## Resumo do conteúdo apresentado:

* Variáveis
* Tipos
* Comandos Básicos
* Estruturas de Controle
  + Estrutura de Sequência
    - **suite**
  + Estrutura de Seleção
    - **if,if-else,if-elif,if-elif-else**
  + Estrutura de repetição
    - **while** e **for**
* Subprogramação:
  + Funções
  + Passagem de Parâmetros
    - Passagem por Valor,
    - Passagem de função
  + Recursividade
* Estrutura de dados:
  + Vetor
    - **split()**
  + Matriz
    - **append()**
  + String (Cadeia de Caracteres)
    - **find()**
  + Tupla
  + Lista
    - Lista de listas
    - Lista Lisp-Like
* Persistência dos dados
  + Arquivo texto
    - **read()**
    - **write()**
    - **append()**

## Variáveis

### Tipos básicos:

* Tipos Integrais:
  + Inteiro: (**int**)
    - Pode ter centenas de dígitos
    - O padrão é decimal, mas pode usar bases como binária (iniciada com **0b**), octal (iniciada com **0o**) ou hexadecimal (iniciada com **0x**)
  + Lógico (ou Booleano): (**bool**)
    - 0 é **False** e 1 é **True**
* Tipos de Ponto-Flutuante
  + Número de ponto-flutuante: (**float**)
  + Número Complexo: (**complex**)
* String (**str**)
  + Representada por uma sequência de caracteres Unicode, iniciada e terminada por aspas simples ou duplas

## Tipos

### Conversão de Tipos:

* **int**: para inteiro
* **float**: para ponto flutuante
* **bool**: para booleano
* **str**: para string

### Operadores únicos:

* **+**: Operador numérico positivo
* **-**: Operador numérico negativo
* **not**: Operador lógico de negação

### Operadores Binários Aritméticos:

* **+**: Soma
* **-**: Subtração
* **\*** : Produto
* **/**: Divisão de ponto flutuante
* **//**: Divisão inteira
* **%**: Resto da divisão inteira
* **\*\***: Potencialização

### Operadores Binários Lógicos:

* **or**: Disjunção lógica ou soma lógica
* **and**: Conjunção lógica ou produto lógico

### Operadores Binários Relacionais:

* **==**: Igual a
* **!=**: Diferente de
* **>**: Maior que
* **<**: Menor que
* **>=**: Maior ou igual
* **<=**: Menor ou igual

## Comandos básicos

### Comandos de saída padrão:

* **print()**
  + Pula para a próxima linha
* **print(***expressão***)**
  + Escreve o resultado da expressão
  + Pula para a próxima linha
* **print(***expressão1*,…,*expressãoN***)**
  + Escreve o resultado de cada expressão
  + Espaço em branco é escrito entre cada par de expressão
  + Pula para a próxima linha
* **print(***expressão*,**end=***término***)**
  + Escreve o resultado da expressão
  + Ao final, escreve o String de *término*
* **print(***expressão1*, …, *expressãoN*,**end=***término***)**
  + Escreve o resultado de cada expressão
  + Espaço em branco é escrito entre cada par de expressão
  + Ao final, escreve o String de *término*

### Expressões Formatadas (operador %(…)):

* Sintaxe da expressão formatada:
  + *texto = “…%formato1 … %formatoN…” %(exp1, …, expN)*
* Formatos mais comuns:
  + **d** (inteiro), **f** (número com ponto flutuante) e **s** (String)

Exemplos:

msg = " A media dos numeros é %4.2f" %(8656656.6164134)

**print**(msg)

**print**("%f + %f = %4.1f" %(15, 9.578421, 15+9.578421 ))

**print**("%d + %d = %d" %(5, 7.578421, 5+7.578421 ), end="!!!")

### Comandos de entrada padrão:

* **input()**
  + Comando que aguarda o usuário fornecer um valor expresso por uma sequencia de caracteres
  + Este comando suspende a execução do programa até que o usuário escreva sua entrada e precione enter.
* **input(***mensagem***)**
  + Comando que escreve a expressão *mensagem* e aguarda o usuário fornecer um valor expresso por uma sequencia de caracteres
  + Este comando suspende a execução do programa até que o usuário escreva sua entrada e precione enter.

