Anotações

1. COMANDO PRINT

O resultado do Comando **print** escreve/publica. Ele recebe os argumentos separados por vírgula.

```
print('Olá Mundo!')
msg = 'Ola Mundo!'
print(msg)
```

```
Olá Mundo!
Ola Mundo!
```

Atenção ao uso das aspas, elas devem ser designadas para mostrar ao programa o que será **string**

```
print(2+2)
print('2'+'2')
```

```
4
22
```

2. FUNÇÃO DO INPUT

espera uma entrada do usuário pelo terminal. O programa lê SEMPRE essa entrada como uma **string**, se a entrada esperada for um número ela deve ser convertida usando-se as funções de conversão **int() ou float().**

```
nome = input('Digite seu nome: ')
print('Oi ',nome,' Tudo bom?! Só alegria?')
```

```
Digite seu nome: Romulo
Oi  Romulo  Tudo bom?! Só alegria?
```

```
# Programa para calcular a media de um aluno
print('Programa para calcular a media de um aluno')
nome = input('Entre com o nome do aluno: ')
nota1 = float(input("Entre com a primeira nota: "))
nota2 = float(input("Entre com a segunda nota: "))
media = (nota1 + nota2)/2
print(nome, 'teve media igual a:', media)
```

```
Entre com o nome do aluno: Romulo
Entre com a primeira nota: 9
Entre com a segunda nota: 6
Romulo teve media igual a: 7.5
```

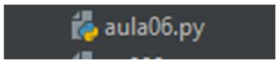
3. TIPOS PRIMITIVOS

Int == 7, -4, 0, 9875

float == 4.5, 0.076, -15.223, 7.0

bool == True, False

str == ‘Olá’, ‘7.5’, ‘’



4. FUNÇÃO DO FORMAT

Serve para criar uma **string** que contem campos entre chaves a serem substituídos pelos argumentos de **format.**

```
nome = input('Digite seu nome: ')
print('Oi',nome,'Tudo bom?! só alegria?!')
print('Oi {} Tudo bom?! só alegria?!'.format(nome))
print(f'Oi {nome} Tudo bom?! Só alegria?!')
```

3 formas de fazer o exercício 002 porem as ultimas duas são utilizando o **format,** a última é uma forma mais atualizada que deixa o código mais simples.

Dentro dos campos a serem substituídos pode-se especificar formatações numéricas ou para strings. < esquerda, > direita, ^ centro

```
n = 'Romulo'
print(f'!{n:>10}!')
print(f'!{n:<10}!')
print(f'!{n:^10}!')
print(f'!{n:=^10}!')
```

```
!    Romulo!
!Romulo    !
!  Romulo  !
!==Romulo==!
```

No exemplo “n” é o objeto,”**:**” a função de formatação, logo após é especifada o tipo de formatação, {n:>10} por exemplo quer dizer: colocar n alinhado a direita em 10 caracteres, se os caracteres não forem especificados eles serão espaços, mas podem ser de qualquer tipo, como no ultimo exemplo {n:=^10}.

Também é utilizado o format nesse exemplo:

```
d = 5 / 3
print(f'A divisão é {d}!')
print(f'A divisão é {d:.3f}!')
```

```
A divisão é 1.6666666666666667!
A divisão é 1.667!
```

No exemplo a divisão inteira retorna um **valor float** com muitas casas após a vírgula, a função format, formata o resultado do print designando quantas casas se deseja observar após a virgula. “**:**” com função de formatação, “**.**” Declara que a formatação para números flutuantes (float), “3” indica a quantidade de casas decimais desejadas, “f” indica que essas casas decimais são um valor flutuante.

```
9   print('Testando ferramentas de quebra de linha.', end='000')
10  print('Testando ferramentas de quebra de linha.')
Testando ferramentas de quebra de linha.000Testando ferramentas de quebra de linha.
```

No exemplo acima foi utilizado o “end = “. A linha resultado da função print, não foi descontinuada, e logo após foi inserido “000” e já mostra a próxima linha.

```
11  print('Testando \nferramentas\nde \nquebra de linha.')
Testando
ferramentas
de
quebra de linha.
```

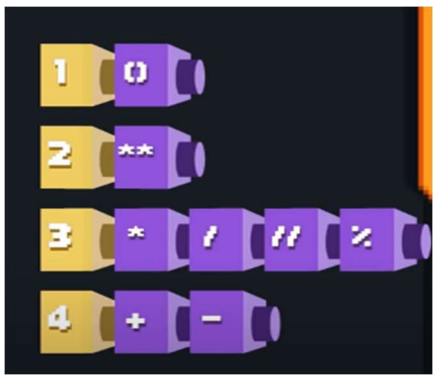
A linha resultado da função print, foi formatada para que tenha uma “quebra de linha” em algumas partes utilizando o \n (contrabarra + n).

