

Introdução a Python



Semana da Computação UNICAP

Tarcísio Marinho de Oliveira



Walkthrough

`#!/bin/bash/env python`

`# Quem eu sou ?`

`# Conteúdo Programático`



Conteúdo

- # Tipos de variáveis
- # Operadores lógicos
- # Estruturas condicionais
- # Estruturas de repetição
- # Funções
- # Manipulações de Strings
- # Listas
- # Tuplas
- # Dicionários
- # Operações com arquivo
- # Tratamento de exceções



~\$ whoami

Tarcísio Marinho

5 período de CC

Projetos_com_Python = ['Geração de música com redes neurais',
'GonnaCry-Ransomware', 'Backdoor',
'Eliza Assistente pessoal', 'Music-Downloader']

Github = "<https://github.com/tarcisio-marinho>"

Email = "tarcisio_marinho09@hotmail.com"

Facebook = "<https://www.facebook.com/OneDayUmay1>"



O que é Python

Propósito Geral

Alto nível

Interpretada / scripts

Imperativa / Orientada a objetos / Funcional

Tipagem dinâmica

Fracamente tipada

Open Source <3

Python == C

Bibliotecas e frameworks

pip

Garbage collector



Quais áreas Python é utilizado

Inteligência Artificial (Machine Learning – ML)

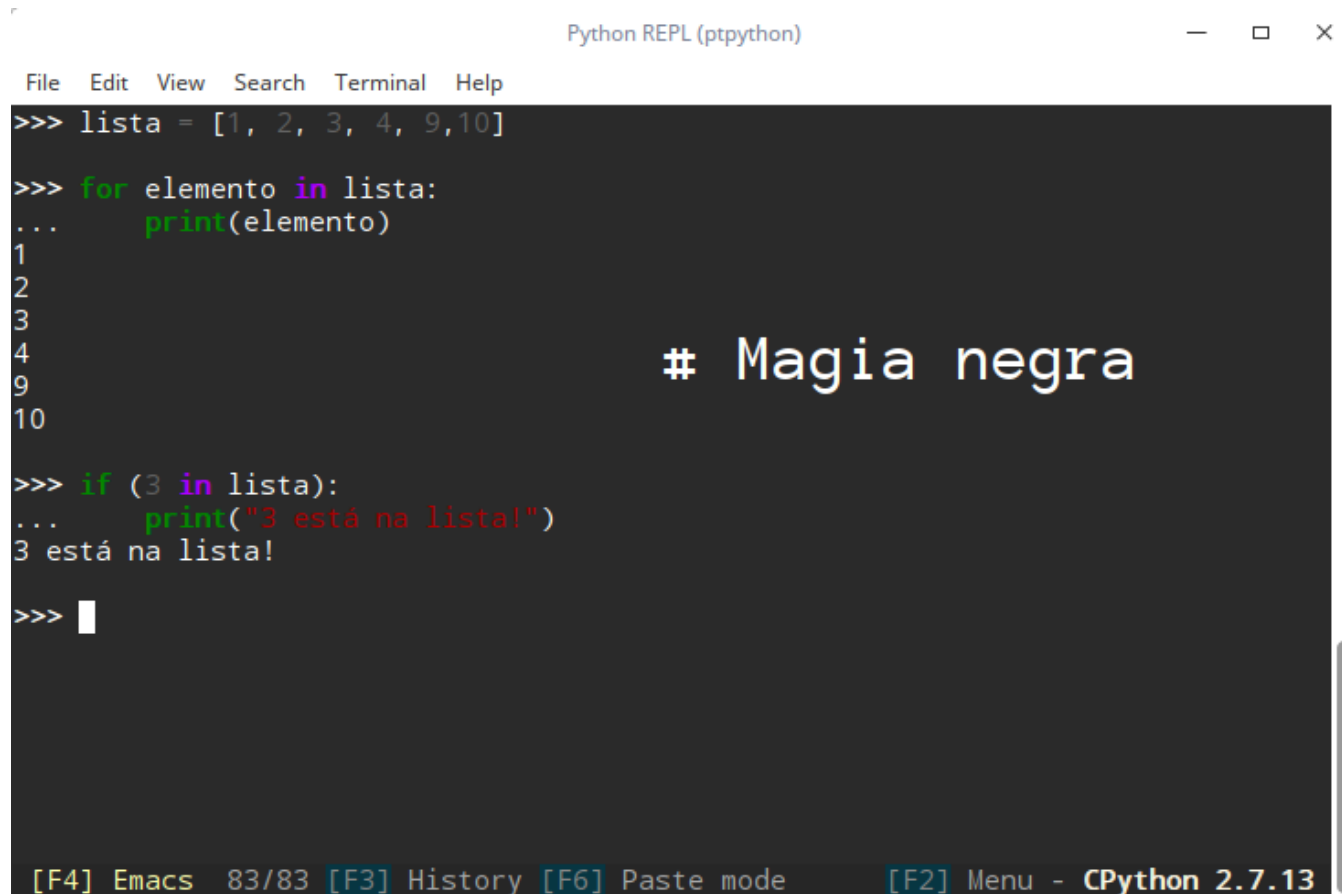
- Tensorflow (Google).
- Keras.
- Pytorch (Facebook).
- Theano.

Data science

- Numpy
- Pandas
- Matplotlib

Security

Python é alto nível



The image shows a screenshot of a Python REPL window titled "Python REPL (ptpython)". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The main area is a dark terminal with light-colored text. The code being executed is as follows:


```
>>> lista = [1, 2, 3, 4, 9,10]
>>> for elemento in lista:
...     print(elemento)
1
2
3
4
9
10
>>> if (3 in lista):
...     print("3 está na lista!")
3 está na lista!
>>> 
```

Overlaid on the right side of the terminal is the text "# Magia negra" in a large, white, serif font.

