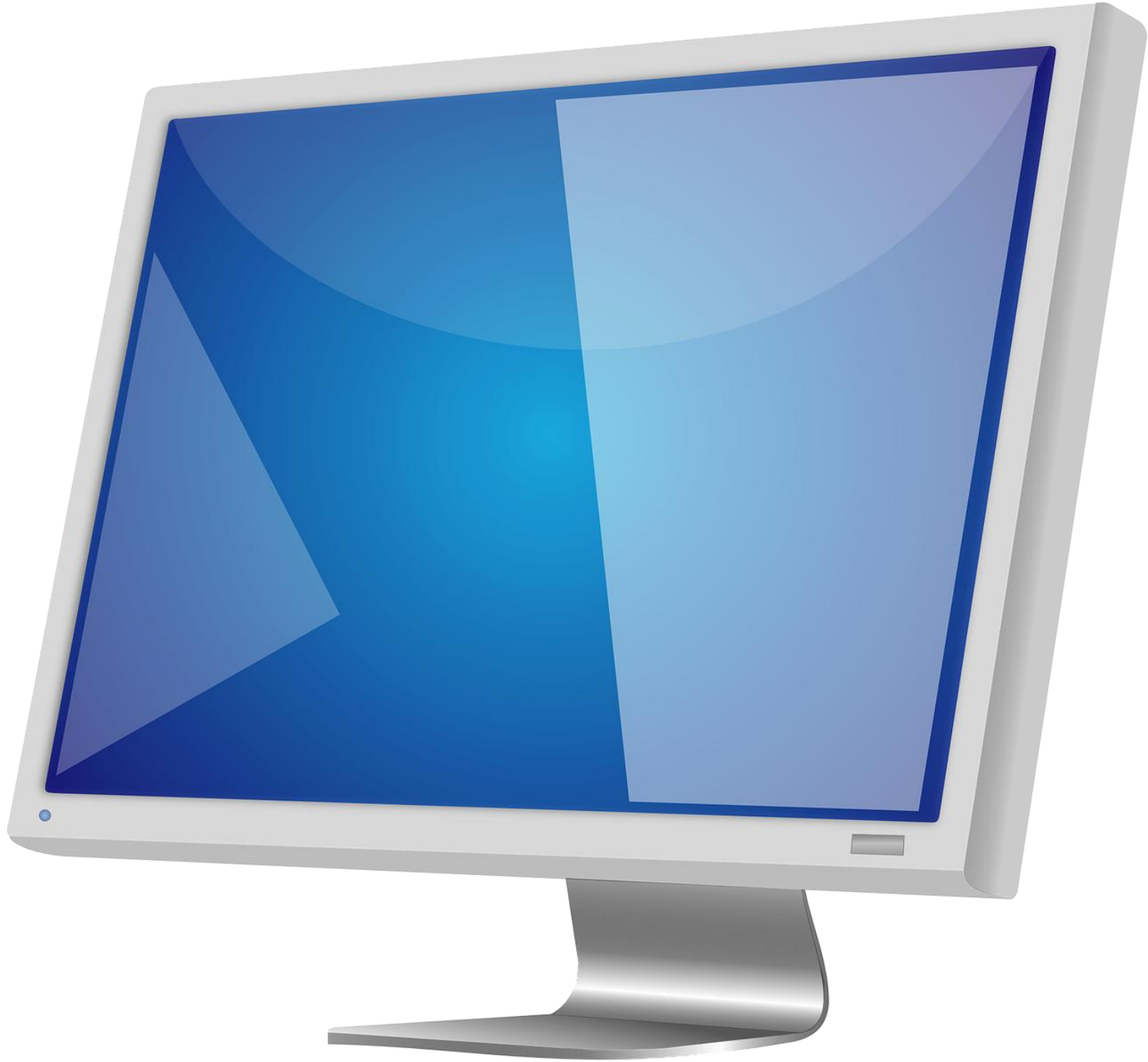
PYTHON FAST



Universidade PYTHON

Introdução ao Python

Vamos escrever nosso primeiro programa em Python, "Olá, mundo!". É um programa simples que imprime Olá Mundo!

```
print("Hello, World!")
```