5. OPERADORES ARITMÉTICOS

Os **operadores** no python precisam de **operandos** para serem executados, no caso dos **operadores aritméticos** os operandos não podem ser str devem ser numéricos e precisam de 2 operandos.

+ == adição (5 + 2 == 7)
- == subtração (5 - 2 == 3)
* == multiplicação (5 * 2 == 10)
/ == divisão (5 / 2 == 2,5)
** == potência (5 ** 2 == 25)
// == divisão inteira (5 // 2 == 2)
% == resto da divisão (5 % 2 == 1)

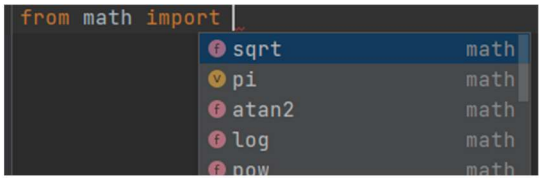
Ordem de precedência



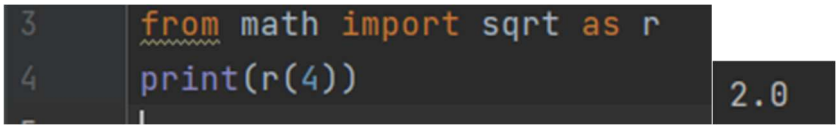
6. MÓDULOS

Os comandos **import** e **from/import** no Python servem para carregar bibliotecas de funções e utilizar vários recursos adicionais nos seus programas utilizando módulos **built-in** (já instalados no python ou na sua biblioteca padrão) e módulos externos, oferecidos no **Pypi.**

Utilizando o comando "**import math**" por exemplo, são adicionadas novas funções matemáticas ao seu projeto. Dentre elas temos, **ceil**: serve para arredondar um número flutuante para cima, **floor**: serve para arredondar um número flutuante para baixo, **trunc**: para quebrar a vírgula de um número e tornar uma porção inteira sem se importar com arredondamento, **pow**: função para cálculo de potência, **sqrt**: função para calculo de raiz quadrada, **factorial**: função para cálculo de fatorial.



Tem ainda como vc escolher como vc quer chamar os métodos, por exemplo: Utilizando "as" vc pode dar apelido a alguma função da biblioteca. Vejamos com o sqrt:



Para verificar as bibliotecas instaladas e desinstalar ou alterar alguma é só lembrar que a biblioteca de funções instalada esta instalada no projeto que está sendo executado (file/ settings/ project/ interpreter).

7. MANIPULANDO TEXTO

Para o Python os que está entre aspas é considerado uma **cadeia de texto ou String.** As principais operações que vamos aprender são o Fatiamento de String, Análise com len(), count(), find(), transformações com replace(), upper(), lower(), capitalize(), title(), strip(), junção com join().

No exemplo acima essa cadeia de texto ou string atribuída á variável “frase” ocupa espaços na memória, cada letra ocupa um espaço e esse espaço é atribuído a um índice (um número sequencial). No exemplo acima a frase ocupa 21 espaços (0 a 20).

7.1 FATIAMENTO DE STRING

O “**Fatiamento de String**” onde como o próprio nome sugere, fatia a string podendo retornar somente a fatia selecionada. Como no exemplo:

```
1  frase = "Curso em Vídeo Python"
2  print(frase[9])
```
```
V
```

```
1  frase = "Curso em Vídeo Python"
2  print(frase[9:13])
```
```
Víde
```

Nesse exemplo [9:13] (do índice nove até o índice treze) significa “range”, esse range nunca pega o último índice sugerido, ele sempre vai do primeiro até o penúltimo.

```
1  frase = "Curso em Vídeo Python"
2  print(frase[9:21:2])
```
```
VdoPto
```

Nesse exemplo [9:21:2] (do índice nove até o índice treze, pulando duas casas) vai pular duas casas a cada índice mostrado.

```
frase = "Curso em Vídeo Python"
print(frase[:5])
```
```
Curso
```

Nesse exemplo não foi atribuído o índice de início, então o início será o começo da string.

