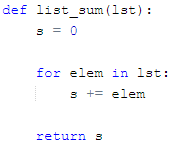
Exemplo 1:



Exemplo 2:



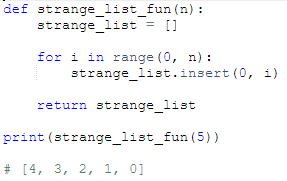
**Listas como argumento de uma função:** é aceitável para o Paython porém é necessário ter cuidado com a sintaxe.



Para invocar 🡪 

Se passar apenas um numero inteiro como argumento haverá erro já que **um único valor inteiro não deve ser iterado pelo loop for. EX:** print(list\_sum(5))

**Listas como resultado de uma função:**



**Escopo**  (por exemplo, um nome de variável) é a parte de um código onde o nome é devidamente reconhecido. Uma variável criada dentro de uma função existe apenas dentro dela.

Texto

Descrição gerada automaticamente

**Global:**  método especial Python que permite que um**a variável existe , mesmo que fora do corpo da função**.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

[Built-in Functions — Python 3.10.8 documentation](https://docs.python.org/3/library/functions.html)

[Faça login na Cisco](https://id.cisco.com/)

**EXERCICIOS**

power = 1

for expo in range(16):

print("2 to the power of", expo, "is", power)

power \*= 2

## **Cenário**

Como certamente sabe, devido a algumas razões astronómicas, os anos podem ser *bissextos* ou *comuns*. Os primeiros têm 366 dias de duração, enquanto os segundos têm 365 dias de duração.

Desde a introdução do calendário gregoriano (em 1582), a seguinte regra é utilizada para determinar o tipo de ano:

* se o número do ano não for divisível por quatro, é um *ano comum*;
* caso contrário, se o número do ano não for divisível por 100, é um *ano bissexto*;
* caso contrário, se o número do ano não for divisível por 400, é um *ano comum*;
* caso contrário, é um ano *bissexto*.

Veja o código no editor - lê apenas um número de ano, e precisa de ser completado com as instruções de implementação do teste que acabámos de descrever.

O código deve fazer output de uma de duas mensagens possíveis, que são Leap year ou Common year, dependendo do valor inserido.

Seria bom verificar se o ano introduzido cai na era Gregoriana, e faz output de um aviso caso contrário: Not within the Gregorian calendar period. Dica: use os operadores != e % .

Teste o seu código utilizando os dados por nós fornecidos.

**Resposta**

ano = int(input("Entre com o ano(AAAA): "))

if ano < 1582:

print("Ano não faz parte da era Gregoriana")

else:

if ano % 100 != 0 and ano % 4 == 0 or ano % 400 == 0:

print("Ano bissexto")

else:

print("Ano comum")

## **Cenário**

Sabe o que é o Mississippi? Bem, é o nome de um dos estados e rios dos Estados Unidos. O rio Mississippi tem cerca de 3.765 quilómetros de comprimento, o que o torna o segundo rio mais longo dos Estados Unidos (o mais longo sendo o rio Missouri). É tão longo que uma única gota de água precisa de 90 dias para percorrer toda a sua extensão!

A palavra *Mississippi* é também usada para um propósito ligeiramente diferente: *contar mississippily*.

Se não está familiarizado com a frase, estamos aqui para lhe explicar o seu significado: é usado para contar segundos.

A ideia por detrás disto é que adicionar a palavra *Mississippi* a um número ao contar segundos em voz alta faz com que soem mais perto do tempo do relógio, e por isso "um Mississippi, dois Mississippi, três Mississippi" levará aproximadamente três segundos de tempo real! É frequentemente utilizado por crianças que brincam às escondidas para garantir que o buscador faz uma contagem honesta.

A sua tarefa aqui é muito simples: escreva um programa que utilize um loop for para “contar mississippily” até cinco. Tendo contado até cinco, o programa deve imprimir para o ecrã a mensagem final "Ready or not, here I come!"

Use o esqueleto que fornecemos no editor.

**Resposta**

import time

for i in range(6):

print(i , "Mississipi")

time.sleep(1)

print('fim')

## **Cenário**

O comando continue é utilizada para saltar o bloco atual e avançar para a próxima iteração, sem executar as declarações dentro do loop.