At the bottom of the window, there is a status bar with the following text: "[F4] Emacs 83/83 [F3] History [F6] Paste mode [F2] Menu - CPython 2.7.13".

Python é interpretada

Python é uma linguagem que quando interpretada, não gera código intermediário “executavel”. Quem interpreta é a maquina virtual do Python.

A terminal window with a dark background and light text. The title bar at the top reads 'fish /home/tarcisio/Desktop/apresentacao-python' and includes standard window controls (minimize, maximize, close). The menu bar shows 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows a prompt 'tarcisio' followed by a blue arrow pointing to the directory '~/D/apresentacao-python', then the command 'python helloworld.py' in green. The output 'Hello World!' is displayed on the next line. A second prompt 'tarcisio' with a blue arrow to the same directory and a white cursor is shown on the following line.

```
fish /home/tarcisio/Desktop/apresentacao-python
File Edit View Search Terminal Help
tarcisio ➤ ~/D/apresentacao-python ➤ python helloworld.py
Hello World!
tarcisio ➤ ~/D/apresentacao-python ➤
```


Python é Imperativa

```
script1.py
1  #!/bin/bash/env python
2
3  def soma(a, b):
4      return a+b
5
6  if __name__ == "__main__":
7      print(soma(2, 5))
8  |
```

Python tem suporte à OO

```
script1.py
1  #!/bin/bash/env python
2
3  class Carro():
4      consumo = 0 # km/litro
5      combustivel = 0 # quantidade no tanque
6
7      def __init__(self, consumo): # construtor
8          self.consumo = consumo
9
10     def andar(self, km):
11         litros = self.consumo*km
12         if(litros > self.combustivel):
13             print('Não há combustível suficiente')
14         else:
15             print('andou')
16
17     def getGasolina(self):
18         return self.combustivel
19
20     def addGasolina(self, litros):
21         self.combustivel +=litros
22
23 def teste():
24     # fusquinha
25     meu_fusca = Carro(12)
26     meu_fusca.addGasolina(20)
27     meu_fusca.andar(99)
28
29     # Gol bala
30     gol = Carro(15)
31     gol.addGasolina(100)
32     gol.andar(2)
33
34     # Palio
35     palio = Carro(10)
36     palio.addGasolina(30)
37     print(palio.getGasolina())
38
39 if __name__ == "__main__":
40     teste()
41
```

Python tem suporte à programação funcional



Exemplos

Alguns exemplos do uso das ferramentas funcionais do python.

- Gerar lista dos primos entre 2 e 50

```
>>> print filter(None, map(lambda y: y * reduce(lambda x, y:
x * y != 0, map(lambda x, y: y % x, range(2, int(pow(y, 0.5) + 1))),
1), range(2, 50)))
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

- Geras os 10 primeiros valores da sequência de fibonnaci

```
>>> print map(lambda x, f=lambda x, f: (x <= 1) or (f(x-1, f)
+f(x-2, f)): int(f(x, f)), range(10))
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Tipagem dinâmica

Python REPL (ptpyth

File Edit View Search Terminal Help

```
>>> inteiro = 10
>>> string = "eae men kkk"
>>> ponto_flutuante = 1.3
>>> lista = [10, "ola", 3.5]
>>> type(inteiro)
<type 'int'>
>>> type(string)
<type 'str'>
>>> type(ponto_flutuante)
<type 'float'>
>>> type(lista)
<type 'list'>
>>> 
```

Python REP


File Edit View Search Terminal Help

```
>>> inteiro = 10
>>> inteiro = "sou uma string"
>>> print(inteiro)
sou uma string
>>> 
```

Open source <3

<https://github.com/python>

<https://www.python.org/downloads/source/>



Python

Repositories related to the Python Programming language

<https://www.python.org/>

Repositories 57

People 85

Pinned repositories

cpython

The Python programming language

Python ★ 13.8k 🍴 2.9k

mypy

Optional static typing for Python 2 and 3 (PEP484)

Python ★ 3k 🍴 375

python-dotorg

Source code for python.org

Python ★ 722 🍴 253

peps

Python Enhancement Proposals

Python ★ 642 🍴 237

typeshed

Collection of library stubs for Python, with static types

Python ★ 510 🍴 328

devguide

The Python developer's guide

Python ★ 282 🍴 98

Bibliotecas + Frameworks

<https://github.com/vinta/awesome-python>

frameworks_web = ["Django", "Flask", "requests", "urllib"]

web_scrapping = ["beautiful Soup 4", "Requests", "Selenium"]

análise_de_dados = ["pandas", "matplotlib", "numpy"]

Inteligencia_artificial = ["TensorFlow", "Pytorch", "scikit-learn"]

GUI = ["Tkinter", "kivy", "PyQt"]

games = ["Pygame"]

""" e muito mais: audio, autenticação, visão computacional
parallelism, criptografia, banco de dados, email, logging

Processamento de linguagem natural, network, Processes, RESTful
API, software tests, web crawling """