7.2 LISTA

O python faz uma lista através do uso dos “[ ]”

```
compras = ['Pão', 'Leite', 'Ovos']
print(random.choice(compras))
```
```
Leite
```

No exemplo acima foi criada uma lista com “Pão”, “Leite” e “Ovos” através disso foi utilizado como exemplo a biblioteca “random” para selecionar aleatoriamente um deles, foi selecionado randomicamente o “Leite”.

7.3 ANÁLISE DE STRING

Analisar uma String é saber algumas informações dela, como por exemplo, a letra de início, a letra do final, qual o tamanho da string, qual a primeira palavra inteira, etc.

```
frase = "Curso em Vídeo Python"
print(len(frase))
```
```
21
```

Função len: len vem de length que significa comprimento, essa função então mostra qual é o comprimento da frase, (quantos espaços na memória). Lembrando que os índices são contabilizados iniciando pelo 0 ou seja a variável (frase) vai do 0 ao 20.

```
frase = "Curso em Vídeo Python"
print(frase.count('o'))
```
```
3
```

Função .count: Esse comando vai contar quantos caractéres designados aparecem dentro da variável. O comando a baixo é o mesmo porem fatia e especifica onde “o” deve ser procurado dentro da variável (do 0 até o 12 já que o 13 não é considerado)

```
frase = "Curso em Vídeo Python"
print(frase.count('o', 0, 13))
```
```
1
```

```
frase = "Curso em Vídeo Python"
print(frase.find('deo'))
```
```
11
```

Função .find: Essa função vai procurar na string da variável se existe oque foi especificado, retornando como resultado o índice/localização da primeira letra. É possível utiliza **r.find**, pois vai procurar da direita (rigth) para esquerda e retornar o primeiro local encontrado da letra.

```
frase = "Curso em Vídeo Python"
print(frase.find('Android'))
```
```
-1
```

Nesse exemplo foi específicado para a função **.find** encontrar um valor inexistente dentro da variável, isso vai retornar então como um valor -1.

```
frase = "Curso em Vídeo Python"
print('Curso' in frase)
print('curso' in frase)
```
```
True
False
```

Operador in: Esse comando assim como o anterior, também vai procurar algo específico dentro da variável, porém ele não vai retornar a localização mas somente True ou False (Verdadeiro ou Falso). Lembrando também que as letras maiúsculas e minúsculas são consideradas diferentes caractéres no python, por isso devem ser específicadas adequadamente.

7.4. TRANSFORMAÇÃO DE STRING

Por via de regra uma string é imutável, ou seja, não é possível alterar os valores da variável quando especificada, porém é possível "transformá-la" de certa maneira, através de alguns comandos. O modo mais correto é executar os comandos atribuindo-os a outra variável.

```
frase = "Curso em Vídeo Python"
print(frase.replace('Python', 'Android'))
```
```
Curso em Vídeo Android
```

```
frase = "Curso em Vídeo Python"
frase.replace('Python', 'Android')
print(frase)
```
```
Curso em Vídeo Python
```

Função .replace: replace vai procurar a palavra designada, vai trocar ou substituir pela palavra designada .

```
1  frase = "Curso em Vídeo Python"
2  print(frase.upper())
```

```
CURSO EM VÍDEO PYTHON
```

Função Upper: Essa função vai mostrar todas as strings da variável em maiúscula. Porém assim como a ultima função ela não altera ou muda a variável em si, somente quando o comando é executado.

```
1  frase = "Curso em Vídeo Python"
2  print(frase.lower())
```

```
curso em vídeo python
```

Função .lower: Ao contrário da outra função ela vai retornar as strings em letras minúsculas.

```
1  frase = "Curso em Vídeo Python"
2  print(frase.capitalize())
```

```
Curso em vídeo python
```

Função .captialize: Essa função torna a retorna letra da frase maiúscula, como se tivesse iniciado um parágrafo.

```
1  frase = "Curso em Vídeo Python"
2  print(frase.title())
```

```
Curso Em Vídeo Python
```

