



```

1 |     print(f"Status: {response.status_code} - Try rerunning the code!")
2 | else:
3 |     print(f"Status: {response.status_code}\n")
4 |
5 |     # using BeautifulSoup to parse the response object
6 |     soup = BeautifulSoup(response.content, "html.parser")
7 |
8 |     # Post images in the soup
9 |     for img in soup.findAll("img", attrs={"alt": "Post image"}):

```

PYTHON (UDEMY)

<https://www.udemy.com/course/python-3-do-zero-ao-avancado/learn/lecture/15099532#overview>

● Seção 01: Introdução

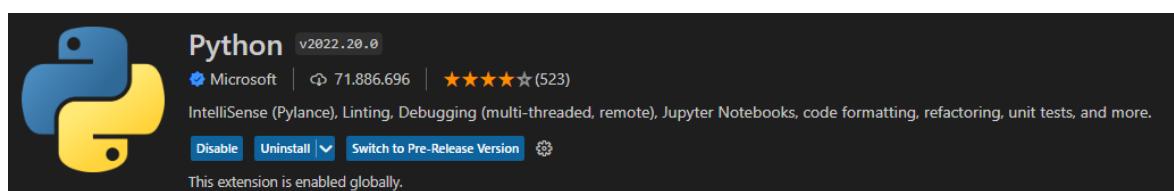


● Seção 02: Python e VSCode

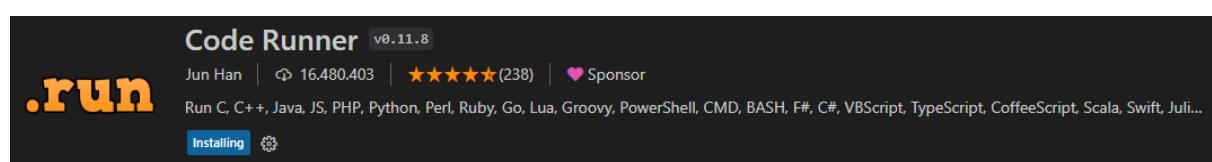


Extensões

Python



Code Runner



```

// settings.json
{
    "code-runner.runInTerminal": true,
    "code-runner.clearPreviousOutput": true,
    "code-runner.executorMap": {
        "python": "cls ; python -u",
    },
    "code-runner.ignoreSelection": true
}

```

Settings.json (configurações do VS Code)

```
// settings.json
{
    "editor.fontSize": 18,
    "explorer.compactFolders": false,
    "workbench.iconTheme": "material-icon-theme",
    "workbench.colorTheme": "Nord",
    "code-runner.runInTerminal": true,
    "code-runner.clearPreviousOutput": true,
    "code-runner.executorMap": {
        "python": "cls ; python -u",
    },
    "code-runner.ignoreSelection": true,
    "python.defaultInterpreterPath": "python",
    "diffEditor.wordWrap": "off",
    "editor.wordWrap": "on"
}
```

● Seção 03: Introdução ao Python e Lógica de programação



Extensão do arquivo: file.py

Comentário: “# comentários”

DocString: utilizado para comentar multi linhas, lembrando que não é um comentário. O interpretador lê, mas não faz nada com essas anotações, apenas guarda na memória.

```
"""
docstrings (aspas duplas)
"""
# ou
'''
docstrings (aspas simples)
'''
```

print(): por padrão, o print ja adiciona um separado entre os argumentos, mas ele pode ser alterado utilizando o sep.

```
print(12, 34) # 12 34 (padrão)
print(12, 34, sep = "-") # 12-34
```

Tambem é possivel alterar o que ocorre no final do comando print, com o end.

```
# \r\n -> CRLF (padrão windows)
# \n -> LF (padrão unix)

# \n (padrão)
print(56, 78, sep = "-", end = '\n')
print(78, 1011, sep = "-", end = '\n')

"""
saída
56-78
78-1011
"""

# outros caracteres
print(56, 78, sep = "-", end = '##')
print(78, 1011, sep = "-")

"""
saída
```

```
56-78##78-1011
"""


```

■ Tipos de Dados

O python é linguagem de programação **DINÂMICA** (a linguagem já sabe o tipo da informação que está sendo passada.) e **FORTE**.