~# pip install pyinstaller

Gerenciador de pacotes

```
fish /home/tarcisio/RSB-Framework

File Edit View Search Terminal Help

tarcisio ~ /RSB-Framework master ls
LICENSE README.md requeriments.txt server/ victim/
tarcisio ~ /RSB-Framework master cat requeriments.txt
pynput
pyscreenshot==0.4.2
pyinstaller
ImageGrab
pillow
tarcisio ~ /RSB-Framework master sudo pip install -r requeriments.txt
[sudo] password for tarcisio:
Requirement already satisfied: pynput in /usr/local/lib/python2.7/dist-packages/
pynput-1.3.5-py2.7.egg (from -r requeriments.txt (line 1))
Requirement already satisfied: pyscreenshot==0.4.2 in /usr/local/lib/python2.7/d
ist-packages (from -r requeriments.txt (line 2))
Requirement already satisfied: pyinstaller in /usr/local/lib/python2.7/dist-pack
ages (from -r requeriments.txt (line 3))
Collecting ImageGrab (from -r requeriments.txt (line 4))
  Could not find a version that satisfies the requirement ImageGrab (from -r req
ueriments.txt (line 4)) (from versions: )
No matching distribution found for ImageGrab (from -r requeriments.txt (line 4))
x tarcisio ~ /RSB-Framework master
```



Python vs C

Suporte a Orientação a Objetos vs apenas Imperativa e estruturada

Interpretada x Compilada

Python é lento !

tratamento de exceções

Python vs C

```
-bash-3.00$ ./a.out
Name          Number      Rating
-----
Smiley        662          *****
Segmentation Fault (core dumped)
-bash-3.00$
```



```
Python REPL (ptpython)
File Edit View Search Terminal Help
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
integer division or modulo by zero

>>> pow(1, "eae")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'int' and 'str'
unsupported operand type(s) for ** or pow(): 'int' and 'str'

>>>
```



Python vs Java

Interpretado vs Compilado/Interpretado

Imperativo, OO vs OO (only)

Tipagem dinâmica vs estática

Os dois são lentos

<https://pythonconquerstheuniverse.wordpress.com/2009/10/03/python-java-a-side-by-side-comparison/>

Inicialização de variáveis

```
>>> inteiro = 10

>>> string = "bom dia"

>>> ponto_flutuante = 3.5

>>> lista = [10, 20]

>>> dicionario = {"nome": "produto X", "preco": 40}

>>> tupla = (40, 50)

>>> boolean = True

>>> █
```

Condicionais

```
1 inteiro = 10
2 if(inteiro == 10):
3     print("Acertou")
4 else:
5     print("Errou")
6
7
8 string = "Ola como vai ?"
9
10 if(string == "Ola como vai?"):
11     print("como vai")
12
13 elif(string == "estou bem"):
14     print("estou bem")
15
16 else:
17     print("qualquer outra coisa")
```

Operadores Lógicos

```
1  a = 10
2  b = 20
3
4  if (a == 10 and b == 20): # e
5      pass
6
7  elif (a == 10 or b == 20): # ou
8      pass
9
10 if(a > b): # a maior que b
11     pass
12
13 if (a < b): # b maior que a
14     pass
15
16 if(a >= b): # a maior ou igual a b
17     pass
18
19 if (a <= b): # a menor igual a b
20     pass
21
22 c = a + b
23 c = a - b
24 c = a * b
25 c = a / b
26 c = a ** b
27 c = a % b
28 a+=1 # a = a + 1
29 a*=2 # a = a * 2
30
```

Estruturas de repetição

```
1  while(True):
2      ## repete this forever
3      print("hello world")
4
5  for variavel in range(10):
6      print(variavel)
7      #imprime de 0 ate 9
8
9  palavra = "ola"
10 for letra in palavra:
11     print(letra)
12     #imprime cada letra da palavra
13
14 lista = [1, "eae", 3]
15 for elemento in lista:
16     print(elemento)
17     # imprime cada elemento da lista
```

Funções

```
1
2  def soma(a, b):
3      |   return a+b
4
5  def subtracao(a, b):
6      |   return a-b
7
8  def multiplicacao(a, b):
9      |   return a*b
10
11 def divisao(a, b):
12     |   return a/b
13
```

Entrada de texto do teclado

```
1  entrada = input() # lê do teclado
2
3  entrada2 = input("digite seu nome: ") # imprime o texto e lê do teclado
4
5  print(entrada, entrada2)
6
7  |
```


Conversão de tipos (cast)

```
1 inteiro = 10
2
3 inteiro = str(inteiro) # converte para uma string
4
5 string = "10"
6
7 string = int(string) # converte a string para inteiro
8
9
```

Imports

```
1  import os # importa um pacote inteiro
2
3  from os import path # importa apenas uma
4  |                   # função/método/classe/variavel do pacote
5
6  from os import * # importa tudo e deixa sem o nome do pacote na chamada
7
8  import BeautifulSoup4 as bs # nomeia o pacote para bs
9
10 import pasta/script # importa de outro arquivo do seu projeto
```

Strings

Concatenação de strings

```
nome = "tarcisio" + " " + "marinho"  
print("bem vindo " + nome) # bem vindo tarcisio marinho
```

String é uma sequência de caracteres

```
string = "eu sou uma lista de caracteres"  
  
for caracter in string:  
    print(caracter)
```

```
6     print nome[0]
```

```
7
```



Strings

Substring

```
if "tar" in nome :  
    print "achou a substring"
```

Operações com strings

lower

```
1 nome = "tARCisio MarINH0"  
2  
3 nome = nome.lower()  
4 print(nome)
```

upper

```
nome = "tARCisio MarINH0"  
  
nome = nome.upper()  
print(nome)
```

split

```
1 nome = "tarcisio marinho"  
2  
3 nome = nome.split(' ') # separa onde tiver espaco  
4 # vira uma lista  
5 print(nome)
```

```
tarcisio ~ /D/a/codigos python strings.py  
['tarcisio', 'marinho']  
tarcisio ~ /D/a/codigos
```