Função .title: Essa função transforma a frase em um título, ela identifica os espaços e torna a primeira letra de cada parágrafo maiúscula.

```
1  frase = "   Aprenda Python  "
2  print(frase)
3  print(frase.strip())
```

```
   Aprenda Python
Aprenda Python
```

Função .strip: Essa função vai retirar espaços excedentes dos dois lados da frase.

```
1  frase = "   Aprenda Python  "
2  print(frase)
3  print(frase.rstrip())
4  print(frase.lstrip())
```

```
   Aprenda Python
   Aprenda Python
Aprenda Python
```

Função .rtrip: retira espaços somente da direita (**rigth**).

Função .ltrip: retira espaços somente da esquerda (**lefth**).

7.5. DIVISÃO DE STRINGS

Dividir strings através de algumas funções.

```
frase = "Curso em vídeo Python"
print(frase.split())
```

```
['Curso', 'em', 'vídeo', 'Python']
```

Função .split: Essa função vai identifica onde existem espaços e vai dividir as palavras da String, gerando uma **lista** em cada palavra (sendo ‘Curso’ [0], ‘em’ [1], ‘vídeo’ [2], ‘Pytho’ [3])

```
print(frasediv[0])
```

```
Curso
```

Esse comando vai retornar somente a lista [0].

```
print(frasediv[2][3])
```

```
e
```

Esse comando vai retornar na lista [2] a letra/índice [3].

7.6.JUNÇÃO DE STRINGS

Através dessa função é possível juntar Strings que estão separadas em lista para se tornar uma string só (uma frase só).

```
frase = "Curso em vídeo Python"
frasediv = frase.split()
print('-'.join(frasediv))
```

```
Curso-em-vídeo-Python
```

Função .join: Essa função vai juntar as strings divididas de uma variável com o termo que foi designado.

8. CONDIÇÕES

Em uma condição existem dois tipos de blocos, denominados “bloco verdadeiro” (True) e o bloco falso (False), somente um será executado. Isso será determinado pela condição **(if, else)**.

Como no exemplo abaixo, em Python a condição deve ser realizada lendo da seguinte forma: Se condição + dois pontos + tecla tab = bloco verdadeiro, Senão = bloco falso.

```
tempo = int(input('Quantos anos tem o seu carro: '))
if tempo <= 3:
    print('Seu carro é novo')
else:
    print('Seu carro é velho')
```

```
Quantos anos tem o seu carro: 2
Seu carro é novo
---Fim---
```

Existem dois tipos de estruturas condicionais, a simples e a composta, a estrutura simples vai utilizar somente o **if**, no exemplo acima iria executar o comando print somente se valor inserido fosse menor do que 3 (se o carro fosse novo). A estrutura composta pode executar um comando dependendo da condição, essa condição vai utilizar **if** e **else,** se o valor for menor do que 3 serão executados os comandos do bloco verdadeiro, se o valor for maior do que 3, serão executados os comandos do bloco falso.

8.1. INDENTAÇÃO

```
tempo = int(input('Quantos anos tem o seu carro: '))
if tempo <= 3:
    print('Seu carro é novo')
else:
    print('Seu carro é velho')
print('---Fim---')
```

Importante notar que o comando “tempo = int(input('Quantos anos tem o seu carro: '))” sempre será executado, todos os comandos com indentação para fora, ou seja sem parágrafo como no exemplo acima, serão sempre executados.

Comandos com indentação para dentro, podem ou não serem executados dependendo da condição.

8.2. CONDIÇÃO SIMPLIFICADA

```
tempo = int(input('Quantos anos tem o seu carro: '))
print('Carro novo' if tempo <= 3 else 'Carro velho')
```

Caso os comandos dos blocos sejam simples, é possível modificar a estrutura dos comandos da condição para serem realizados em poucas linhas, como no exemplo acima