Variáveis e literais

Uma variável é um local nomeado usado para armazenar dados na memória. Aqui está um exemplo:

```
a = 5
```

Aqui, **a** é uma variável. Atribuímos **5** à variável **a.**
Não precisamos definir o tipo de variável no Python. Você pode fazer algo assim:

```
a = 5
print("a =", 5)
a = "cinco"
print("a =",a)
```

Variáveis e literais

Inicialmente, o valor inteiro 5 é atribuído à variável **a** . Então, a string “cinco” aqui é atribuído à mesma variável.

A propósito, 5 é um literal numérico e "cinco" é uma string literal.

Quando você executa o programa, a saída será:

```
a = 5
a = cinco
```

Operadores

Operadores são símbolos especiais que executam operações em operandos (variáveis e valores). Vamos falar sobre operadores aritméticos e de atribuição nesta parte. Operadores aritméticos são usados para executar operações matemáticas como adição, subtração, multiplicação etc.

```
x = 14
y = 4

# Adicao
print('x + y =', x+y) # saida: x + y = 18

# Subtracao
print('x - y =', x-y) # saida: x - y = 10

# Multiplicacao
print('x * y =', x*y) # saida: x * y = 56
```

```
# Divisao
print('x / y =', x/y) # saida: x / y = 3.5

# Divisao Inteira
print('x // y =', x//y) # saida: x // y = 3

# Resto da divisao
print('x % y =', x%y) # saida: x % y = 2

# Exponenciacao
print('x ** y =', x**y) # saida: x ** y = 38416
```

Operadores

Operadores de atribuição são usados para atribuir valores a variáveis. Você já viu o uso do = (operador). Vamos tentar mais alguns operadores de atribuição.

```
x = 5

# x += 5 ----> x = x + 5
x +=5
print(x) # saida: 10

# x /= 5 ----> x = x / 5
x /= 5
print(x) # saida: 2.0
```

Outros operadores de atribuição comumente utilizados: -=, *=, %=, //= e **=.

Entrada de Dados

No Python, você pode usar a função input() para receber informações do usuário. Por exemplo:

```
inputString = input('Digite uma frase:')
print('A frase digitada foi:', inputString)
```

Comentários

Existem 3 formas de criar comentários em Python

```
# Este é um comentário

"""Isto é um
   multilinha
   Comente."""

'''Este também é um
    multilinha
    Comente.'''
```

Mudando o tipo de valor

O processo de conversão do valor de um tipo de dados (inteiro, string, float etc.) para outro é chamado de conversão de tipo. Python tem dois tipos de conversão de tipos.

Conversão implícita de tipo

A conversão implícita não precisa de nenhum envolvimento do usuário. Por exemplo:

```
num_int = 123  # Tipo inteiro
num_flo = 1.23 # Tipo float

num_new = num_int + num_flo

print("Valor da variavel num_new:",num_new)
print("tipo de dado da variavel num_new:",type(num_new))
```

Aqui, **num_new** possui um tipo de dado flutuante, porque o Python sempre converte um tipo de dado menor para outro, para evitar a perda de dados.

Mudando o tipo de valor

Aqui está um exemplo em que o interpretador Python não pode implicitamente fazer a conversão.

```
num_int = 123    # Tipo Inteiro
num_str = "456"  # Tipo String

print(num_int+num_str)
```

Quando você executa o programa, você receberá a mensagem:

TypeError: unsupported operand type(s) for +: 'int' and 'str'.

No entanto, o Python tem uma solução para esse tipo de situação, conhecida como conversão explícita.