## Estrutura de Controle

### Sequência

#### Suite

* Em Python, ao construir uma sequência de comandos, devemos colocá-los um por linha abaixo do outro, mantendo-os na mesma tabulação.
* Um bloco de comando que possui um ou mais comandos é camado de **suite**

#### Comentários

* Comentários simples: iniciados com o caractere #
* Comentarios com múltiplas linhas: delimitados por “”" “”"

### Estruturas de seleção

#### Estrutura de seleção de 1 ramo (*if*)

Uma estrutura de seleção com um ramo **if** é utilizada quando se deseja executar um comando (ou uma suite) apenas no caso de determinada condição ser satisfeita.

Exemplo do uso:

dinheiro = 50000

**if** dinheiro>10000:

imps = dinheiro \* 0.1

multa = dinheiro \* 0.15

**print**( "Valor do Imposto:", imps, "e da multa:", multa)

Para condições compostas pode ser feita a condição utilizando:

* **and**,
* **or**,
* **10<=x<=20**

#### Estrutura de seleção de 2 ramos (*if-else*)

A estrutura de dois ramos *if-else* é utilizada quando deseja-se executar uma entre duas suites, dependendo de como foi avaliada a condição (como **True** ou **False**).

Exemplo de uso:

x=10

**if** x>0:

**print**(x, "é maior do que zero.")

**else**:

**else**(x, "é menos ou igual a zero.")

##### Estrutura de Seleção Aninhadas

Veja o exemplo a seguir:

"""

Calcula ajuste de dinheiro, se tempo < 1 mantém , se tempo < 10 faz uma coisa, se tempo >= 10 faz outra:

"""

dinheiro = 1000.50

tempo = 20

**if** tempo<1:

**print**("Seu dinheiro continua:", dinheiro)

**else**:

**if** tempo<10:

dinheiro = dinheiro \* 1.1

**else**:

dinheiro = dinheiro \* 1.2

**print**("Seu dinheiro apos o ajuste:", dinheiro)

#### Estrutura de Seleção com Múltiplos Ramos (*if-elif* ou *if-elif-else*)

As estruturas de seleção **if-elif** e **if-elif-else** tem uso quando deseja-se executar uma entre várias suites, dependendo do resultado da condição avaliada.

veja o exemplo a seguir com uso do **if-elif** :

""

Faremos um comando que funciona como calculadora.

""

valores = [5,9] *#Poderiamos pedir que o usuario informasse com:*

*#valores=input("Digite dois valores: ").split()*

x = int(valores[0])

y = int(valores[1])

op = "+" *#Poderiamos pedir que o usuario informasse*

**if** op=="+":

resultado = x+y

**elif** op=="-":

resultado = x-y

**elif** op=="\*":

resultado = x\*y

**elif** op=="/":

resultado = x/y

**elif** op=="\*\*":

resultado = x\*\*y

**print**(x,op,y,"=",resultado)

veja o exemplo a seguir com uso do **if-elif-else** :

""

Faremos um comando que funciona como calculadora.

""

valores = [5,9] *#Poderiamos pedir que o usuario informasse com:*

*#valores=input("Digite dois valores: ").split()*

x = int(valores[0])

y = int(valores[1])

op = "+" *#Poderiamos pedir que o usuario informasse*

**if** op=="+":

resultado = x+y

**elif** op=="-":

resultado = x-y

**elif** op=="\*":

resultado = x\*y

**elif** op=="/":

resultado = x/y

**elif** op=="\*\*":

resultado = x\*\*y

**else**:

resultado = None

**if** resultado == None:

**print**(op, "Operador inexistente!!")

**else**:

**print**(x,op,y,"=",resultado)

### Estruturas de Repetição

#### Estruturas de Repetição Indefinida (*while*):

A estrutura de repetição indefinida **while** deve ser utilizada quando se deseja executar zero ou mais vezes uma suite enquanto a condição estipulada for verdadeira.

Exemplo de uso:

i=1

**while** i<=10:

**print**(i, end=" ")

i=i+1

**print**()

#### Estruturas de Repetição Definida (*for*):

A estrutura de repetição definida \*\*for\* deve ser utilizada quando se deseja, um determinado número de vezes, executar uma mesma suite, enumerado ou nao por uma lista de valores.

