**APRENDENDO PYTHON**

Pode usar ‘ ’ ou “ ” para mensagens. Número não precisa.

Toda função tem que ter ( ).

**- Print():** Escreve na tela.

**- Input(‘texto’):** Recebe/lê informação externa e acrescenta à variável.

**- + ou ,:** Concatenação.

**- Format():**

Substitui as {} (nesse caso chamados de máscaras) dentro de uma string pelo conteúdo/variável que estiver dentro dos parênteses do comando. Pode colocar números como {0}, {1} e {2}, para identificar qual variável é. Pode-se fazer alinhamentos com esse tipo de comando, basta colocar em quantos caracteres quer que a string seja escrita. Ex.: print(‘Prazer em te conhecer, {:20}!’.format(nome)) == Prazer em te conhecer, nome !. Ou: print(‘Prazer em te conhecer {:>20}!’.format(nome)) == Prazer em te conhecer, nome!. Ele vai alinhar o nome à direita em 20 espaços. Se fosse um sinal de menor (<) ele iria alinhar à esquerda. Se fosse um sinal de circunflexo (^), o nome ficaria centralizado em 20 espaços. Se colocasse igual = e circunflexo (^) o nome ficaria centralizado em 20 espaços e com o sinal de igual em volta. Ex.: {:=^20} == ‘========Ana========’.

Se colocar 2 variáveis a serem somadas dentro do () do format, ele irá mostrar o valor somado no lugar da máscara, sem precisar de outra variável. Só irá criar outra variável quando for usar esse valor para outra coisa mais para frente no programa.

Em números flutuantes dá para colocar quantas casas quer que seja exibida, basta colocar um sinal de dois pontos, um ponto, a quantidade de casas e a letra “f” para número flutuante. Ex.: {:.3f}

Pode mudar o formato desse comando colocando um f antes das aspas simples ou compostas e colocando a variável dentro das chaves ({}) depois. Para usar a função do parágrafo anterior, basta colocar os dois pontos ( : ) depois da variável.

**- Type()**: Mostra qual o tipo primitivo de uma variável se estiver dentro de um print.

**- Isnumeric():** Se colocado depois de uma variável diz true se o valor digitado no input for um número e false se não for.

**- Isalpha():** Se colocado depois de uma variável diz true se o valor digitado no input for uma string alfabética e false se não for.

**- Isalnum():** Se colocado depois de uma variável diz true se o valor digitado no input for alfanumérico, ou seja, um número e uma letra, e false se não for.

**- Isupper():** Se uma string estiver toda em letra maiúscula ele dirá true, se não, dirá false.

**- End=’’:** Quando colocado dentro do print e com espaço vazio dentro das ‘ ’ ele irá juntar o fiz desse print com o começo do próximo, mantendo os dois na mesma linha. Se colocar alguma coisa dentro ele coloca entre as duas linhas e segue sua função.

**- \n:** Faz a quebra da linha em qualquer lugar de um print, não tendo a necessidade de criar outro abaixo.

**- ‘’’ texto ‘’’:** Coloca-se as três aspas antes e depois de um texto/comando/programa quando se quer transformar mais de uma linha em comentário. Pode usar essa técnica em um print também, dessa forma, ele irá printar várias linhas de uma vez sem a necessidade de criar um print para cada linha do texto desejado, facilitando assim a criação de menus dentre outros esquemas de interatividade.

**TIPOS PRIMITIVOS:**

**- Int():** Transforma uma variável string do input em número inteiro. Ex.: 7, -4, 0...

**- Str():** Transforma uma variável numérica em uma string. Ex.: Nomes, ‘CPF’, ’RG’, Mês, ‘’(string vazia)...

**- Float():** Transforma uma variável em um número real. Ex.: 4.5, 0.076, -15.223...

**- Bool():** Valores de boolean ou booleanos. Ex.: True, False (Primeira letra sempre maiúscula).

**OPERADORES**

**OPERAÇÕES ARITMÉTICAS:**

1. **+:** Adição

2. **-:** Subtração

3. **\*:** Multiplicação

4. **/:** Divisão

5. **\*\*:** Potência/Exponenciação

6. **//:** Divisão inteira

7. **%:** Módulo/Resto da Divisão.

8. **==:** Igual.