Pode ser utilizada tanto com os loops while e for .

A sua tarefa aqui é muito especial: tem de conceber um vowel eater (comedor de vogais)! Escreva um programa que use:

* um loop for ;
* o conceito de execução condicional (*if-elif-else*)
* a declaração continue .

O seu programa deve:

* pedir ao utilizador para introduzir uma palavra;
* usar user\_word = user\_word.upper() para converter a palavra introduzida pelo utilizador em maiúsculas; falaremos sobre os chamados **métodos de strings** e o método upper() muito em breve - não se preocupe;
* usar execução condicional e a declaração continue para “comer” as seguintes vogais *A*, *E*, *I*, *O*, *U* da palavra introduzida;
* imprimir as letras não comidas para o ecrã, cada uma delas numa linha separada.

Teste o seu programa com os dados que lhe fornecemos.

**Resposta**

palavra = input("Entre com uma palavra: ").upper()

for letra in palavra:

if letra == "A" :

continue

elif letra == "E":

continue

elif letra == "I":

continue

elif letra == "O":

continue

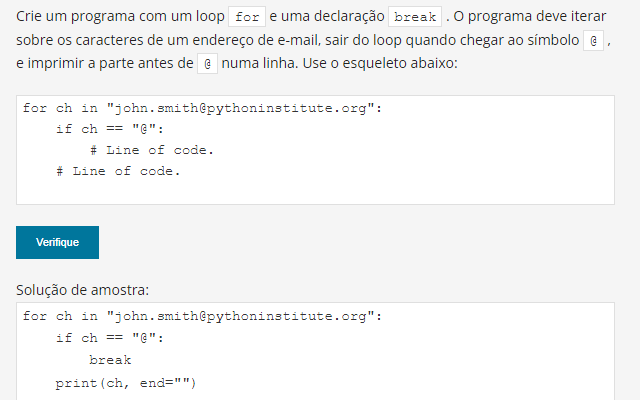
elif letra == "U":

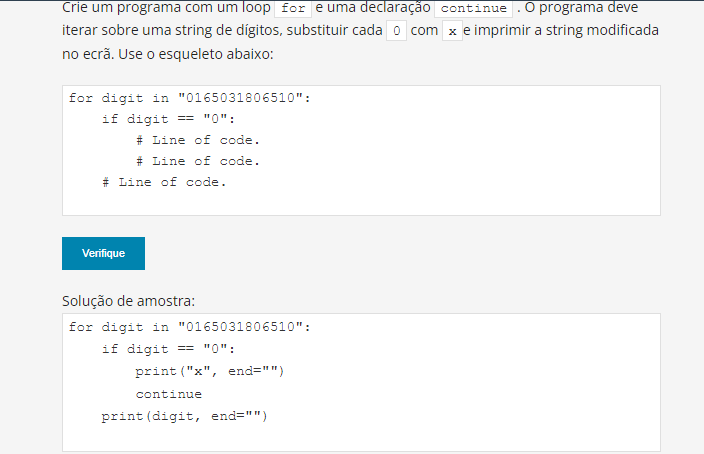
continue

else:

print(letra)

Ex





## **Cenário**

A sua tarefa aqui é ainda mais especial do que antes: deve redesenhar o comedor de vogais (feio) do laboratório anterior (3.1.2.10) e criar um comedor de vogais (bonito) melhor e mais aperfeiçoado! Escreva um programa que use:

* um loop for ;
* o conceito de execução condicional (*if-elif-else*)
* a declaração continue .

O seu programa deve:

* pedir ao utilizador para introduzir uma palavra;
* usar user\_word = user\_word.upper() para converter a palavra introduzida pelo utilizador em maiúsculas; falaremos sobre os chamados **métodos de strings** e o método upper() muito em breve - não se preocupe;
* usar execução condicional e a declaração continue para “comer” as seguintes vogais *A*, *E*, *I*, *O*, *U* da palavra introduzida;
* atribuir as letras não comidas à variável word\_without\_vowels e imprimir a variável para o ecrã.