Exemplo de uso:

**for** i **in** [2,3,4,5,6,7,8,9]:

**print**(i, end=" ")

**print**()

##### Listas de Valores para uma Repetição *for*

Para criar uma lista para repetições **for**:

* **range(***valor limite***)**
  + Cria uma lista com P.A. de itens de razão 1, iniciada pelo valor 0 e termina no valor que antecede o *limite*

Exemplo:

**print**(range(7))

* **range(***valor inicial, valor limite***)**
  + Cria uma lista com P.A. de itens de razão 1, iniciada pelo *valor inicial* e termina no valor que antecede o *limite*

Exemplo:

**print**(range(7,20))

* **range(***valor inicial, valor limite, avanço***)**
  + Cria uma lista com P.A. de itens de razão *avanço*, iniciada pelo *valor inicial* e termina no valor que antecede o *limite*

Exemplo:

**print**(range(9,50,3))

## Subprogramação

### Funções

São suites, comandos e/ou estruturas de controle, ao qual atribuímos um nome e após a sua execução obtemos um valor.

A utilização de funções permite que o programa possa ser desenvolvido e testado separadamente, além de permitir a reutilização de algumas partes em diferentes pontos do programa.

Sendo assim, é possível montar programas mais complexos a partir de funções menores já desenvolvidas e testadas.

#### Declaração de uma função em Python

A função deve ser declarada da seguinte forma:

def nomeEscolhido(\_lista de parâmetros\_)

suite do corpo da funcao

onde:

* a *lista* de parâmetros pode ter zero ou mais parâmetros separados por vírgulas
* A *suite do corpo da função+* deve possuir zero ou mais retornos de valores, expressos por **return** *valor apropriado* (caso nenhum valor seja retornado, **return None**)

### Passagem de Parâmetros

Caso uma função possa ser aplicada a diferentes valores de entrada, esta função deve conter parâmetros em sua definição.

#### Passagem de parâmetro por valor

* No início da função, os parâmetros sempre são inicializados com a cópia dos ponteiros (referências) para os valores implementados na ativação da função
  + valor de tipo básico são imutáveis
  + Valores de tipos estruturados são mutáveis
* Os valores podem vir de:
  + Constantes
  + Variáveis
  + Resutlados de funções
  + Ou seja, resutlado de expressões

#### Passagem por função

Da mesma maneira que por valor, a função pode receber ou retornar outra função definida previamente

### Recursividade

Chama-se de função recursiva quando a função possui no seu corpo uma chamada a ela própria

Exemplo, no cálculo de fatorial:

**def** fat(n):

**if** n==0:

**return** 1

**else**:

**return** n\*fat(n-1) *#Aqui esta a chamada recursiva*

## Estrutura de Dados

### Vetor

Um vetor é um **agregado** de elementos (também chamados de valores) de um **mesmo** tipo.

Imagine um vetor, do qual para a seleção de elementos dentro desse vetor, podemos utilizar um índice ou seletor do elemento que indica sua posição ou ordem.

vetor=[4,3,2,1]

**print**(vetor[0])

**print**(vetor[3])

* Caracteristicas:
  + É uma estrutura homogênea
  + Todos os elementos da estrutura são igualmente acessíveis
  + cada elementro da estrutura tem um nome próprio

Um vetor pode ser simples ou estruturado

* Acesso a elementos do vetor?
  + Suponha que se queira somar todos os elementos do vetor?

*#Utilizando o range()*

soma = 0

**for** i **in** range(3):

soma = soma + vetor[i]

*#Utilizando o len()*

soma = 0

**for** i **in** range(len(itens)):

soma = soma + itens[i]

* A leitura de um vetor pode ser realizada:
  + Elemento a elemento
  + Todas as informações podem ser lidar de uma vez, como uma linha de caracteres, sobre o qual se aplica a operação **split** para separa-las

Exemplo de leitura:

*#Leitura de todos os valores de uma vez:*

valores = input("Digite os valores na mesma linha: ").split()

*#Leitura um de cada vez:*

valores=[None]\*10

**for** i **in** range(len(valores)):

valores[i] = input("Digite um valor: ")

* A escrita de um vetor pode ser realizada;
  + Toda de uma vez
  + Elemento a elemento

Exemplo de escrita:

*#Todos de uma vez:*

numeros= [4,6,2,5,1,25,12,32]

**print**(numeros)

*#Escrita um de cada vez:*

numeros= [4,6,2,5,1,25,12,32]

**for** x **in** numeros:

**print**(x, end=" ")

**print**() *#Para pular uma linha*

### Matriz

A matriz 2D é um vetor de vetores, vejamos um exemplo:

aposta = [[" ","X"," "],

[" "," ","X"],

[" ","X"," "],

["X"," "," "],

[" ","X"," "],

[" "," ","X"],

[" ","X"," "]]