Strings → str

```
"rafael" # o python reconhece que isso é uma string
```

caractere de escape: \

```
print("Rafael \"Oliveira") # Rafael "Oliveira
"""\\" -> diz para não interpretar o próximo caractere ir
ao próximo. """
```

E se a \ precisa ser mostrada? Utiliza-se o \r

```
print(r"Rafael \"Oliveira\\") # Rafael \"Oliveira\\"
```

Uma maneira mais simples, é utilizar aspas duplas dentro de aspas simples, ou ao contrário.

```
print('Rafael "Oliveira") # Rafael "Oliveira"
```

Int e Float

int: números inteiros, positivos e negativos → [10](#)

float: números com ponto flutuante (casas decimais), positivos e negativos → [10.1](#)

Para saber o tipo de dado, utiliza-se a função [type](#)

```
print(type(10)) # <class 'int'>
print(type(10.1)) # <class 'float'>
print(type("Rafael")) # <class 'str'>
```

Boolean → Bool

Duas respostas possíveis, [True\(sim\)](#) ou [False\(não\)](#)

```
print(10 == 10) # Sim -> True
print(10 == 11) # Não -> False
print(type(10 == 10)) # <class 'bool'>
```

Coerção/Conversão de tipos

Converter um tipo de dado para outro, também conhecido como: type conversion, typecasting, coercion.

Str, int, float e bool são tipos **primitivos** e **imutáveis** (não podem ser alterados ao longo do programa)

Existem funções que convertem um tipo em outro

```

print(1 + 1) # 2
print('oi' + ' Rafael') # oi Rafael

print('1' + 1) # erro de tipos, pois nao se pode concatenar str com int, somente str com str

# str -> int
int('1') # -> 1
print(int('1') + 1) # 2

# str -> float
float('1') # -> 1.0
print(float('1') + 1) # 2.0

# str -> bool
print(bool('')) # -> False -> str vazia é considerada False
print(bool(' ')) # -> True -> str com conteúdo é considerada True

# int -> str
str(11) # -> '11'
print(str(11) + 'b') # 11b

```

■ Variáveis

São usadas para **salvar algo na memória do computador**. A **PEP8** (guia de estilos do python) indica que o nome das variáveis sejam com letras minúsculas, podendo utilizar números e *underline*.

O sinal de `=` é um **operador de atribuição**, ou seja, atribui um valor à variável.

Ex: `nome_variavel = expressao`

```

nome_completo = "Rafael Oliveira" # str
idade = 18 # num int
maior_de_idade = idade >= 18 # expressao

print('Nome: ', nome_completo, 'Idade: ', idade, 'Maior de Idade: ', maior_de_idade) # Rafael Oliveira 18

```

■ Operadores Aritméticos (matemáticos)

```

adicao = 10 + 12 # 22
subtracao = 22 - 10 # 12
multiplicacao = 10 * 2 # 20
divisao = 10 / 3 # 3.33333 (o retorno sempre é float)
divisao_inteira = 10 // 3 # 3.0
exponenciacao = 2 ** 10 # 1024
modulo = 55 % 2 # 1 (resto da divisão)

```

(+) Concatenação

"Apenas strings" `+` "devem ser concatenadas" `+` "com strings"

```

# concatenação (+)
concatenacao = 'A' + 'B' + 'C'
print(concatenacao) # ABC

```

(*) Repetição

Um inteiro `*` "é a string que deve ser repetida"

```

# repeticao (*)
r_dez_vezes = 'R' * 10
print(r_dez_vezes) # RRRRRRRRRR

```

```
rafael_duas_vezes = 'Rafael' * 2
print(rafael_duas_vezes ) # RafaelRafael
```

Precedência entre os operadores aritméticos

Cada operador é executado de acordo com sua precedência, ou seja, sua hierarquia.

```
# 1 - parenteses (n + n)
# 2 - exponenciacao **
# 3 - * / // % (com multiplicação e divisão na mesma conta, a ordem sera da esquerda pra direita)
# 4 - + -
```

```
conta_1 = 1 + 1 ** 5 + 5 # 1 ** 5 = 1 -> +1 = 2 -> +5 = 7
conta_2 = (1 + 1) ** (5 + 5) # 1024

print(conta_1)
print(conta_2)
```

... → Ellipsis, usado pra indicar algum código que será escrito

```
altura = 1.85
peso = 90
imc = ...
```

Formatação de Strings (f-strings)

utilizando o `f'texto {variavel}'`

```
texto = f'{nome}, tem {altura} de altura\npesa {peso} quilos e seu IMC é {imc}'
```

quantas **cotas decimais depois da vírgula** → `{variavel:.2f}` → **duas casas decimais** → **1.85**

```
texto = f'{nome}, tem {altura:.2f} de altura\npesa {peso} quilos e seu IMC é {imc}'
print(texto)

# com vírgula
valor = 10050.4
print(valor:,.2f) # -> 100,050.4 -> cem mil e 50
```