Operações com strings

replace

```
1 nome = "tarcisio marinho"
2
3 nome = nome.replace("a","*")
4
5 print(nome)
```

tarcisio ~/D/a/codigos python strings.py
t*rcisio m*rinho
tarcisio ~/D/a/codigos

join

```
1 lista_nomes = ["tarcisio", "lucas", "euler", "victor", "hildemir"]
2
3 nova_string = ' '.join(lista_nomes)
4
5 print(nova_string)
```

tarcisio ~/D/a/codigos python strings.py
tarcisio lucas euler victor hildemir
tarcisio ~/D/a/codigos

Operações com strings

title

```
1 nome = "meU NOmE eH TARCisio marINHO"
2 nome = nome.title()
3 print(nome)

tarcisio ~/D/a/codigos python strings.py
Meu Nome Eh Tarcisio Marinho
tarcisio ~/D/a/codigos
```

substrings

```
>>> string = "tarcisio marinho"

>>> string[9:18]
'marinho'

>>>
```

Operações com strings

reverse

```
>>> string = "euler"
>>> string[::-1]
'rellue'
>>> 
```




Outros métodos

isalnum - retorna True se todos os caracteres forem alfanumericos a-Z, 0-9

isalpha - for do alfabeto

isdigit - de 0-9

islower - se todos forem minusculos

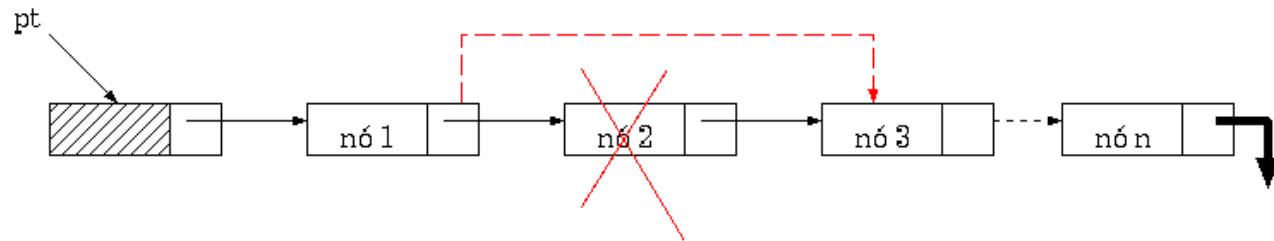
isupper - se todos forem maiúsculos

isspace - se for espaço: " "

Listas

- # Listas são estruturas lineares
- # Podem ser utilizadas como filas ou pilhas
- # esqueça listas encadeadas (C)

```
typedef struct node{  
    struct node *prox;  
    char * info[3];  
}List;
```



```
void append(List **l, char *file_path, char *key, char *iv);  
void destroy(List **l);  
void print(List *l);  
int length(List *l);  
#endif
```

Listas

Declaração

```
>>> minha_lista = ["campo1", "campo2", 3, 4.5]
>>> █
```

append

```
>>> minha_lista = ["campo1", "campo2", 3, 4.5]
>>> minha_lista.append("tarcisio")
>>> minha_lista
['campo1', 'campo2', 3, 4.5, 'tarcisio']
>>> █
```

Listas

remove

```
>>> minha_lista.remove('tarcisio')
>>> minha_lista
['campo1', 'campo2', 3, 4.5]
>>> █
```

insert

```
>>> minha_lista
['campo1', 'campo2', 3, 4.5]
>>> minha_lista.insert(2, "tarcisio")
>>> minha_lista
['campo1', 'campo2', 'tarcisio', 3, 4.5]
>>> █
```

Listas

index

```
>>> minha_lista
['campo1', 'campo2', 'tarcisio', 3, 4.5]

>>> minha_lista.index("tarcisio")
2

>>> minha_lista.index(4.5)
4

>>> 
```

count

```
>>> lista = ["30", "30", "20", 30, "40"]

>>> lista.count(30)
1

>>> lista.count("30")
2

>>> 
```

Listas

sort

```
>>> lista = [10, 5, 17, 31, 4, 2]
>>> lista.sort()
>>> lista
[2, 4, 5, 10, 17, 31]
>>> 
```

reverse

```
>>> lista
[2, 4, 5, 10, 17, 31]
>>> lista.reverse()
>>> lista
[31, 17, 10, 5, 4, 2]
>>> 
```

Listas como pilhas

append e pop são operações básicas da pilha

```
>>> lista = [10, 20, 30, 40, 50, "60"]
>>> lista.pop()
'60'
>>> lista
[10, 20, 30, 40, 50]
>>> retorno = lista.pop()
>>> retorno
50
>>> lista
[10, 20, 30, 40]
>>> 
```