Mudando o tipo de valor

No caso de conversão explícita, você converte o tipo de dados de um objeto no tipo de dados necessário. Usamos funções predefinidas como int() , float() , str() etc. para executar conversão de tipo explícita. Por exemplo!

```
num_int = 123  # Tipo Inteiro
num_str = "456" # Tipo String

# conversão explicita
num_str = int(num_str)

print(num_int+num_str)
```

Tipos numéricos de Python

O Python suporta números inteiros, números de ponto flutuante e números complexos. Eles são definidos como int, float e complex, e são classes em Python. Além disso, os booleanos: True e False são um subtipo de números inteiros.

```
# Output: <class int>
print(type(5))

# Output: <class float>
print(type(5.0))

# <class complex>
c = 5 + 3j
print(type(c))
```

Estruturas de Dados Python

O Python oferece uma variedade de tipos de dados compostos, geralmente chamados de sequências. Você aprenderá sobre esses tipos internos.

Listas
Tuplas
Strings
Sets (Conjuntos)
Dicionários

Conceito Importante (Posição e Indíce)

Um conceito Importante que você precisa ter daqui para frente

```
#Nome

Item      -->   H  E  N  R  I  Q U E
Tamanho   -->   1  2  3  4  5  6  7 8

Indíce    -->   0  1  2  3  4  5  6  7
Item      -->   H  E  N  R  I  Q  U  E
```

Listas

Uma lista é criada colocando todos os itens (elementos) dentro de um colchete [ ] separado por vírgulas. Pode ter qualquer número de itens e eles podem ser de tipos diferentes (número inteiro, número flutuante, sequência etc.)

```
# lista vazia
minha_lista = []
# lista de inteiros
minha_lista = [1, 2, 3]
# lista com tipos de dados variados
minha_lista = [1, "Hello", 3.4]
```

Você pode usar a função list() para criar listas.

```
minha_lista = list([1,2,3])
```

Listas

Veja como você pode acessar os elementos de uma lista.

```
idioma= ["Franças", "Alemão", "Inglês", "Russo"]

# Acessando o primeiro elemento
idioma[0]
print(idioma[0])
# Acessando o quarto elemento
print(language[3])
```

Você pode usar a função list() para criar listas.

```
minha_lista = list([1,2,3])
```

Listas

Você usa o operador de índice [ ] para acessar um item em uma lista. O índice começa em 0. Portanto, uma lista com 10 elementos terá um índice de 0 a 9.

O Python também permite indexação negativa para suas sequências. O índice de -1 refere-se ao último item, -2 ao penúltimo item e assim por diante.

```
#Nome
Item        -->   H  E  N  R  I  Q  U  E
Posição     -->   1  2  3  4  5  6  7  8

Indíce      -->   0  1  2  3  4  5  6  7
Item        -->   H  E  N  R  I  Q  U  E
Indíce      -->   -8 -7 -6 -5 -4 -3 -2 -1
```

Tuplas

A tupla é semelhante a uma lista, exceto que você não pode alterar os elementos de uma tupla uma vez definida. Enquanto em uma lista, os itens podem ser modificados.

Basicamente, a lista é mutável, enquanto a tupla é imutável.

```
language = ("French", "German", "English", "Polish")
print(language)
```

Tuplas

Você também pode usar a função tuple( ) para criar tuplas.

```
t = tuple([1, 4, 6])
print('t=', t2)
```

Você pode acessar elementos de uma tupla de maneira semelhante a uma lista.

```
language = ("French", "German", "English", "Polish")

print(language[1]) #Saída: German
print(language[3]) #Saída: Polish
print(language[-1]) #Saída: Polish
```

String

Uma sequência é uma sequência de caracteres. Aqui estão diferentes maneiras de criar uma string.

```
# Todas as sequências abaixo são equivalentes
minha_string = 'Hello'
print(minha_string)

minha_string = "Hello"
print(minha_string)

minha_string = '''Hello'''
print(minha_string)

# A sequência de aspas triplas pode extender multiplas linhas
minha_string = """Hello, bem vindo ao
                  mundo do Python"""
print(minha_string)
```

Strings

Você pode acessar caracteres individuais de uma sequência usando a indexação (de maneira semelhante, como listas e tuplas).

```
str = 'programar'
print('str = ', str)

print('str[0] = ', str[0]) # saída: p

print('str[-1] = ', str[-1]) # saida: r
```

Strings

Strings são imutáveis. Você não pode alterar elementos de uma sequência depois de atribuída. No entanto, você pode atribuir uma sequência a outra.