8.3. CONDIÇÕES ANINHADAS

A estrutura condicional **if** e **else** possibilita a execução de 1 ou 2 blocos de comandos apenas. A condição aninhada resolve esse problema introduzindo mais uma condição dentro da condição, através do uso do **elif** (else + iif, ou em português “senão se”), possibilitando inserir infinitas condições e infinitos caminhos ou blocos de comandos possíveis, como por exemplo:

```
nome = (str(input('Insira seu nome: '))).capitalize()
if nome == "Romulo":
    print('Que nome bonito!')
elif nome == "Kelly":
    print('Que nome feio!')
else:
    print('Seu nome não tem nada de especial!')
print(f'Tenha um bom dia {nome}!')
```

Este comando está solicitando um nome, com a condição de que o nome seja “Romulo”, ele vai executar um comando, se obedecer a outra condição como o nome ser “Kelly” ele vai executar outro comando, se não for nenhuma das duas condições então o comando vai executar outro comando, e por fim vai executar um comando independente. É possível inserir condições sucessivamente com o uso do **elif.**

Outro exemplo de condição aninhada é o uso de **if** e **elif** dentro de outro, formando um “ninho” (condição aninhada):

```python
if a < (b + c) and b < (a + c) and c < (a + b):
    print('É possível formar um triângulo!')
    '''t = 1'''
    if a == b == c:
        print('Será um triângulo EQUILÁTERO!')
    elif a == b != c or b == c != a or c == a != b:
        print('Será um triângulo ISÓSCELES!')
    elif a != b != c != a:
        print('Será um triângulo ESCALENO!')
else:
    print('Não é possível formar um triângulo!')
    '''t = 0'''
```

É possível observar na leitura do código que existe uma condição “if” e uma “else” (verdadeiro e falso) porém, dentro dessa condição “if” estão implementadas outras condições “if” e “elif” que só vão acontecer se acontecer a primeira principal, formando uma espécie de ninho de condições, denominadas de condições aninhadas.

9. CORES NO TERMINAL

As cores adicionadas no terminal do Python são determinadas pelo padrão ANSI que é um padrão de normalização internacional e ele tem um padrão chamado de scape sequence. Esse padrão sempre começa com contrabarra “\” seguido de um código. O código do padrão ANSI para cores sera **\033[m** (**\ Contrabarra** para iniciar o padrão, **033** é o valor do código para cores, **[** será o local onde serão inseridos os estilos de cores, a letra **m** finaliza o código). Todo código deve ser iniciado e finalizado com o mesmo padrão \033[m.

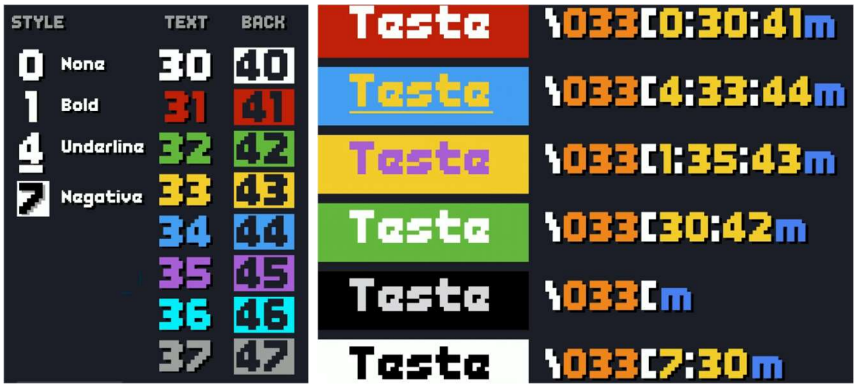
Entre a chave “[“ e a letra “m” deverão ser colocados os códigos do comportamento, somente três códigos são aceitos, o código do style (relacionado ao estilo da fonte), o código text (relacionado a cor da fonte) e o código back(relacionado a cor de fundo). Esses códigos são seguidos de “;” (ponto e vírgula) para demonstrar que você está inserido outro código. Veja o exemplo:

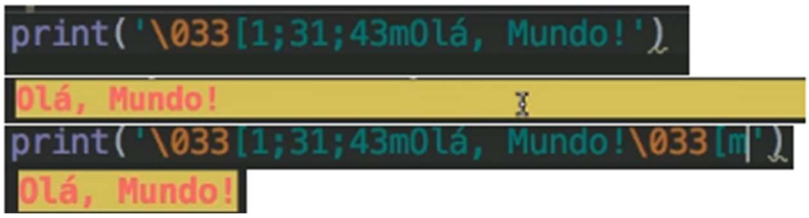

```python
print('\033[0;33;44mTESTE\033[m')
```

TESTE

Esse código está especificando que o estilo da fonte é padrão, a cor da fonte é amarela e a cor de fundo é azul.