**print**(aposta)

Para acessar os elementos desse vetor:

aposta = [[" ","X"," "],

[" "," ","X"],

[" ","X"," "],

["X"," "," "],

[" ","X"," "],

[" "," ","X"],

[" ","X"," "]]

aposta *#Representa uma matriz de 7 linhas por 3 colunas*

aposta[0] *#Representa a primeira linha da matriz*

aposta[0][0] *#Representa o caractere da primeira linha da primeira coluna*

#### Entrada dinâmica de Vetores e Matrizes

Em python, vetores e matrizes são implementados por listas, uma forma alteranativa é utilizando a operação que anexa um valor ao final de uma lista: **append()**.

### String (Cadeia de Caracteres)

Um objeto do tipo **str** representa uma cadeia de caracteres, de tamnho e valor imutáveis, veja um exemplo de declaração e atribuição:

nome = "Ana"

**print**(len(nome)) *# confima o comprimento ou quantidade de caracteres na string*

* Comparação de Strings
  + comparação lexicografica de strings com os operadores ==,!=, <=, >=, <, >
* Indexação de cada caractere
  + O tipo de String pode ser tratado tambem como vetor, vejamos:

*#Se:*

nome = "Ana"

*#Então:*

**print**(nome[0])

**print**(nome[1])

**print**(nome[2])

* O operador **+**, quando aplicado a dois Strings *a* e *b* retorna uma String concatenada das Strings *a* e *b*
* Retornando uma “nova” sub-String de uma String:

nomeString[posicao inicial : posicao final+1]

* Método **find(***subStringProcurada***)**
  + Retorna a posição do indice da primeira ocorrencia da subStringProcurada na String sendo consultada. Caso nao seja encontrada, retorna menor um (-1)
* Outros métodos **importantes**:
  + **replace(** *subStringProcurada*, *subStringNova* **)**
    - Retorna uma copia da string sendo consultada, substituindo
  + **count(** *subStringProcurada* **)**
    - Retorna a quantidade de ocorrencias
  + **upper()**
    - Retorna uma cópia da String, convertendo tudo para maiúsculo
  + **lower()**
    - retorna uma cópia da String, convertendo tudo para minusculo
  + **strip()**
    - Retorna uma cópia da String, removendo todos caracteres brancos do início e do final
  + **split()**
    - Retorna uma lista de todas as palavras String
  + **split(** *subStringSeparadora* **)**
    - Retorna uma lista de todas as palavras String, sendo o delimitador procurado entre palavras aquele especificado em subStringSeparadora.
* Metodos para leitura e escrita:
  + **input()**, **readline**, **print**, **write**

### Tupla

Uma sequência ordenada de zero ou mais referências a objetos. Tuplas suportam o mesmo fatiamento, o mesmo acesso por iteradores e o mesmo desempacotamento que Vetores e Strings. São imutáveis e pode ser vazia.

Exemplo:

vazio = tuple()

**print**(vazio)

Mais exemplos, agora de tuplas não vazias:

val = ("abacaxi", 500, 4.99)

**print**(val)

*#ou ainda:*

val = "abacaxi", 500, 4.99

**print**(val)

Sendo assim, vetores e strings podem ser empacotados como tuplas através da funcao **tuple()**, veja:

trio = tuple([1,2,3]) *# ou trio = (1,2,3)*

**print**(trio)

letras = tuple("aeiou") *# ou letras = ("a", "e", "i", "o", "u")*

**print**(letras)

#### Exemplo de desempacotamento e iteração

Seu conteúdo pode ser facilmente atribuído a variáveis por meio de desempacotamento, veja:

individuo, altura = ("Maria Jose", 139)

**print**(individuo)

**print**(altura)

*#ou ainda:*

(nome, idade) = ("Maria Jose", 139)

**print**(nome)

**print**(idade)

*#Iterando sobre os itens de uma tupla:*

tomDePele = ("Marrom", "Amarelo", "Azul", "Branco")

**for** cor **in** tomDePele:

**print**(cor)

#### Métodos disponíveis sobre Tuplas

* **count(***valor***)**
  + Retorna a quantidade de ocorrencias de um determinado valor da tupla, exemplo:

v = ("morango", 500, 4.99, 500)

**print**(v.count(500)) *#Retorna quantidade*

* **index(** *valor* **)**
  + Retorna o índice da primeira ocorrência do valor informado como argumento, exemplo:

v = ("morango", 500, 4.99, 500)