A concatenação é provavelmente a operação de string mais comum. Para concatenar cadeias, use o operador + Da mesma forma, o operador * pode ser usado para repetir a sequência por um determinado número de vezes.

```
str1 = 'Hello '
str2 ='World!'

# saída: Hello World!
print(str1 + str2)

# Hello Hello Hello
print(str1 * 3)
```

Sets (Conjuntos)

Um conjunto é uma coleção não ordenada de itens em que cada elemento é único (sem duplicados).

Aqui está como você cria conjuntos em Python.

```
# Conjunto de Inteiros
meu_set = {1, 2, 3}
print(meu_set)

# Conjunto variado
meu_set = {1.0, "Hello", (1, 2, 3)}
print(meu_set)
```

Sets (Conjuntos)

Você também pode usar a função set( ) para criar conjuntos.

```
meu_set = set('Python')
print(ms)
```

Vamos tentar algumas operações de conjunto comumente usadas:

```
A = {1, 2, 3}
B = {2, 3, 4, 5}

print(A | B) # A união com B - Saída: {1, 2, 3, 4, 5}

print (A & B) # A intersecção com B - Saída: {2, 3}

# Diferença dos conjuntos
print (A - B) # Diferença em Quantidade de Itens: {1}

# Diferença Simetrica
print(A ^ B)  # Saída: {1, 4, 5}
```

Dicionários

Dicionário é uma coleção não ordenada de itens. Enquanto outros tipos de dados compostos têm apenas valor como um elemento, um dicionário tem um par de chave:valor. Por exemplo:

```
#dicionário vazio
meu_dict = {}

# dicionario com chaves inteiras
meu_dict = {1: 'maça', 2: 'bola'}

# dicionario com mix de chaves
meu_dict = {'nome': 'John', 1: [2, 4, 3]}
```

Dicionário

Você também pode usar a função dict( ) para criar dicionários.

```
pessoa = {'nome':'Jack', 'idade': 26, 'salario': 4534.2}
print(pessoa['idade']) # saída: 26

numeros = dict(x=5, y=0)
print('numeros =', numeros)
print(type(numberos))
```

RANGE

range() retorna uma sequência imutável de números entre o número inteiro inicial especificado e o número inteiro final.

```
print(range(1, 10)) # saida: range(1, 10)
```

A saída é iterável e você pode convertê-la em lista, tupla, conjunto e assim por diante. Por exemplo:

```
numeros= range(1, 6)

print(list(numeros)) # saída: [1, 2, 3, 4, 5]
print(tuple(numeros)) # saída: (1, 2, 3, 4, 5)
print(set(numeros)) # saída: {1, 2, 3, 4, 5}

# saída: {1: 99, 2: 99, 3: 99, 4: 99, 5: 99}
print(dict.fromkeys(numeros, 99))
```

RANGE

Omitimos o parâmetro opcional step range()nos exemplos acima. Quando omitido, o step padrão é 1. Vamos tentar alguns exemplos com o parâmetro step

```
#Equivalente: numeros = range(1, 6)
numeros1 = range(1, 6 , 1)
print(list(numeros1 )) # saida: [1, 2, 3, 4, 5]

numeros2 = range(1, 6, 2)
print(list(numeros2)) # saida: [1, 3, 5]

numeros3 = range(5, 0, -1)
print(list(numeros3)) # saida: [5, 4, 3, 2, 1]
```

Fluxo de controle Python - Instrução if ... else

A instrução if...else é usada se você deseja executar ações diferentes (executar código diferente) em diferentes condições. Por exemplo:

```
num = -1

if num > 0:
    print("Número positivo")
elif num == 0:
    print("Zero")
else:
    print("Número negativo")
```

Instrução if ... else

Pode haver zero ou mais elif , e o else é opcional. A maioria das linguagens de programação usa { } para especificar o bloco de código. Python usa indentação.

