

Dependências ou Pacotes

→ Uma boa prática de programação é instalar pacotes de forma local, ou seja, de globalmente, em "virtual environment"

Criando virtual environment (Windows)

```
py -3 -m venv .venv  
.venv\scripts\activate
```

Instalando novo pacote

```
python -m pip install [pacote]
```

Mílio do Vídeo

Trabalhando com String

Concatenando

x = "a"

```
print("Olá" + x)
```

→ Olá a

Quebra de linha

```
print("primeira linha\nSegunda linha")
```

primeira linha
Segunda linha

quando adicionarmos \ e em seguida um caractere o python entende que o próximo caractere deve ser escrito de forma literal

```
ex: print("a \"b")
```

→ a"b

frase = "Exemplo"
print(frase.lower()) → exemplo
print(frase.upper()) → EXEMPLO
print(frase.isupper()) → false (true se for inteira maiúscula)
print(len(frase)) → 7
print(frase[0]) → E
frase = "um dois"
print(frase.index("dois")) → 3
print(frase.replace("um", "tres")) → tres dois

Tipos de variáveis São definidos na declaração

X = 1 | int
X = 1.01 | float
X = "1" | string
X = true | bool

Manipulando Números

`print(abs(-5))` → 5
`print(pow(2, 4))` → 16 ou $2^{**} 4$
`print(min(5, 6))` → 5 {`max()` → 6}
`print(round(3, 4))` → 3

`from math import * | modulo math`

`print(floor(3.7))` → 3 | arredonda pra baixo
`print(ceil(3.4))` → 4 | arredonda pra cima
`sqrt()` → raiz quadrada

Recebendo Input do Usuário

`input()` → espera um valor do usuário

`nome = input("Qual é seu nome")`

→ Qual é seu nome -

* O python entende automaticamente o valor inserido como um string

Transformando tipos de Var

Num 1 = `input()`: 3 → String
Num 2 = `input()`: 4 → String

`print(Num1 + Num2) → 34`
`print(int(Num1) + int(Num2)) → 7`

`float()` / funcionamento análogo
`str()`

List

Declarando

exemplo = [] { ex = 5 * [0] → [0, 0, 0, 0, 0]
exemplo = [true, "dois", 3, 4]

Em Python não importa o tipo da variável para list

Acessando

`print(exemplo[1]) → dois`

List pode ser lida de trás para frente:

`print(exemplo[-1]) → 4`

Pode se ler a partir de uma posição

Print exemplo [1:3] → ['dois', 3]

Começa na pos 1 ignorando-a e para antes da posição 3

funções da List

Lucky_Numbers = [4, 8, 15, 16, 23, 42]
friends = ["Keren", "Karen", "Jim", "Crean", "Toby"]

Print(friends. **extends**(Lucky_Numbers))

→ ['Keren', 'Karen', 'Jim', 'Crean', 'Toby', 4, 8, 15, 16, 23, 42]

Print(friends. **append**("Creed")) | Adiciona no fim

→ ['Keren', 'Karen', 'Jim', 'Crean', 'Toby', 'Creed']

Print(friends. **insert**(1, "Kelly"))

→ ['Keren', 'Kelly', 'Karen', 'Jim', 'Crean', 'Toby']

Print(friends. **remove**("Jim"))

→ ['Keren', 'Kelly', 'Karen', 'Crean', 'Toby']

Print(friends. **clear**())

→ []

`Print(friends.pop())` | retorna o último

$\rightarrow ['Keren', 'Karen', 'Jim', 'Oscar']$

`Print(friends.index("Oscar"))`

$\rightarrow 3$

* também é útil para saber se o elemento pertence a lista, já que retorna um erro caso não pertença

`friends = ["Keren", "Karen", "Jim", "Jim", "Oscar", "Toby"]`

`Print(friends.count("Jim"))`

$\rightarrow 2$

`Print(friends.sort())` | Ordena em Ordem Alfabética

$\rightarrow ['Jim', 'Jim', 'Karen', 'Keren', 'Oscar', 'Toby']$

`Lucky_Numbers.sort()` Ordem Crescente

`Print(Lucky_Numbers.reverse())`

$\rightarrow [23, 16, 15, 8, 42]$

`friends2 = friends.copy()`

* `friends2 = friends`

Tuples

Data Structure (Container) que pode armazenar diferentes repositórios.
São imutáveis.