**print**(v.index(500)) *#Retorna indice*

#### Operadores sobre Tuplas

* **concatenação:** *a+b*
  + Gera uma nova tupla a partir do conteúdo de a seguido de b
* **replicação:** \_a\*n\_
  + Gera uma nova Tupla a partir do conteúdo de (a) repetida n-1 vezes
* **fatiamento:** a[*posição inicial : posição final+1*]
  + Gera uma nova Tupla a partir do subconjunto de elementos contidos em a
* **atribuição incremental:** *a +=b* ou \_a\*=n\_
  + Equivalente a concatenação e repetição, porém atribui à variável (a) a referência para a nova tupla gerada
* **comparação:** *< , <+ , == , !=, >, >=*
  + Compara item a item
* **associação:** *in* e *not in*
  + Verifica a pertinência de um valor em uma tupla

## Listas

### Lista

Uma lista é uma sequencia ordenada pelo índice, de zero ou mais referências a objetos.

Características: \* É uma estrutura de dado recursiva \* Representada por uma sequência, fechada por colchetes ( **[\*\* e \*\*]**) \* O primeiro elemento esta na posição zero \* Lista são mutáveis, podem receber novos elementos, substituir ou remover antigos elementos

Exemplo:

*#Declarando uma lista:*

salada = []

*#Incluindo valores:*

salada = ["manda", "pera", "uva"]

Operações básicas para manipulação de listas:

* Operações de **inclusão** de novos elementos:
  + **apend(** *novoElemento* **)**: anexa um *novoElemento no final* da lista
  + **insert(** *pos,novoElemento* **)**: insere o *novoElemento) na posição* pos\* da lista, caso essa posição não exista, será criada

Exemplo:

*#Incluindo valores:*

salada = ["manda", "pera", "uva"]

**print**(salada)

salada.append("banana")

**print**(salada)

salada.insert(2,"goiaba")

**print**(salada)

## ['manda', 'pera', 'uva']

## ['manda', 'pera', 'uva', 'banana']

## ['manda', 'pera', 'goiaba', 'uva', 'banana']

Criando uma lista de numeros aleatórios em um intervalo pre estabelecido pelo usuário:

*#Exemplo 1*

*#subprogramas*

**def** preencher(listaElems, qtd, min, max):

**from** random **import** randint *#Requer pacote com funcao externa*

**for** item **in** range(qtd):

listaElems.append(randint(min, max))

**return** None

*#programa principal*

qtdNumeros = int(input("A Lista deve ter quantos valores?"))

minimo = int(input("Menor valor da faixa:"))

maximo = int(input("Maior valor da faixa:"))

numeros = []

preencher(numeros, qtdNumeros, minimo, maximo)

**print**(numeros)

* Operações de **exclusão**: de novos elementos:
  + **pop(** **)**: retorna e remove o último elemento da lista.
  + **pop(** *pos* **)**: retorna e remove o elemento na posição *pos* da lista.
  + **remove(** *x* **)**: remove a primeira ocorrência do item *x*.
* Outras operações úteis com listas:
  + **les(** **)**: retorna o comprimento da lista.
  + lista.**count(** *elemento* **)**: retorna quantas vezes o *elemento* aparece na lista
  + lista.**sort(** *lista* **)**: ordena o conteúdo da *lista*

#### Fatiamento de Listas

* Fatiamento de Listas:
  + *listaAntiga*[*posInicio* : *posFim* \*]\*: retorna uma nova lista composta de referências para elementos existentes na listAntiga

Exemplo:

saladaComposta = ["banana", "caju", "uva", "pera", "manga", "kiwi"]

**print**(saladaComposta)

saladaSimples = saladaComposta[1:4]

**print**(saladaSimples)

## ['banana', 'caju', 'uva', 'pera', 'manga', 'kiwi']

## ['caju', 'uva', 'pera']

### Listas de listas

Os indices fazem referências a novas listas contidas no seu interior

Veja um exemplo:

mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30,5.9], ["caju", 15.35]]

**print**(mercado)

## [['pera', 100, 4.9], ['manga', 20, 3.9], ['uva', 30, 5.9], ['caju', 15.35]]

#### Operando elementos de uma lista de listas

Confira:

mercado = [["pera", 100, 4.9], ["manga", 20, 3.9], ["uva", 30,5.9], ["caju", 15.35]]