Veja o código no editor. Criámos word\_without\_vowels e atribuimos-lhe uma string vazia. Utilize a operação de concatenação para pedir ao Python que combine as letras selecionadas numa string mais longa durante os loops subsequentes, e atribua-a à variável word\_without\_vowels .

Teste o seu programa com os dados que lhe fornecemos.

**Resposta**

palavra = input("Entre com uma palavra: ").upper()

for letra in palavra:

if letra == "A" :

continue

elif letra == "E":

continue

elif letra == "I":

continue

elif letra == "O":

continue

elif letra == "U":

continue

else:

novapalavra = letra

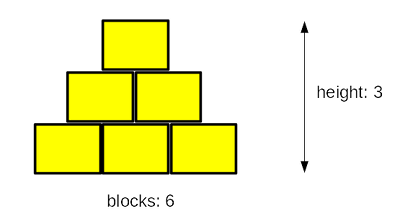
print(novapalavra, end="")

## **Cenário**

Leia esta história: um rapaz e o seu pai, um programador de computador, estão a brincar com blocos de madeira. Eles estão a construir uma pirâmide.

A sua pirâmide é um pouco estranha, pois na realidade é uma parede em forma de pirâmide - é plana. A pirâmide é empilhada de acordo com um princípio simples: cada camada inferior contém mais um bloco do que a camada superior.

A figura ilustra a regra utilizada pelos construtores:



A sua tarefa é escrever um programa que leia o número de blocos que os construtores têm, e que produza a altura da pirâmide que pode ser construída utilizando estes blocos.

Nota: a altura é medida pelo número de **camadas completamente preenchidas** - se os construtores não tiverem um número suficiente de blocos e não conseguirem completar a camada seguinte, terminam o seu trabalho imediatamente.

Teste o seu código utilizando os dados por nós fornecidos.

## Dados de teste:

Input de amostra: 6

Output esperado: The height of the pyramid: 3

Input de amostra: 20

Output esperado: The height of the pyramid: 5

## **Cenário**

Em 1937, um matemático alemão chamado Lothar Collatz formulou uma hipótese intrigante (ainda não provada) que pode ser descrita da seguinte forma:

1. tomar qualquer número inteiro não-negativo e não-nulo e nomeá-lo c0;
2. se for par, avalie um novo c0 como c0 ÷ 2;
3. caso contrário, se for ímpar, avalie um novo c0 como 3 × c0 + 1;
4. Se c0 ≠ 1, saltar para o ponto 2.

A hipótese diz que, independentemente do valor inicial de c0, irá sempre para 1.

É claro que é uma tarefa extremamente complexa utilizar um computador para provar a hipótese de qualquer número natural (pode até requerer inteligência artificial), mas pode usar o Python para verificar alguns números individuais. Talvez até encontre o que possa refutar a hipótese.

Escreva um programa que leia um número natural e execute os passos acima indicados, desde que c0 permaneça diferente de 1. Também queremos que conte os passos necessários para alcançar o objetivo. O seu código deve fazer output de todos os valores intermédios de c0, também.

Dica: a parte mais importante do problema é como transformar a ideia de Collatz num loop while - esta é a chave para o sucesso.

Teste o seu código utilizando os dados por nós fornecidos.

## Dados de teste

Input de amostra: 15

Output esperado:

46

23

70

35

106

53

160

80

40

20

10

5

16

8

4

2

1

steps = 17

## Cenário

Houve uma vez um chapéu. O chapéu não continha nenhum coelho, mas uma lista de cinco números: 1, 2, 3, 4, e 5.

A sua tarefa é:

* escrever uma linha de código que peça ao utilizador para substituir o número médio na lista por um número inteiro introduzido pelo utilizador (Passo 1)
* escrever uma linha de código que remova o último elemento da lista (Passo 2)
* escrever uma linha de código que imprima o comprimento da lista existente (Passo 3).

hat\_list = [1, 2, 3, 4, 5] # This is an existing list of numbers hidden in the hat.

print(hat\_list)

number = int(input("Entre com um valor"))

hat\_list[(len(hat\_list)-1)//2] = number

del hat\_list[-1]

print (len(hat\_list))

print(hat\_list)