utilizando o `.format()`

Tudo em python é um objeto, e objeto **têm métodos dentro dele**, ou seja, algumas ações que podem ser realizadas, por exemplo, as strings tem o método `format()`

```
a = 'A'
b = 'B'
c = 1.1

formato = 'a = {} b = {} c = {:.2f}'.format(a, b, c)
print(formato) # -> a = A b = B c = 1.10
# a,b e c são argumentos
# seguindo a ordem, dentro da primeira chave (a = {}) vai vir o
# primeiro argumento passado em format(), que no caso, é o a.

# ou
string = 'a = {} b = {} c = {}'
formato = string.format(a, b, c)
print(formato) # -> a = A b = B c = 1.1

# utilizando os índices
string = 'a = {0} a = {0} a = {0} b = {1} c = {2}'
formato = string.format(a, b, c)
```

```

print(formato) # -> a = A a = A a = A b = B c = 1.1

# erro
# string = '\na = {} b = {} c = {} {}'
# formato = string.format(a, b, c)
# daria um erro de "index out of range", pois esta buscando algo que já acabou
# foram passadas 4 chaves (a = {} b = {} c = {} {}), para 3 argumentos (a, b, c), a ultima chave, ficaria sem argumento

# utilizando parametros nomeados
string = 'a = {nome1} b = {nome2} c = {nome3}'
formato = string.format(nome1 = a, nome2 = b, nome3 = c)
print(formato) # -> a = A b = B c = 1.1

```

■ Input

Para **receber dados do usuário**, utiliza-se a função `input()`. E os inputs **sempre vão retornar uma str**, independente do que for digitado.

```

nome = input('Qual o seu nome? ') # usuário digitou 'Rafael'

print(f'O seu nome é {nome}') # O seu nome é Rafael
print(f'O seu nome é {nome=}') # O seu nome é nome='Rafael'

```

```

# números no input
num1 = input('Digite um número: ') # 10
num2 = input('Digite outro número: ') # 5
print(f'A soma dos números é: {num1 + num2}') # 105 -> como são duas strs, ele vai concatenar, e não somar

int_num1 = int(num1)
int_num2 = int(num2)
print(f'A soma dos números é: {int_num1 + int_num2}') # 15

```

E porque não converter direto no input? O código poderia ser quebrado antes mesmo de ser realizada alguma verificação. **Pois, se o usuário digita “a” por exemplo, o código já retornaria um erro, pois não tem como converter “a” em inteiro.** O mais recomendado é usar uma estrutura condicional (if / elif) depois que o usuário digitar para verificar se ele realmente digitou um número.

■ Estrutura condicionais (if / elif ... else)

Com as estruturas condicionais, é **possível mudar o fluxo do código**.

```

entrada = input('Você quer "entrar" ou "sair"? ')

if entrada == 'entrar':
    print('Você entrou no sistema')
elif entrada == 'sair':
    print('Você saiu do sistema')
else:
    print(f'A opção "{entrada}" é inválida. Digite "entrar" ou "sair".')

```

■ Operadores relacionais (comparativos)

Servem pra **fazer comparações**, sempre retornando `True` ou `False`.

```

maior = 2 > 1 # True
maior_ou_igual = 2 >= 2 # True
menor = 4 < 2 # False
menor_ou_igual = 2 <= 2 # True
igual = 'r' == 'r' # True
diferente = 'a' != 'a' # False

```

■ Operadores lógicos

And (e)

Todas as condições precisam ser verdadeiras. Ou seja, se qualquer valor for considerado falso, toda a expressão é dada como falsa.

```
entrada = input('[E]ntrar [S]air: ')
senha_digitada = input('Senha: ')

senha_correta = '1234'

if entrada == 'E' and senha_correta == '1234':
    print('Entrar')
else:
    print('Sair')
```

Valores **considerados falsos** no python (falsys)

```
# (falsys)
0
0.0
''
False
None
```

O python **checa valor por valor, quando ele se depara com um False**, no caso do operador and, **ele para e retorna o False**. Isso gera uma economia de recurso.

```
# Avaliação de curto circuito
print(True and False and True) # retorno -> False
print(True and 0 and True) # retorno -> 0
```

Or (ou)

Qualquer valor verdadeiro, considera a expressão toda verdadeira.

```
entrada = input('[E]ntrar [S]air: ')
senha_digitada = input('Senha: ')

senha_correta = '1234'

if (entrada == 'E' or entrada == 'e') and senha_correta == '1234':
    print('Entrar')
else:
    print('Sair')
```

O python **checa valor por valor, quando ele se depara com um True**, no caso do operador or, **ele para e retorna o True**. Isso gera uma economia de recurso.

```
# Avaliação de curto circuito
print(True or False or 0) # retorno -> True
print(0 or False or 0.0 or 'r' or True) # retorno -> 'r'
```

Not (negação)

Usado para inverter expressões, ou seja, o que for False vira True, e o que for True vira False.

```

# not True -> False
# not False -> True

senha = input('Senha: ')
if not senha:
    print('você não digitou nada')

print(not True)
print(not False)

```

In e Not in

Esses operadores são **utilizados para verificar se aquele valor está entre alguma expressão**.