**print**(mercado)

mercado[1][2] \*= 0.5 *#Manga pela metade do preço*

**print**(mercado)

mercado[3][1] -= 10 *#caju com dez quilos a menos*

**print**(mercado)

mercado.remove(["uva", 30,5.9]) *#O produto uva é removido do mercado*

**print**(mercado)

mercado.insert(1, ["kiwi", 200, 1.99]) *#O produto kiwi é inserido*

**print**(mercado)

## [['pera', 100, 4.9], ['manga', 20, 3.9], ['uva', 30, 5.9], ['caju', 15.35]]

## [['pera', 100, 4.9], ['manga', 20, 1.95], ['uva', 30, 5.9], ['caju', 15.35]]

## [['pera', 100, 4.9], ['manga', 20, 1.95], ['uva', 30, 5.9], ['caju', 5.35]]

## [['pera', 100, 4.9], ['manga', 20, 1.95], ['caju', 5.35]]

## [['pera', 100, 4.9], ['kiwi', 200, 1.99], ['manga', 20, 1.95], ['caju', 5.35]]

#### Lista (Lisp-Like)

Três operações primitivas do paradigma de programação funcional na lisguagem **Lisp** (List Processing):

* **car(** *dados* **)**: é a operação seletora que retorna o primeiro elemento de uma lista de dados.
* **cdr(** *dados* **)**: é uma operação seletora que retorna uma lista com todos os elementos da lista dados, exceto o primeiro.
* **cons(** *item*, *dados* **)**:é uma operação construtora que retorna uma lista que contem o item como primeiro elemento, seguido pela lista de dados.

Exemplo (trivial):

**def** car(dados):

**return** dados[0]

**def** cdr(dados):

**return** dados[1:len(dados)]

**def** cons(item, dados):

**return** [item] + dados

Utilizando as operações seletoras **car** e **cdr**, vejamos uma função recursiva que some o conteúdo de uma lista de numeros:

**def** soma(dados):

**if** dados==[]:

**return** 0

**else**:

**return** car(dados) + soma(cdr(dados))

## Persistência dos dados

### Arquivos

Programas interativos sao diferentes de manusear arquivos

arquivos podem ser texto

O arquivo texto é um sequencia de caracteres organizadas em linhas

em python, a leitura de arquivos é feito da seguinte maneira:

dados = open(caminho do arquivo, "r") *#r: leitura ;*

*#w: apenas escrita;*

*#a: escreve no final do arquivo*

dados.close() *# o arquivo nao sera mais utilizado*

#### Metodo readline()

Aplicada sob um arquivo txt aberto, retorna uma linha completa, incluind o final da linha (\* que pula a linha). Portando o inicio do leitor avança para a proxima linha, veja:

dados = open("exemplo.txt", "r")

linha = dados.readline()

**print**(linha, end="")

dados.close()

##### Leitura do arquivo texto com o metodo *readline()*

Veja:

dados\_abrir=input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(dados\_abrir, "r")

linha = dados.readline()

**while** linha != "":

**print**(linha, end="")

linha=dados.readline()

dados.close()

Outra maneira:

dados\_abrir=input("Digite o nome do arquivo que deseja visualizar: ")

dados = open(dados\_abrir, "r")

**for** linha **in** dados:

**print**(linha, end="")

dados.close()

#### Comando readlines()

Funciona apenas para pequenos arquivos, veja:

dados\_abrir = input("Digite o nome do arquivo:")

dados=open(dados\_abrir, "r")

linhas = dados.readlines()

**for** linha **in** linhas:

**print**(linha, end"")

dados.close()

#### Produzindo um arquivo de texto com *write()*

dados=open("teste.txt", "w")

dados.write("qualquercoisa")

##### Com quebra de linha:

Veja:

dados=open("teste.txt", "w")

dados.write("qualquercoisa\n")

Mais um exemplo:

arquivo = input("Digite o nome do arquivo que sera criado:")

qnt\_linhas = int(input("Quantas linhas:"))

dados = open(arquivo, "w")

**for** i **in** range(qnt\_linhas):

nova=input("linha"+str(i+1)+":")

dados.write(nova+"\n")

dados.close()

#### Anexando uma nova linha no final de um arquivo

Veja:

nome=input("Diga o nome do arquivo que deseja anexar")

arquivo=open(nome, "a")

noma\_linha = input("Diga a nova linha:")

arquivo.write(nova\_linha + "\n")

arquivo.close()