Lista não pythonica

```
>>> lista = []  
  
>>> for elemento in range(10):  
...     lista.append(elemento**2)  
  
>>> lista  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
  
>>> 
```

```
>>> combs = []  
>>> for x in [1,2,3]:  
...     for y in [3,1,4]:  
...         if x != y:  
...             combs.append((x, y))  
...  
>>> combs  
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

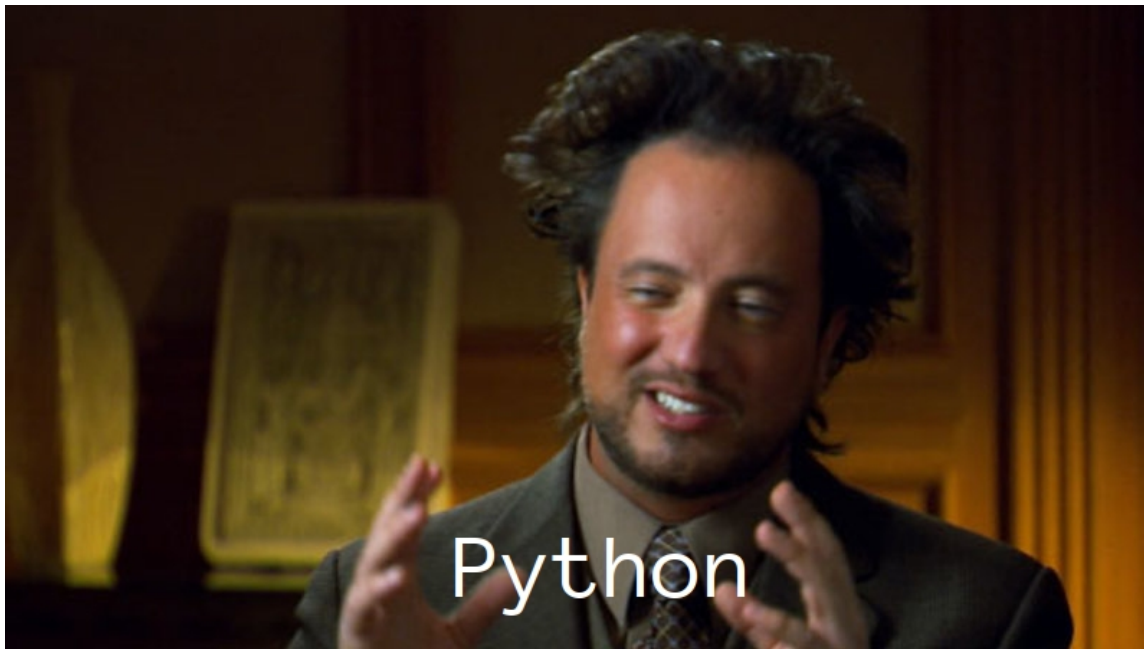

Lista pythonica

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]  
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

```
>>> lista = [2, 3, 4, 5]  
  
>>> [x*2 for x in lista]  
[4, 6, 8, 10]  
  
>>> 
```

Lista pythonica

```
>>> sites = ["www.google.com", "www.facebook.com", "www.youtube.com"]
>>> sites = [site.replace("www.", "") for site in sites]
>>> sites
['google.com', 'facebook.com', 'youtube.com']
>>> █
```



Tuplas

- # Tuplas são parecidas com listas
- # Tuplas são imutáveis (tamanho fixo)
- # Inalteráveis

```
208 def conecta(IP, PORT):
209     try:
210         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
211         s.connect((IP, PORT))
212         s.send('[+] Conectado :))')
213         return s
214     except socket.error as erro:
215         return None
216
```

Tuplas

Tipos de inicialização

```
>>> tupla = ("oi", 3, 4.5)
>>> tupla2 = 3, "tudo bem", 54
>>> 
```

Tuplas

Retorno de funções

```
>>> def retornar_tupla():  
...     return 4, 3, 2, 1  
  
>>> tupla = retornar_tupla()  
  
>>> tupla  
(4, 3, 2, 1)  
  
>>> type(tupla)  
<type 'tuple'>  
  
>>> 
```

Dicionarios

- # É uma estrutura de {chave : valor}
- # Similar ao JSON (js)

```
>>> agenda = {"tarcisio":"081995035594", "euller": "12312312312"}  
>>> agenda["diego"] = "1239188312123" # adicionando novo contato  
>>> 
```

Diccionarios

has_key

```
>>> agenda
{'tarcisio': '081995035594', 'diego': '1239188312123', 'euler': '12312312312'}

>>> agenda.has_key("tarcisio")
True

>>> agenda.has_key("hildemir")
False

>>> █
```

Dicionarios

get

```
>>> agenda
{'tarcisio': '081995035594', 'diego': '1239188312123', 'euller': '12312312312'}

>>> agenda.get("tarcisio")
'081995035594'

>>> agenda["tarcisio"]
'081995035594'

>>> 
```


Dicionarios

pop

```
>>> agenda
{'tarcisio': '081995035594', 'diego': '1239188312123', 'euller': '12312312312'}

>>> agenda.pop("euller")
'12312312312'

>>> agenda
{'tarcisio': '081995035594', 'diego': '1239188312123'}

>>> 
```

Dicionarios de listas

```
1 agenda = {"tarcisio": ["081995035594", "github.com/tarcisio-marinho", "20"]
2 |         , "euller":["12312312312312", "gihub.com/euller", "18"]}
3 print(agenda.get("euller"))
```