Ex:

Tuples

Coordenadas = (4, 5)

* Não acessávamos assim como list

* Pode-se criar list de tuples: [(4, 5), (6, 7)]

Funções

Tuples em python têm o seu escopo definido pela identação

def exemplo():

 in

 in

out

* "def exemplo():": Cria o escopo da definição da função "exemplo()" chama ela.

* ao criar parâmetros não é necessário identificar o tipo de retorno

Return Statement

Em python não é declarado o return da função na sua declaração

```
def exemplo(num)
    return num * num
```

if Statement

* não se usa parênteses na declaração e o escopo é definido pelo indentação. assim para todos como while

```
is_male = true  
is_tall = true
```

* not() inverte o bool

```
if is_male and is_tall:
```

(code)

```
elif is_male and not(is_tall):
```

(code)

```
else:
```

(code)

(out)

Dictionaries

Data structure armazena informações em Key value pairs. Permite criar vários Key values, mas é quando queremos acessar um pedaço específico de informação, dentro do Dictionary, é só aceder-lhe por um 'Key'

"Key": "Value"

Def:

monthConversion = {
 "Jan": "January",
 "Feb": "February",
 ...
}

* Key deve ser único

Viro:

```
print(monthConversion["Jan"]) → January  
print(monthConversion.get("lwo", "Not a valid Key"))  
→ Not a valid Key
```

For loop

For letter in "Giraffe":
print(letter)

-> G
i
r
a
f
f
e

friends = ["Jim", "Karen", "Kerim"]

For friend in friends:
print(name)

-> Jim
Karen
Kerim

For index in range(3):
print(index)

-> 0
1
2

* range() retorna uma
sequência de números, que
por default começo em 0, e
incrementa 1 (default) e acaba
quando chegar no número
especificado

range range(1,3)

-> 1
-> 2

[0, 1, 2]

for index in range(len(friends)):
 print(friends[index])

len() returns o tamanho,
neste caso da lista

-> Jim
Karen
Keren

2D List & Nested Loops

number_grid = [
 [1, 2, 3],
 [4, 5, 6],
]

print(number_grid[0][0]) → 0

for row in number_grid:
 for col in row:
 print(col)

Comments

1 linha
''' multiplo linhas (apenas simples)'''

Try / Except

Ex: try:

(Código)

except ZeroDivisionError as err:

print(err)

except ValueError

Reading Files

open("caminho/ arquivo.txt", "r")

? ^{read}

"w" ^{write}

não modifica, apenas acrescenta ← "a" append
ler e escrever ← "r+"

* open() pode ser armazenado em uma variável

arquivo = open()

* lembre de fechar

arquivo.close()

Funções

arquivo.readable() → bool

arquivo.read() → lê o conteúdo completo

arquivo.readline() → lê a primeira linha quando chamado
e move o cursor para a linha seguinte

arquivo.readlines() → Retorna uma lista com todas
as linhas

Writing to Files

Se o arquivo não existir ele criará o arquivo quando
o open() for chamado com a opção 'w'

arquivo = open("caminho/arquivo.txt", "a")

arquivo.write("example")



arquivo.write("exemplo")

arquivo.txt
exemplo exemplo

arquivo.write("/n exemplo")

arquivo.txt
exemplo exemplo
exemplo

arquivo = open("caminho/arquivo.txt", "w")

arquivo.write("exemplo")

arquivo.txt
exemplo

Modules & Pip

Module: arquivo do python que importamos para o arquivo no qual estamos trabalhando

import arquivo

Acessando:

arquivo.{ aqui é possível acessar tudo que for nece-

Wörter {

Perquin List of python modules

python module Index

Python - dock

Pip: "package Manager" instala, desinstala, actualiza
python module

Classe e Objetos

Class Student : initialize function
def __init__(self): "What is the student type in python"

def __init__(self, name, major, gpa):
 data type, todos os atributos associados

self.name = name semelhante ao this de C++ e java

Importando em Outros Arquivos

from Student import Student
(arquivo) (classe)

Student1 = Student("Jim", "Business", 3.1)
print(Student1.name) → Jim

Object Function

Class Student :

```
def __init__(self, name, major, gpa):
```

 self.name = name

 self.major = major

 self.gpa = gpa

```
def on_honor_roll(self):
```

 if self.gpa >= 3.5:

 return true

 else

 return false

* funções que podem ser usadas pelos objetos das classes

Inheritance

from Chef import Chef

Class ChineseChef(Chef):

aqui ChineseChef herda todas as funções e atributos de Chef. E possui "Overwrite" qualquer função