Um bloco de código começa com recuo e termina com a primeira linha não recuada. A quantidade de recuo depende de você, mas deve ser consistente ao longo desse bloco. Geralmente, quatro espaços em branco são usados para indentação e são preferidos às guias. Vamos tentar outro exemplo:

```
if False:
  print("Estou dentro do if")
  print("Eu também estou dentro do if")
print("Eu estou fora do if")

# saida: Eu estou fora do if
```

Loop While

Como a maioria das linguagens de programação, o while loop é usado para iterar sobre um bloco de código, desde que a expressão de teste (condição) seja true. Aqui está um exemplo para encontrar a soma dos números naturais:

```
n = 100

# inicializar soma e contador
soma = 0
i = 1

while i <= n:
    soma = soma + i
    i = i+1    # contador de atualização
print("A soma é:", soma)

# Saida: A soma é:5050
```

Loop FOR

No Python, o loop for é usado para iterar sobre uma sequência (lista, tupla, string) ou outros objetos iteráveis. A iteração sobre uma sequência é chamada de travessia.

Aqui está um exemplo para encontrar a soma de todos os números armazenados em uma lista.

```
numeros = [6, 5, 3, 8, 4, 2]

soma = 0

# iterar sobre a lista
for valor in numeros:
  soma = soma+valor

print("A soma é:", soma) # saida: A soma é: 28
```

Loop FOR

observe o uso do operador in no exemplo acima. O operador in retorna True se valor/variável for encontrado na sequência.

Declaração break

A instrução break termina o loop que a contém. O controle do programa flui para a instrução imediatamente após o corpo do loop. Por exemplo:

```
for val in "string":
    if val == "r":
        break
    print(val)

print("Fim")
```

Declaração Continue

A instrução continue é usada para ignorar o restante do código dentro de um loop apenas para a iteração atual. O loop não termina, mas continua com a próxima iteração. Por exemplo:

```
for val in "string":
    if val == "r":
        continue
    print(val)

print("Fim")
```

Declaração Pass

Suponha que você tenha um loop ou uma função que ainda não foi implementada, mas deseja implementá-la no futuro. A função não pode ter um corpo vazio. O interpretador reclamaria. Então, você usa a instrução pass para construir um corpo que não faz nada.

```
sequence = {'p', 'a', 's', 's'}
for val in sequence:
    pass
```

Funções

Uma função é um grupo de instruções relacionadas que executam uma tarefa específica. Você usa a palavra-chave def para criar funções no Python.

```
def print_linhas():
  print("Eu sou a linha 1.")
  print("Eu sou a linha 2.")
```

Você precisa chamar a função para executar os códigos dentro dela. Aqui está como:

```
def print_linhas():
  print("Eu sou a linha 1.")
  print("Eu sou a linha 2.")

# chamando a função
print_linhas()
```

Funções

Uma função pode aceitar argumentos.

```
def soma_numeros(a, b):
  soma = a + b
  print(soma)

soma_numeros(4, 5)

# saída: 9
```

Função

Você também pode retornar o valor de uma função usando a instrução return.

```
def soma_numeros(a, b):
  soma = a + b
  return soma

resultado = soma_numeros(4, 5)
print(resultado)

# saida: 9
```

Módulos

Módulos referem-se a um arquivo que contém instruções e definições Python.

Um arquivo contendo código Python, por exemplo:, exemplo.py é chamado de módulo e seu nome seria exemplo.