9. pow(3,2): Eleva a potência também. Nesse caso é 3\*\*2

Criar a raiz quadrada de um número é o mesmo que fazer a potência dele por ½(meio). Ex: 25\*\*(1/2) == 5.0. A raiz cubica é elevar o número a 1/3. Ex.: 127\*\*(1/3) == 5.0265.

Se colocar uma string \* um número, ela irá se repetir a quantidade de vezes. Ex.: ‘Oi’\*5 == ‘OiOiOiOiOi’.

**Ordem de precedência:**

1. ().
2. \*\*.
3. \*, /, //, %. Se tiver vários na mesma expressão, faz o que vier primeiro da esquerda para a direita.
4. +, -.

**OPERAÇÕES LÓGICAS:**

**- =:** Recebe.

**- ==:** Igual.

**- !=:** Diferente.

**- >:** Maior que.

**- >=:** Maior ou igual à.

**- <:** Menor que.

**- <=:** Menor ou igual à.

Todo objeto possui características e realiza funcionalidades. Tem atributos e métodos. Todo objeto string tem esses métodos. Os métodos são todos os “is..()” etc.

**MÓDULOS:**

Existem os instaláveis e os que já acompanham de padrão.

**- Import:** É o comando que importa as bibliotecas/Módulos inteiros, basta colocar o nome dela na frente do comando. **Ex.: import math**

**- From:** Comando usado para fazer uma importação única de bibliotecas. Basta colocar o nome da biblioteca depois, seguido de ”import” e o que quer importar logo em seguida. Ex.: from math import sqrt. Você pode digitar a função que quer importar ou apertar CTRL + SPACE para ele exibir todas as funções para que você possa escolher. Posso importar quantas e quais funções quiser, basta colocar uma vírgula após cada função escolhida até a última. Se colocar a palavra **AS** depoisitem que quer importar ele muda de nome, por exemplo: from math import sqrt as r. A partir de agr, o comando sqrt foi abreviado para r. Para fazer o uso da função nesse caso, não se usa a referência “math”, coloca-se apenas a função que foi importada e a variável do número dentro dos parênteses. Ex.: trunc(n), ao invés de math.trunc(n).

**Bibliotecas:**

- **Math:** Traz funcionalidades matemáticas extras. Antes de qualquer comando colocar “math.”.

Comandos e funcionalidades**:**

1. Ceil: Faz o arredondamento para cima.
2. Floor: Faz o arredondamento para baixo
3. Trunc: Trunca o número. Elimina da vírgula para frente sem fazer arredondamentos.
4. Pow: Potência. Funciona igual ao \*\*.
5. Sqrt: Calcula raiz².
6. Factorial: Calcula fatorial.

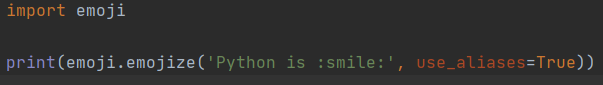
Precisa colocar () depois de todos.

**- Random:** Gera números aleatórios.

Comandos e funcionalidade:

1. Random(): Gera números floats aleatórios de 0 até 1.
2. Randint( , ): Gera números inteiros aleatórios. Coloca-se dentro dos () de qual número até qual número quer que ele gere, respectivamente.
3. Choice(): escolhe um item/variável aleatório presente dentro de uma **lista**, cujo nome é colocado dentro dos ()
4. Shuffle(): Embaralha os itens/variáveis de uma lista, cujo será determinada sendo colocada dentro dos ().

**- Emoji:** Cria emojis no programa. Link para todos: <https://www.webfx.com/tools/emoji-cheat-sheet/>

Para dar o print segue a estrutura: 

**- Pygames:** Feita para desenvolvimento de jogos. Precisa ser iniciada com o comando

1. Pygames.init(): Inicia a biblioteca para que ela funcione.

2. Pygames.mixer.init(): Inicia o mixer da biblioteca, usado para reprodução e controle de áudio.

3. Pygames.mixer.music.load(‘nome da música’): Carrega o áudio que foi copiado e colado para dentro do projeto com extensão mp3.

4. Pygames.mixer.music.play(): Inicia a música.

5. While(pygames.mixer.music.get\_busy()): pass: É uma estrutura de repetição que faz com que o programa rode até o áudio parar de ser tocado.