Esses são os códigos de estilo, de cor de letra e de cor de fundo. É possível observar que os códigos “style” são indicados por somente uma unidade numérica, os códigos de cor de letra são indicados pelo número 30 em diante e os códigos de cor de fundo são indicados pelo número 40 em diante.



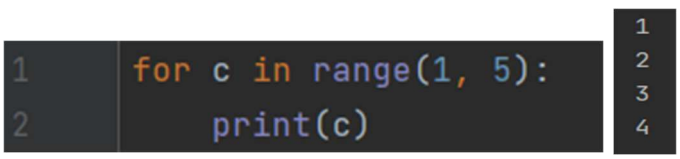


Lembrando que \033[m retira as formatações, deve sempre ser colocado no final do local especificado.

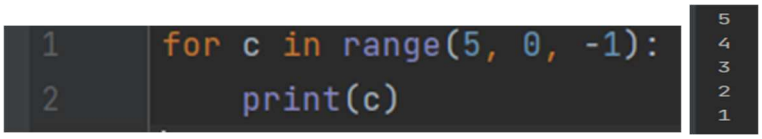
10. ESTRUTURA DE REPETIÇÃO FOR

Denominamos “Laço” (Loop em inglês) como uma estrutura de repetição. O comando **“for “** é muito útil quando se sabe de antemão quantas vezes a repetição deverá ser executada. Ele cria um laço que utiliza uma variável para controlar a contagem do loop, bem como seu incremento **iteração**. É conhecida como uma estrutura de repetição com variável de controle, onde existe um controle que demonstra o quanto o comando será repetido.

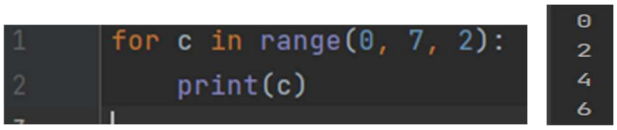
Na imagem abaixo o comando **“for”** cria o laço **“c”** no **“range”** (intervalor) entre 1,5 (sendo 4 índices já que o ultimo não é contabilizado):



É possível observar assim como estruturas condicionais, a presença de endentação, ela vai determinar os comandos que serão executados pela estrutura de laço “for”. Importante notar também, como o comando deve ser executado.

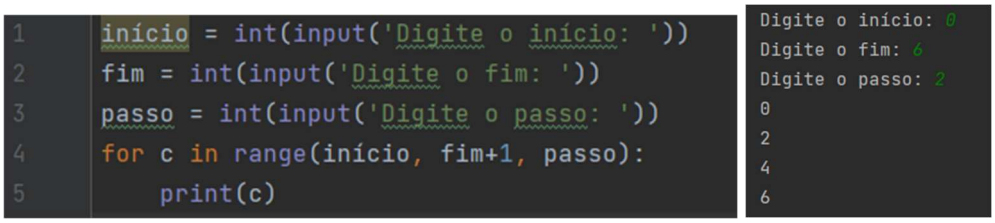


No exemplo acima, estrutura de repetição **“for”,** o **“range”** determina o intervalo em que a repetição será realizada, sendo o primeiro valor o início, o segundo valor o fim, o terceiro valor será a **iteração** que é o que será feito após o laço (loop) ser realizado. No exemplo acima o laço vai contar de 5 até 0 (sendo o ultimo índice o 1), porém quando chegar ao final ele vai subtrair um número, fazendo então uma contagem regressiva.



No exemplo acima esta sendo solicitado para realizar um laço que vai contar de 0 até 7 pulando de 2 em 2.