```
tarcisio ~ /D/a/codigos python dicionario.py
['12312312312312', 'gihub.com/euller', '18']
tarcisio ~ /D/a/codigos
```

Arquivo (I/O)

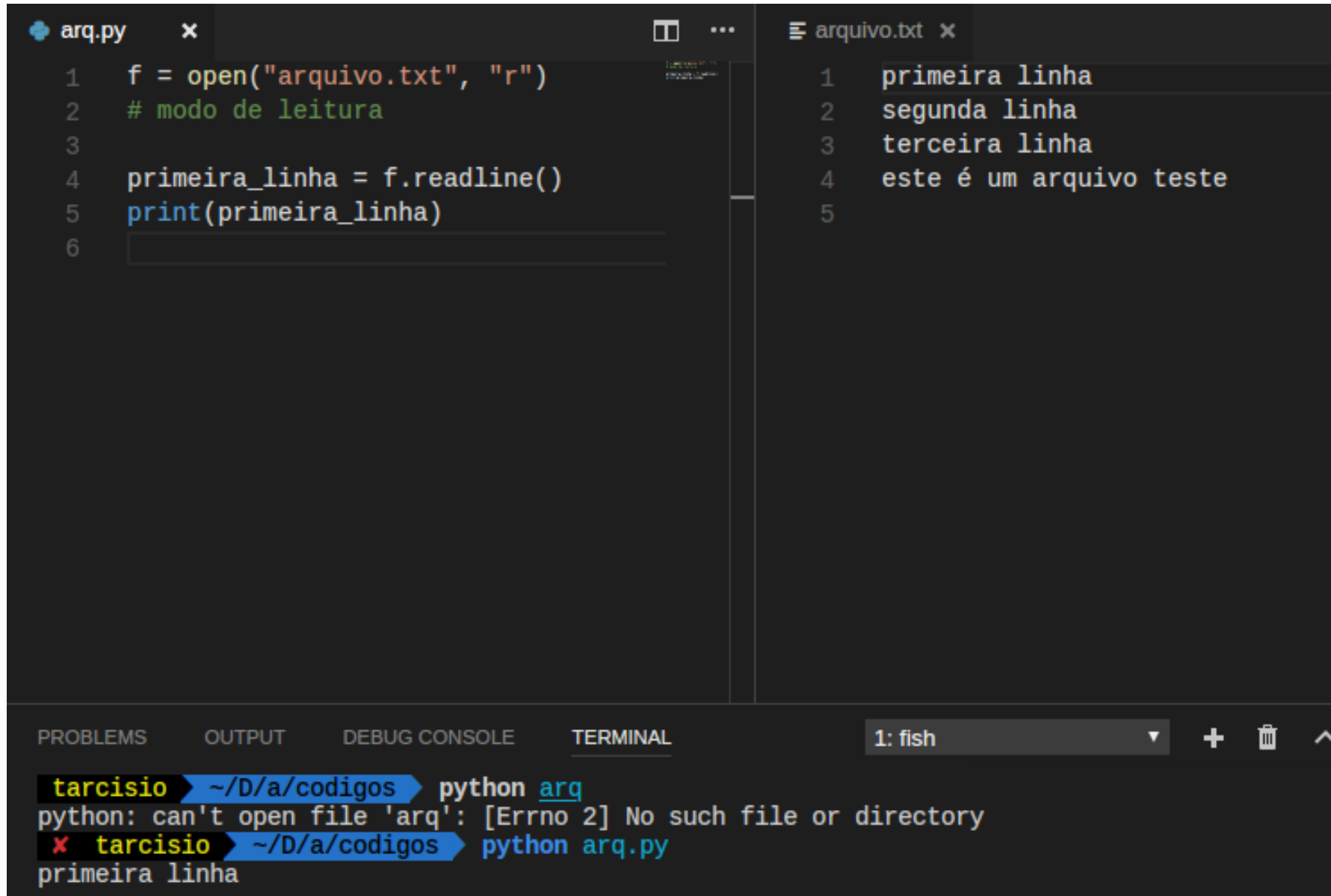
**# Você não precisa saber a quantidade de bytes
Que vai ler ou escrever em arquivo**

esqueça fread, fwrite, fseek (C)

```
f = fopen(new_file, "w");

while(l != NULL){
    line = malloc((sizeof(char)*strlen(l->info[0]) + strlen(l->info[1]) + strlen(l->info[2]) + 11));
    strcpy(line, l->info[0]);
    strcat(line, ":");
    strcat(line, l->info[1]);
    strcat(line, ":");
    strcat(line, l->info[2]);
    strcat(line, "\n");
    fwrite(line, strlen(line), 1, f);
    memset(line, 0, strlen(line));
    l = l->prox;
}
```

Arquivo (I/O)



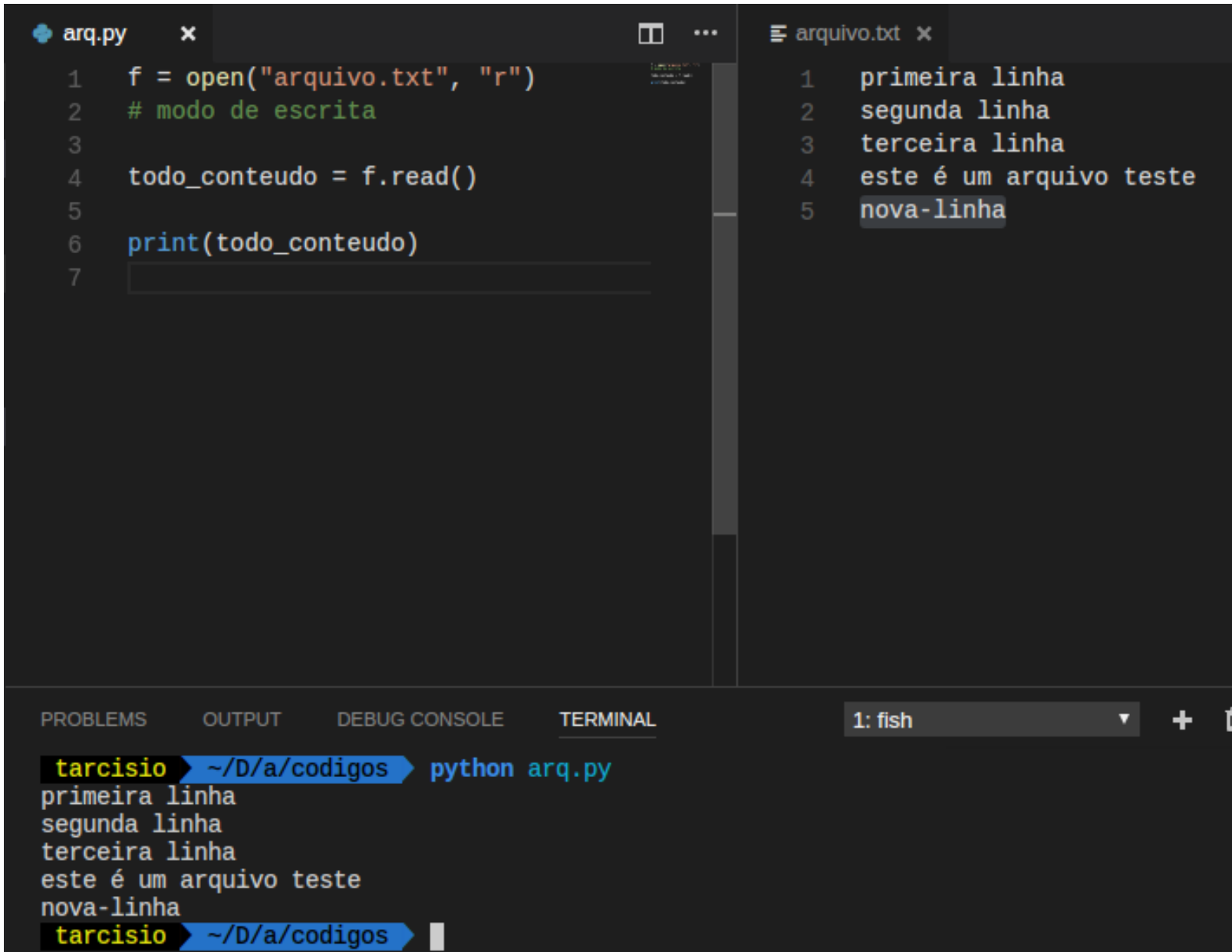
The screenshot shows a code editor with two tabs: `arq.py` and `arquivo.txt`. The `arq.py` tab contains a Python script that opens `arquivo.txt` in read mode and prints the first line. The `arquivo.txt` tab contains five lines of text. At the bottom, a terminal window shows the execution of the script, which results in the first line of the file being printed.