CENARIO

Escreva um programa que reflita estas mudanças e lhe permita praticar com o conceito de listas. A sua tarefa é:

* passo 1: criar uma lista vazia com o nome beatles;
* passo 2: utilizar o método append() para adicionar os seguintes membros da banda à lista: John Lennon, Paul McCartney, e George Harrison;
* passo 3: utilizar o loop for e o método append() para solicitar ao utilizador que adicione os seguintes membros da banda à lista: Stu Sutcliffe, e Pete Best;
* passo 4: utilizar a instrução del para remover Stu Sutcliffe e Pete Best da lista;
* passo 5: utilizar o método insert() para adicionar Ringo Starr ao início da lista.

# step 1

beatles = []

print("Step 1:", beatles)

# step 2

beatles.append('John Lenon')

beatles.append('Paul McCoartney')

beatles.append('George Harrison')

print("Step 2:", beatles)

# step 3

for i in range(2):

novoelemento= input("Entre com um novo integrante: ")

beatles.append(novoelemento)

print("Step 3:", beatles)

# step 4

del beatles[4]

del beatles[3]

print("Step 4:", beatles)

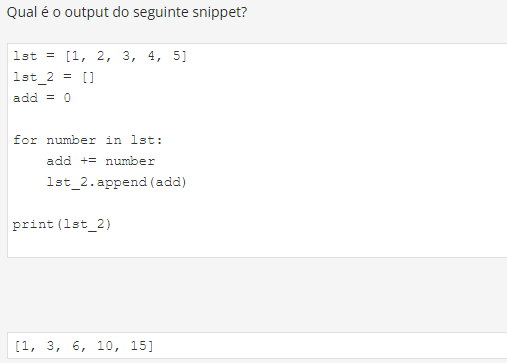
# step 5

beatles.insert(0, 'Ringo Star')

print("Step 5:", beatles)

# testing list legth

print("The Fab", len(beatles))



## Cenário

Imagine uma lista - não muito longa, não muito complicada, apenas uma simples lista contendo alguns números inteiros. Alguns desses números podem ser repetidos, e esta é a pista. Não queremos repetições. Queremos que sejam removidas.

A sua tarefa é escrever um programa que remova todas as repetições de números da lista. O objetivo é ter uma lista na qual todos os números não aparecem mais de uma vez.

Nota: suponha que a source list está codificada dentro do código - não tem de a introduzir a partir do teclado. É claro que pode melhorar o código e adicionar uma parte que pode realizar uma conversa com o utilizador e obter todos os dados a partir dele.

my\_list = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]

listadois = []

for number in my\_list:

if my\_list[number] in listadois:

continue

else:

listadois.append(my\_list[number])

print("The list with unique elements only:")

print(my\_list)

print(listadois)

## Cenário

A sua tarefa é escrever e testar uma função que leva um argumento (um ano) e devolve True se o ano for um ano *bissexto*, ou False caso contrário.

A seed da função já está semeada no código esqueleto no editor.

Nota: preparámos também um pequeno código de teste, que pode utilizar para testar a sua função.

O código utiliza duas listas - uma com os dados do teste, e a outra com os resultados esperados. O código dir-lhe-á se algum dos seus resultados é inválido.

def is\_year\_leap(year):

#

# put your code here

#

test\_data = [1900, 2000, 2016, 1987]

test\_results = [False, True, True, False]

for i in range(len(test\_data)):

yr = test\_data[i]

print(yr,"->",end="")

result = is\_year\_leap(yr)

if result == test\_results[i]:

print("OK")

else:

print("Failed")

## Cenário

A sua tarefa é escrever e testar uma função que toma dois argumentos (um ano e um mês) e devolve o número de dias para o par mês/ano dado (enquanto apenas fevereiro é sensível ao valor year , a sua função deve ser universal).

A parte inicial da função está pronta. Agora, convença a função a devolver None se os seus argumentos não fizerem sentido.

É claro que pode (e deve) usar a função previamente escrita e testada (LAB 4.3.1.6). Pode ser muito útil. Encorajamo-lo a utilizar uma lista preenchida com a duração dos meses. Pode criá-la dentro da função - este truque irá encurtar significativamente o código.