**Lista:**

Sempre recebe [], e dentro dele, coloca-se as variáveis que irão fazer parte dessa lista. Ex.: lista = [n1, n2, n3, n4]

**Manipulando Cadeias de Textos**

Deve estar dentro de aspas simples (‘ ’) ou duplas (“ ”). Quando se atribui uma str à uma variável, o python vai colocar todas as letras, incluindo espaços, dentro de micro espaços nessa memória. Cada micro espaço é marcado por um número que vai de 0 até o número necessário para chegar ao final da str. Existem várias técnicas de manipulação de str, algumas delas são:

**- Fatiamento:** Pega pedaços da str. Se mandar printar o número 9 de uma str, ele irá mostrar somente o que existe naquele micro espaço e nada mais. Ex.: eu faço a seguinte atribuição: frase = ‘Curso em Video Python’. Cada letra e espaço será referenciado à um número. Se eu mandar print(frase[9]) (o número referente ao fatiamento sempre tem que estar entre []), ele irá exibir a letra “V” na tela. O python faz diferenciação de maiúsculas e minúsculas, uma letra “V” e uma letra “v” são coisas diferente.



Se colocar “ : ” e outro número nos colchetes, ele irá pegar do primeiro número digitado até o anterior ao último e exibir na tela. Ele sempre irá pegar um número a menos do fim. Ex.: print(frase[9:13]).



Se colocar mais um “ : ” e outro número depois, dentro dos colchetes, ele irá pular a quantidade de casas referente ao número. Ex: print(frase[9:21:2]). Ele irá da letra cujo índice é 9, até a letra 20 (já que ele pega uma a menos do que é especificado) pulando de 2 em 2 casas. Ou seja, ele vai pegar as letras “V”, ”d”, “o”, “P”, “t”, “o”.



Se colocar apenas “ : “ e um número depois do colchete, ele irá pegar da letra 0 até a letra determinada depois dos “ : “, ou seja, quando eu omito o número antes dos “ : “, ele considera que é para pegar do início. Ex.: print(frase[:5]). Ele iria mostrar a palavra “curso” que vai da letra 0 até a 4.

O contrário também é verdade, ou seja, se eu coloco um número e depois os “ : “, eu mando o python mostrar tudo o que tem da letra cujo índice eu determinei, até o final da str. Ex.: print(frase[15:]). Ele irá mostrar da letra 15 até a 20, exibindo a palavra “Python”.

Se colocar o valor de início, pular o de final e colocar o de pulos, ele irá da letra cujo índice foi esclarecido e ir até o final da str pulando a quantidade de casas especificadas e exibindo apenas as letras equivalentes aos pulos. Para pular o número da casa final para que o fatiamento seja feito até o fim da str, basta colocar um “ : “ seguido do outro. Ex.: print(frase[9::3]). Dessa maneira, ele irá da casa 9 até o final pulando de 3 em 3 casas, mostrando apenas as letras “V”, ”e”, “P”, “h”.



**- Análise:** Mostra algumas informações da str, como: qual o tamanho dela, com que letra começa, com que letra termina, qual é a primeira palavra inteira...

Usando o comando **leng(**frase**)** você consegue descobrir qual é o tamanho/comprimento da str. Se mandar mostrar na tela, ele dirá que tem 21.



Usando o comando frase.**count**(‘o’) ele irá contar quantas letras “o” minúsculas aparecem na frase. As maiúsculas não são contadas. E se mandar mostrar na tela, ele dirá que tem 3.

****

Se a letra que você pediu para ele contar for seguida de 2 números separados por vírgulas, ele irá fazer a contagem com fatiamento. Ex.: frase,count(‘o’, 0, 13). Ele irá contar quantas letras ‘o’ existem do micro espaço 0 até o 13, já que ele sempre conta até uma a menos do que foi exibido como final. Se mandar printar, ele irá mostrar o valor 1.



Usando o comando frase.**find(**‘deo’**)** ele irá dizer quantas vezes ele encontrou essa sequência de caracteres dentro da variável, mostrando onde essa sequência se inicia. Ex.:

****

Nesse caso ele mostraria o valor 11, que é onde a sequência se inicia. Se você pede para ele achar uma string que não existe na sua variável, ele te retorna o valor -1.

Uma variação da função acima é o **in**, que não é uma funcionalidade, mas sim um operador. Pode-se pedir para o python procurar uma palavra ou sequência de letras dentro da variável a usando. Ex.: ‘curso’ in frase. Porém, diferentemente do find, ele irá te retornar True caso tenha essa palavra, ou false, caso não tenha. Ele não irá dizer onde a sequência se inicia, apenas se tem ou não.