Vamos criá-lo e salvá-lo como exemplo.py.

```
#modulo Python exemplo
def soma(a, b):
    return a + b
```

Módulos

Para usar este módulo, usamos a palavra-chave **import**

```
#importando o módulo exemplo
import exemplo
# acessando a funcao contida no módulo usando o operador .

exemplo.soma(4, 5.5)
```

Módulos

O Python tem uma tonelada de módulos padrão prontamente disponíveis para uso. Por exemplo:

```
import math

resultado = math.log2(5) # retorna o logaritmo base-2
print(resultado) # saída: 2.321928094887362
```

Módulos

Você pode importar nomes específicos de um módulo sem importar o módulo como um todo. Aqui está um exemplo.!

```
from math import pi
print("O valor de pi é:", pi)

# saida: O valor de pi é: 3.1415
```

Módulos

Você pode importar nomes específicos de um módulo sem importar o módulo como um todo. Aqui está um exemplo.!

```
from math import pi
print("O valor de pi é:", pi)

# saida: O valor de pi é: 3.1415
```

Trabalhando com arquivos em Python

Uma operação de arquivo ocorre na seguinte ordem.

1. Abra um arquivo
2. Leitura ou gravação (executar operação)
3. Feche o arquivo

Como abrir o arquivo?
Você pode usar a função open() para abrir um arquivo.

```
f = open("teste.txt")   # abre o arquivo teste.txt
f = open("C:/Python33/README.txt") # especificar o
caminho.
```

Arquivos

Podemos especificar o modo ao abrir um arquivo.

Modo	Descrição
'r'	Abra um arquivo para leitura. (padrão)
'W'	Abra um arquivo para gravação. Cria um novo arquivo se ele não existir ou truncará o arquivo, se ele existir.
'x'	Abra um arquivo para criação exclusiva. Se o arquivo já existir, a operação falhará.
'a'	Abra para anexar no final do arquivo sem truncá-lo. Cria um novo arquivo se ele não existir.
't'	Abra no modo de texto. (padrão)
'b'	Abra no modo binário.
'+'	Abra um arquivo para atualização (leitura e gravação)

Arquivos em Python

```
f = open("teste.txt")     # equivalente to 'r' or 'rt'
f = open("teste.txt",'w')  # escreve em modo texto
f = open("img.bmp",'r+b') # le e grava em modo binário
```

Arquivos

Como fechar um arquivo?
Para fechar um arquivo, você usa o close()método

```
f = open("teste.txt",encoding = 'utf-8')
# executar operações dentro do arquivo
f.close()
```

Arquivos

Como gravar em um arquivo?
Para gravar em um arquivo em Python, precisamos abri-lo no modo de gravação 'w', acrescentar o 'a' ou criação exclusiva 'x'.

```
with open("teste.txt",'w',encoding = 'utf-8') as f:
    f.write("meu primeiro arquivo\n")
    f.write("Este arquivo\n\n")
    f.write("contém 3 linhas\n")
```

Arquivos

Como ler arquivos?
Para ler um arquivo no Python, você deve abrir o arquivo no modo de leitura. Existem vários métodos disponíveis para esse fim. Podemos usar o read(size)método para ler o tamanho do número de dados.

```
f = open("teste.txt",'r',encoding = 'utf-8')
f.read(4)   # le os 4 primeiros dados
```

Diretórios

Um diretório ou pasta é uma coleção de arquivos e subdiretórios. O Python possui o módulo os , que fornece muitos métodos úteis para trabalhar com diretórios e arquivos.

```
import os
os.getcwd()  // diretorio atual
os.chdir('D:\\Hello') // muda o diretório para Hello
os.listdir()  // lista todos os subdiretorios e arquivos no caminho
os.mkdir('teste') // cria um diretorio de nome teste
os.rename('teste','testi') // renomeia o diretorio para testi
os.remove('velho.txt')  // apaga o arquivo velho.txt
```