Strings são iteráveis, ou seja, pode se navegar item por item.

```

# 0 1 2 3 4 5 -> indices positivos
# R a f a e l
#-6 -5 -4 -3 -2 -1 -> índices negativos

nome = 'Rafael'
print(nome[2]) # -> f
print(nome[-2]) # -> e

# Nesse caso, o python vai checar letra pro letra para verificar se o 'R' está entre as letras do nome
print('R' in nome) # True
print('y' in nome) # False
print('afa' in nome) # True
print('ele' not in nome) # True

# ex
nome = input('Digite um nome: ')
encontrar = input('Digite o que deseja encontrar: ')

if encontrar in nome:
    print(f'{encontrar} está em {nome}')
else:
    print(f'{encontrar} não está em {nome}')

```

■ Interpolação básica de strings (%tipo_do_dado)

Formatação de um jeito diferente do format.

```

"""
s - string
d e i - int
f - float
x(gera um hexa minúsculo) e X(gera um hexa maiúsculo) - Hexadecimal (ABCDEF0123456789)
"""

nome = 'Rafael'
preco = 1000.928350334
texto_montado = '%s, o preço é R$%.2f' %(nome, preco)
print(texto_montado) # Rafael, o preço é R$1000.92
print('O HEXADECIMAL de %d é %04X' %(15, 15)) #ou somente %x, colocando %04X ele preenche o que faltar, ou %08X

```

■ Formatação básica de strings

```

"""
Formatação básica de strings utilizando o f strings
s - string
d - int
f - float
.<número de dígitos>f
x ou X - Hexadecimal
(Caractere)(><^)(quantidade)

```

```

> - Esquerda
< - Direita
^ - Centro
= - Força o número a aparecer antes dos zeros
Sinal - + ou -
Ex.: 0>-100,.1f
Conversion flags - !r(__repr__) !s(__str__) !a(__asc__)
"""

variavel = 'ABC'
print(f'{variavel}') # ABC
print(f'{variavel:$>10}') # #####ABC
print(f'{variavel:#<10}') # ABC#####
print(f'.{variavel:^10}.') # . ABC .

print(f'{1000.48342342234:0=+10,.1f}') # +001,000.5
print(f'0 hexadecimal de 1500 é {1500:08X}')

```

■ Fatiamento de strings e função len()

[i: f: p] [: :]

```

# Fatiamento [i: f: p] [ : : ]
# i -> inicio f -> final p -> passo (de quantos em quantos caracteres)

# 012345678
# Ola mundo
#-987654321
variavel = 'Ola mundo'
print(variavel[4:8]) # mund -> nesse caso, o indice final não é incluido, ou seja, o 8, entao esta indo do 4 ao 7
print(variavel[4:]) # mundo -> omitindo o final, ele sabe que tem que ir até o final da string
print(variavel[:5]) # Ola m -> omitindo o inicio, ele sabe que tem que começar no inicio da string

# len, retorna a quantidade de caracteres da string
print(len(variavel)) # 9

#
print(0:len(variavel):1) # Ola mundo
print(0:len(variavel):2) # Oamno
print(-1:-10:-1) # odnum alo

```

Exercício

```

# Exercicio

# Peça ao usuário para digitar seu nome
nome = input('Digite seu nome: ')
# Peça ao usuário para digitar sua idade
idade = input('Digite sua idade: ')

# se nome e idade forem digitados, exiba:
if nome and idade:
    # seu nome é {nome}
    print(f'Seu nome é {nome}')
    # seu nome invertido é {nome invertido}
    print(f'Seu nome invertido é {nome[::-1]}')
    # se o nome contém ou não espaços
    if ' ' in nome: print(f'Seu nome, {nome}, contém espaços')
    else: print(f'Seu nome, {nome}, não contém espaços')
    # seu nome tem {n} letras
    print(f'Seu nome tem {len(nome)} letras')
    # a primeira letra do seu nome é {letra}
    print(f'A primeira letra do seu nome é {nome[0]}')
    # a última letra do seu nome é {letra}
    print(f'A ultima letra do seu nome é {nome[len(nome)-1]}') # ou nome[-1]
# se nada for digitado em nome ou idade:
else:
    print('Desculpe, você deixou campos vazios.')

```