```
arq.py x
1 f = open("arquivo.txt", "r")
2 # modo de leitura
3
4 primeira_linha = f.readline()
5 print(primeira_linha)
6

arquivo.txt x
1 primeira linha
2 segunda linha
3 terceira linha
4 este é um arquivo teste
5

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: fish
tarcisio ~/D/a/codigos python arq
python: can't open file 'arq': [Errno 2] No such file or directory
x tarcisio ~/D/a/codigos python arq.py
primeira linha
```

Arquivo (I/O)



The image shows a code editor with two tabs: `arq.py` and `arquivo.txt`. The `arq.py` tab contains a Python script that opens `arquivo.txt` in read mode, reads its contents, and prints them. The `arquivo.txt` tab contains five lines of text. Below the editor, the terminal shows the command `python arq.py` being executed, and the output matches the contents of `arquivo.txt`.

```
arq.py x
1 f = open("arquivo.txt", "r")
2 # modo de escrita
3
4 todo_conteudo = f.read()
5
6 print(todo_conteudo)
7

arquivo.txt x
1 primeira linha
2 segunda linha
3 terceira linha
4 este é um arquivo teste
5 nova-linha

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: fish
tarcisio ~ /D/a/codigos python arq.py
primeira linha
segunda linha
terceira linha
este é um arquivo teste
nova-linha
tarcisio ~ /D/a/codigos
```

Arquivo (I/O)

```
1 f = open("arquivo.txt", "a")
2 # modo de escrita
3
4 f.write("nova-linha")
5 |
```

```
1 primeira linha
2 segunda linha
3 terceira linha
4 este é um arquivo teste
5
```

```
1 primeira linha
2 segunda linha
3 terceira linha
4 este é um arquivo teste
5 nova-linha
```



Tratamento de exceções

- # O que fazer quando as coisas dão errado?**
- # possibilidade de tratar com vários erros poss**
- # sinais do SO são exceções**

Tratamento de exceções

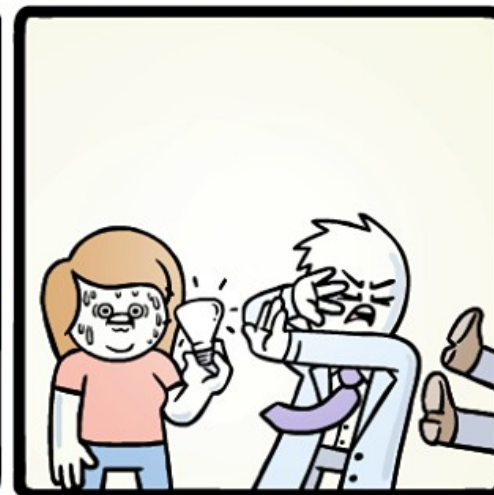
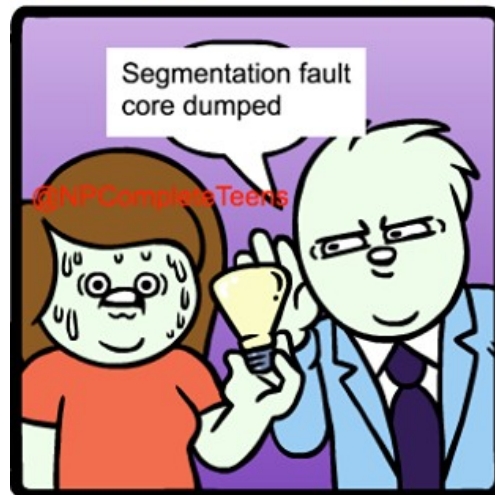
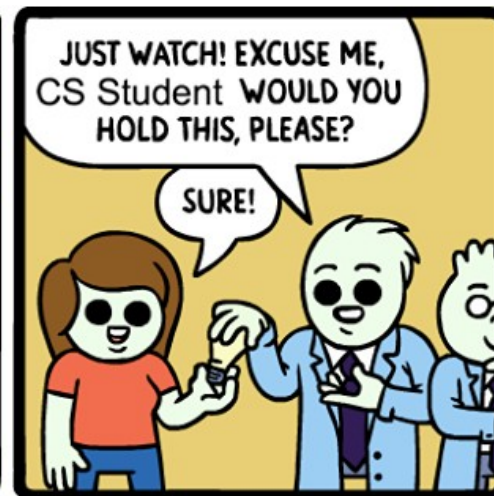
```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
integer division or modulo by zero