Preparámos um código de teste. Expanda-o para incluir mais casos de teste.

def is\_year\_leap(year):

#

# Your code from LAB 4.3.1.6.

#

def days\_in\_month(year, month):

#

# Write your new code here.

#

test\_years = [1900, 2000, 2016, 1987]

test\_months = [2, 2, 1, 11]

test\_results = [28, 29, 31, 30]

for i in range(len(test\_years)):

yr = test\_years[i]

mo = test\_months[i]

print(yr, mo, "->", end="")

result = days\_in\_month(yr, mo)

if result == test\_results[i]:

print("OK")

else:

print("Failed")

## Cenário

A sua tarefa consiste em escrever e testar uma função que toma três argumentos (um ano, um mês e um dia do mês) e devolve o dia correspondente do ano, ou devolve None se algum dos argumentos for inválido.

Use as funções previamente escritas e testadas. Adicione alguns casos de teste ao código. Este teste é apenas um começo.

def is\_year\_leap(year):

#

# Your code from LAB 4.3.1.6.

#

def days\_in\_month(year, month):

#

# Your code from LAB 4.3.1.7.

#

def day\_of\_year(year, month, day):

#

# Write your new code here.

#

print(day\_of\_year(2000, 12, 31))

## Cenário

*Um número natural é****primo****se for maior que 1 e não tiver divisores além de 1 e dele próprio.*

Complicado? De modo algum. Por exemplo, 8 não é um número primo, uma vez que se pode dividi-lo por 2 e 4 (não podemos utilizar divisores iguais a 1 e 8, uma vez que a definição o proíbe).

Por outro lado, 7 é um número primo, uma vez que não conseguimos encontrar quaisquer divisores legais para ele.

A sua tarefa é escrever uma função para verificar se um número é primo ou não.

A função:

* é chamada is\_prime;
* toma um argumento (o valor a verificar)
* devolve True se o argumento for um número primo, e False caso contrário.

Dica: tente dividir o argumento por todos os valores subsequentes (começando por 2) e verifique os restos - se for zero, o seu número não pode ser um primo; pense cuidadosamente sobre quando deve parar o processo.

Se precisar de conhecer a raiz quadrada de qualquer valor, pode utilizar o operador \*\* . Lembre-se: a raiz quadrada de x é o mesmo que x0.5

Complete o código no editor.

Execute o seu código e verifique se o seu output é o mesmo que o nosso.

## Output esperado

2 3 5 7 11 13 17 19

def is\_prime(num):

#

# Write your code here.

#

for i in range(1, 20):

if is\_prime(i + 1):

print(i + 1, end=" ")

print()

## Cenário

O consumo de combustível de um carro pode ser expresso de várias maneiras. Por exemplo, na Europa, é mostrado como a quantidade de combustível consumido por 100 quilómetros.

Nos EUA, é mostrado como o número de milhas percorridas por um carro utilizando um galão de combustível.

A sua tarefa é escrever um par de funções convertendo l/100 km em mpg, e vice-versa.

As funções:

* são nomeadas liters\_100km\_to\_miles\_gallon e miles\_gallon\_to\_liters\_100km respetivamente;
* tome um argumento (o valor correspondente aos seus nomes)

Complete o código no editor.

Execute o seu código e verifique se o seu output é o mesmo que o nosso.

Aqui estão algumas informações para o ajudar:

* 1 milha americana = 1609,344 metros;
* 1 galão americano = 3,785411784 litros.

## Output esperado

60.31143162393162

31.36194444444444

23.52145833333333

3.9007393587617467

7.490910297239916

10.009131205673757

def liters\_100km\_to\_miles\_gallon(liters):

#

# Write your code here.

#

def miles\_gallon\_to\_liters\_100km(miles):

#

# Write your code here

#

print(liters\_100km\_to\_miles\_gallon(3.9))

print(liters\_100km\_to\_miles\_gallon(7.5))

print(liters\_100km\_to\_miles\_gallon(10.))

print(miles\_gallon\_to\_liters\_100km(60.3))

print(miles\_gallon\_to\_liters\_100km(31.4))

print(miles\_gallon\_to\_liters\_100km(23.5))