**- Transformação:** Como via de regra, uma lista de str é imutável, ou seja, não se pode mexer nela, mas pode alterar utilizando métodos.

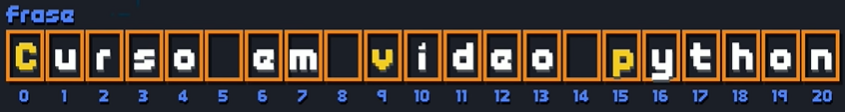
Se utilizar o comando frase.**replace(**‘Python’, ‘Android’), ele irá procurar a palavra python na str e alterar ela para android. Se contar as letras, android tem uma a mais, ou seja, teria que acrescentar mais 1 micro espaço, mas não tem problema, pois o próprio python faz isso. Ele não substitui a palavra diretamente, mas sim de uma forma secundária.

Se utilizar o comando frase.**upper()**, ele irá transformar todas as letras em maiúsculas. Isso vale para o comando frase.**lower()**, que transforma todas em minúsculas.





Se utilizar o comando frase.**captalize()**, ele irá jogar todos os caracteres para minúsculo e a primeira letra ele jogará para maiúscula.

****

De forma parecida, o comando frase.**title()** coloca as iniciais em maiúsculas, porém, diferentemente do anterior, ele irá fazer uma análise da str e ver quantas palavras tem e a partir dessa informação, ele pode jogar a inicial de todas as palavras para maiúsculas. Ele faz isso quebrando as palavras por conta dos espaços entre as letras que ele encontra na variável.

****

Se utilizar o comando frase.**strip()** ele irá remover espaços inúteis excedentes dentro da string. Ex.: Tem uma caixa de texto em determinado site e alguém clica na caixa de texto, e ao invés de começar a digitar direto, ele aperta espaço algumas vezes para ver se está funcionando para só aí começar a digitar. Esse comando serve para remover esses espaços desnecessários no início e final de cadeia de caracteres, os existentes no meio delas permanece.





De forma similar, existe o comando frase.**rstrip()**, ele irá remover todos os espaços apenas do lado direito da str. \*Tem várias funcionalidades que tem uma variante com r nela, usadas para poder tratar coisas pela direita\*.



De forma análoga existe o comando frase.**lstrip()**, ele irá remover todos os espaços apenas do lado esquerdo da str.



**- Divisão:** Serve para dividir uma str em várias partes.

Se utilizar o comando frase.**split()**, ele irá dividir a sua str onde tiver espaços. Ele determina que ali é uma separação de palavras e divide a str nestes pontos. Cada palavra recebe indexação nova começando a partir do 0 e indo até o final da palavra. Cada uma é colocada dentro de uma outra lista. Ele irá gerar uma lista com todas as palavras de uma cadeia de caracteres, cujo também tem numerações. Da para reconfigurar a localização de divisão do split.



**- Junção:** Faz a junção de palavras divididas.

Se utilizar o comando ‘-‘.**join(**frase**)**, ele irá juntar todos os elementos contidos em “frase” e vai utilizar aquele traço dentro das ‘ ‘ para separar as palavras. Se quiser que sejam espaços ao invés de traços, basta substituir por traços em branco.



**ESTRUTURAS DE CONDIÇÃO:**

**- If x (variável) == (qualquer operador lógico) y (variável, número...): :** Cria uma condição. Na linha de baixo, depois dos dois pontos coloca o que acontece se a condição for cumprida, seja um print, conta, whatever. Pode-se colocar um if dentro de outro seguido sem problemas para poder criar outras características, como no D016 que fiz.

**- Else: :** Na linha de baixo, coloca o que acontece caso a condição acima não seja cumprida (não precisa necessariamente estar no programa, apenas se quer que algo específico aconteça). Pode acompanhar um IF na linha de baixo, a fim de criar múltiplas condições. Ex.: Desafio 16 que fiz.

Dá para simplificar essas duas condições as colocando na mesma linha, se o programa for curto. Ex.:



**Condições Aninhadas:**

**- Else if** ou **elif(estrutura simplificada): :** Usado para criar condições dentro de outras condições com o intuito de criar diversos caminhos a se seguir com o programa. Ex.: Meu exercício ex003 no projeto Python Exercícios. Dá para criar uma lista de itens usando a operação “**in**” para fazer com execute o programa. Ex.: elif nome in ‘ana cláudia jéssica juliana’:

**COLOCANDO CORES NO PYTHON:**

Existem módulos para isso, mas inicialmente nós iremos usar o padrão ASNI escape sequence: Tudo começa com contra barra e um código.