No range, no lugar dos números inteiros já pré-estabelecidos, é possível colocar variáveis para determinar o início, fim e a iteração. Como no exemplo:



É possível observar que para contornar a condição da contagem não contabilizar o último número, foi inserido um “+1” ao lado da variável “fim”.

```
s = 0
for c in range(0, 4):
    n = int(input('Digite um número: '))
    s = s + n
print(s)
```

```
Digite um número: 1
Digite um número: 2
Digite um número: 3
Digite um número: 4
10
```

Nesse exemplo a variável s já está determinada com o valor 0, a estrutura de repetição **“for”** cria de um laço que vai repetir a solicitação de um input para a variável n dentro de um intervalo de 0 até 4, ou seja esse laço vai repetir a solicitação 4 vezes. Ainda dentro do laço um comando para alterar a variável s é acrescentado e vai alterar a variável s somando a mesma com o que será inputado na variável n, isso também sera repetido 4 vezes. Ao final o comando print vai exibir o resultado da variável s, que de acordo com a logica colocada nos comandos vai exibir a somatória do que foi imputado na variável n durante as 4 vezes.

O próximo exemplo pede para mostrar todos os números pares até o número 50.

```
for c in range (1,51,):
    if c % 2 == 0:
        print(c, end=' ')
```

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
```

Com isso uma condição vai ser colocada, essa condição diz que, serão mostrados os valores do laço “c” somente se o valor do resto da divisão por 2 for igual a 0, isso vai retornar todos os valores pares. É importante notar o uso da configuração de linha **“end=”** que vai formatar o que vai acontecer no final da linha e ou seja não serão realizadas os pulos de linha, após o final da linha serão colocados espaços “_” para os valores do print, não ficarem sobrepostos e juntos.

```
for n in range(2, 51, 2):
    print(n, end=' ')
```

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
```

Esse exemplo vai executar a mesma ação e fazer a mesma coisa, só que vai utilizar outra lógica, mais simples que vai utilizar menos espaço no processador.

No lugar de inserir a condição **"if"** tudo será realizado através do uso correto da estrutura de repetição **"for"**. No código então esta determinando para contar entre 2 e 51 (51 por que 50 não será contabilizado como último número), tudo será contado de 2 em 2 (de acordo com o que foi determinado na terceira casa do range), por isso a contagem começa no 2.

11. ESTRUTURA DE REPETIÇÃO WHILE

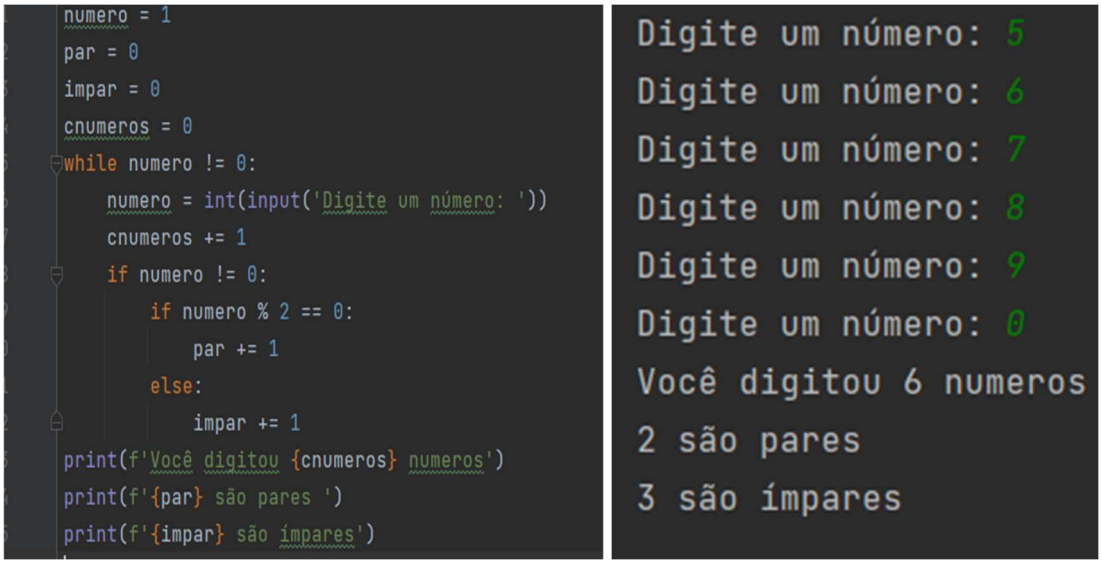
O comando **"while"** (traduzido para enquanto) é uma estrutura de repetição com teste lógico, ou seja, diferente da estrutura de repetição **"for"**, o loop será executado enquanto um teste lógico for aceito como True (Verdadeiro). Exemplo:

```
resposta = 'S'
while resposta == 'S':
    numero = int(input('Digite um valor: '))
    resposta = input('Deseja continuar? [S/N]: ').upper()
print('Fim')
```

```
Digite um valor: 8
Deseja continuar? [S/N]: s
Digite um valor: 5
Deseja continuar? [S/N]: s
Digite um valor: 4
,Deseja continuar? [S/N]: n
Fim
```

Nesse exemplo, primeiramente, é atribuído um valor para a variável resposta, o comando while estabelece um loop que diz que enquanto a variável "resposta" for = "S" (SIM), o loop vai continuar e tudo que está dentro dele fica sendo executado, com isso não há um controle específico sobre quantas vezes deve ser repetido ou qual maneira/caminho as repetições devem seguir, somente

deve ser atendido um teste lógico, se esse teste for atendido como verdadeiro (True) o loop será mantido.



```
numero = 1
par = 0
impar = 0
cnumeros = 0
while numero != 0:
    numero = int(input('Digite um número: '))
    cnumeros += 1
    if numero != 0:
        if numero % 2 == 0:
            par += 1
        else:
            impar += 1
print(f'Você digitou {cnumeros} numeros')
print(f'{par} são pares ')
print(f'{impar} são ímpares')
```

```
Digite um número: 5
Digite um número: 6
Digite um número: 7
Digite um número: 8
Digite um número: 9
Digite um número: 0
Você digitou 6 numeros
2 são pares
3 são ímpares
```

Aqui está um exemplo de uma estrutura de repetição do tipo **while** junto de uma estrutura de condição **if**. Nesse exemplo foram inseridas 4 variáveis que serão determinadas dentro do loop, por isso são atribuídos valores 0, a não ser pela variável “numero”, porque a repetição while diz que, enquanto a variável números for diferente de 0, o loop será realizado, por isso que se números já fosse atribuído 0 todos os comandos já seriam invalidades e o programa não realizaria o lopp. Então continuando, o comando while diz que quando o input na variável números determinar o número como 0, o loop será interrompido, enquanto ele não é interrompido por esse teste logico dado como falso, ele vai executar o comando de, solicitar o input do número, contabilizar quantos números estão sendo preenchidos, realizar uma condição que diz que, se desconsiderando o número 0, se o valor do resto da divisão do número por 2 for igual a 0, o número será par, senão o número será ímpar, o comando então quando executado faz tudo isso como no exemplo solicitou 6 números (contando com o número 0), quando chegou o 0 ele parou de ser executado, contabilizou quantos números foram digitados e deu o valor de quantos foram pares (6 e 8) e quantos foram impares (5,7 e 9) não contando o 0 como par, já que o resto da divisão de 0 por 2 da 0.

11.1. INTERROMPENDO REPETIÇÕES WHILE

Como já mencionado, a estrutura de repetição while vai repetir um loop de comandos enquanto um teste lógico for dado como verdadeiro, porém é possível tornar essa repetição de forma infinita, tornando o próprio teste lógico verdadeiro.

```
1   while True:
2       n = int(input('Digite um número: '))
```

```
Digite um número: 7
Digite um número: 5
Digite um número: 4
Digite um número: 6
Digite um número: ..
Traceback (most rece
```

O programa só foi interrompido por um erro, (o valor do input não era um número inteiro aceito), também pode ser interrompido utilizando a função “stop” do terminal, porém é possível interromper corretamente o loop infinito **“while”** assim como qualquer outro loop utilizando a função **break.**

É importante notar que tornar o a estrutura **while true**, é a maneira correta para fazer loops onde serão inseridos valores de variáveis, já que antes de saber essa função eram atribuídos valores como **0, 1** ou **False**, e isso não é a maneira mais correta de se fazer, já que inserindo uma variável assim, o programa acaba puxando mais espaço da memória do computador.

```
1   while True:
2       n = int(input('Digite um número: '))
3       if n == 0:
4           break
```

```
Digite um número: 7
Digite um número: 5
Digite um número: 6
Digite um número: 0
```

Nesse exemplo, o programa vai solicitar um input para a variável “n”, infinitas vezes, porém foi colocada uma condição que diz que, se a variável “n” for igual a 0, a função **break** será acionada, e essa função vai interromper esse loop assim como foi mostrado.