**\033[m]:** É o código que se dá a referência as cores. Entre o colchete e o “m” é onde você coloca 3 diferentes códigos, sendo eles:

- O do estilo da fonte da letra, se ela vai estar sublinhada, em negrito...;

- A cor do texto;

- A cor do background, do fundo do texto; tudo separado por ponto em vírgula.

**Ex.:** **\033[**style; tex; back**m**. Você pode colocar em qualquer ordem, esse foi apenas um exemplo. Isso acontece porque existem diferenciações numéricas entre esses códigos.

**Ex.: \033[0; 33; 44m**. Esse código se refere ao estilo 0, ou seja, a letra vai ser a padrão do terminal, o texto vai ter a cor referente ao número 33 e o fundo dela ao 44. São todos opcionais, você pode usar um e não usar outro, vai da sua preferência.

**Códigos para estilo:**

- **0:** Sem estilo nenhum, padrão do terminal;

**- 1:** Coloca em negrito;

**-4:** Coloca em sublinhado;

**- 7:** inverte as configurações. O que colocou para fundo ele vai para letra e vice versa. É a forma negativa.

Existem outros, mas esses são os que funcionam melhor para o terminal do python.

**Códigos para a cor do texto:**

**- 30:** Branco

**- 31:** Vermelho

**- 32:** Verde

**- 33:** Amarelo

**- 34:** Azul

**- 35:** Roxo

**- 36:** Ciano

**- 37:** Cinza

Para acrescentar outros tipos de cores você precisa adicionar alguma biblioteca. Apesar de não ter a cor preta, nós conseguimos gerar ela.

**Códigos para a cor do fundo:**

**- 40:** Branco

**- 41:** Vermelho

**- 42:** Verde

**- 43:** Amarelo

**- 44:** Azul

**- 45:** Roxo

**- 46:** Ciano

**- 47:** Cinza

Para retirar as formatações é só colocar **\033[m** sem nenhuma configuração. Esses códigos são colocados dentro dos parênteses do print. Para manter uma configuração de cor somente em uma frase e não deixar que ela continue para o resto da linha é só colocar a configuração antes do texto e o código que reseta as configurações para padrão no final do texto. Ex.: print(‘\033[1;31;43mOlá, Mundo!\033[m’). Esse print faz com que o texto fique em negrito, com a letra vermelha e o fundo amarelo.

Pode-se criar uma variável e deixar programado diversas cores e estilos. Ex.:



Na hora de colocar as cores na formatação, coloca a variável e entre colchetes a formatação que quer. Ex.: cores [‘azul’].

**ESTRUTURAS DE REPETIÇÃO:**

**- For x(variável contadora) in range(n, n)::** Substitui o x pela variável contadora e os n’s colocando de qual número até qual ele irá repetir o que estiver depois dos dois pontos ( : ).

Ele não considera o último n, por exemplo, se colocar pra ele repetir algo de 1 até 6 (1, 6) ele vai repetir 5 vezes, portanto, é interessante colocar pra ele repetir de 0 até 6 (0, 6), porque aí, nesse caso, ele repetirá 6 vezes.

Depois do segundo n você pode colocar um outro número para especificar o tipo de iteração (não é **interação**), como por exemplo, colocar o número 2. Ex.: for c in range(0, 7, 2): print(c). Ele vai contar de 0 até 6 pulando de 2 em 2, exibindo os números 0, 2, 4 e 6.

Outro exemplo seria colocar o -1 para dizer para ele retirar 1 a cada contagem e contar regressivamente. Ex.: for c in range(6, 0, -1): print(c). Ele irá contar de 6 até 1.

Pode-se usar estruturas de controle de condicional dentro dos laços também. O contrário também é verdade.

A variável contadora (colocando a letra c por exemplo) não precisa ter sido declarada anteriormente, pode coloca-la direto no for que irá funcionar.

**\*\* Você pode utilizar qualquer estrutura aninha dentro de outra sem restrições. Se atender a sua necessidade e funcionar, está valendo! \*\***

**- While():**