>>> 13 ** "oi"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'int' and 'str'
unsupported operand type(s) for ** or pow(): 'int' and 'str'

>>>
>>> texto = raw_input()
KeyboardInterrupt

>>> f = open("aksmdkasmda")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'aksmdkasmda'
[Errno 2] No such file or directory: 'aksmdkasmda'

>>> █
```

Tratamento de exceções

try, catch

```
1  try:
2      1/0
3  except:
4      print("Divisao por zero nao pode")
5
```

```
try:
    1/0
except ZeroDivisionError:
    print("Divisao por zero nao pode")
except:
    print("algum outro erro")
```



Orientação a Objetos (OO)

Python tem suporte a orientação a objetos

Python não tem interfaces, porém tem herança

Múltipla e classes abstratas

Python não tem tipos primitivos, tudo é objeto

Não existe a obrigação de criar classes



Classes e Objetos

- # Classe é o molde de como um objeto será e i**
Se comportar
- # Uma classe define comportamento através d**
Métodos e os seus estados são atributos.
- # Objetos são instanciados(alocados) em mem**
- # Método especial `__init__` inicializa o objeto**
(Construtor/ Inicializador)

Classes e seus métodos

```
1 class animal():
2     def __init__(self):
3         self.nome = ""
4
5     def setNome(self, nome):
6         self.nome = nome
7
8 ovelha = animal()
9 print "nome", ovelha.nome
10 ovelha.setNome("ovelha")
11 print "nome", ovelha.nome
```

```
tarcisio ~/D/a/c/00 ptpython classes.py
```

```
nome
```

```
nome ovelha
```

```
tarcisio ~/D/a/c/00
```

Classes e objetos

Todos os métodos devem declarar o self como primeiro parâmetro

Todos acessos a atributos e métodos devem referenciar o self

Ao instanciar um objeto, não precisa utilizar (Java, C++)

```
ovelha = animal()  
print "nome", ovelha.nome  
ovelha.setNome("ovelha")  
print "nome", ovelha.nome
```

Atributos de classe x instân

```
1 class animal():
2     nome = "apolo"
3
4 cachorro = animal()
5 print "nome do cachorro", cachorro.nome
6
7 cachorro.nome = "jubileu"
8
9 print "nome do cachorro", cachorro.nome
10
11 gato = animal()
12
13 print "nome do gato", gato.nome
```

✗ **tarcisio** > ~/D/a/c/00 > ptpython classes.py

nome do cachorro apolo

nome do cachorro jubileu

nome do gato apolo

tarcisio > ~/D/a/c/00 > □

Encapsulamento

```
1 class Animal():
2
3     def __init__(self, nome, idade):
4         self.nome = nome
5         self.__idade = idade
6
7     def get_idade(self):
8         return self.__idade
9
10    def set_idade(self, idade):
11        self.__idade = idade
12
13    def set_nome(self, nome):
14        self.nome = nome
15
16    a = Animal("apolo", 4)
17    print a.nome
18    print a.get_idade()
19    a.set_nome("thor")
20    a.set_idade(5)
21
22    print a.get_idade()
```

```
tarcisio ~ /D/a/c/00 python classes.py
```

```
apolo
```

```
4
```

```
5
```

```
tarcisio ~ /D/a/c/00
```


Atributo x Propriedade

```
1 class Carro():
2     def __init__(self):
3         self.velocidade = 20
4         self.tempo = 2
5
6     def velocidade_media(self):
7         return self.velocidade / self.tempo
8
```



Herança

- # Uma classe herda seus atributos e métodos**
De uma classe pai
- # A classe filha, pode acrescentar novos métodos**
E atributos
- # A classe filha pode sobreescrever métodos da**
Classe pai (Polimorfismo)

Herança

```
1 class Animal():
2     def __init__(self, idade, nome):
3         self.idade = idade
4         self.nome = nome
5
6     def falar(self):
7         print('eu estou falando')
8
9     def comer(self):
10        print("estou comendo")
11
12 class Cachorro(Animal):
13     pass
14
15 beagle = Cachorro(10, "apolo")
16
17 beagle.falar()
18 beagle.comer()
tarcisio ~/D/a/c/00 python3 heranca.py
eu estou falando
estou comendo
tarcisio ~/D/a/c/00
```

Sobrescrita de métodos

```
1 class Animal():
2     def __init__(self, idade, nome):
3         self.idade = idade
4         self.nome = nome
5
6     def falar(self):
7         print('eu estou falando')
8
9     def comer(self):
10        print("estou comendo")
11
12 class Cachorro(Animal):
13     def falar(self):
14         print("eu sou um cachorro e tenho minha propria fala, au au")
15
16
17 beagle = Cachorro(10, "apolo")
18
19 beagle.falar()
tarcisio ~ /D/a/c/00 python classes.py
eu sou um cachorro e tenho minha propria fala, au au
```

Super ()

```
1 class Animal():
2     def __init__(self, idade, nome):
3         self.idade = idade
4         self.nome = nome
5
6     def falar(self):
7         print('eu estou falando')
8
9     def comer(self):
10        print("estou comendo")
11
12 class Cachorro(Animal):
13     def falar(self):
14         super().falar()
15
16 beagle = Cachorro(10, "apolo")
17
18 beagle.falar()
```

```
tarcisio ~ /D/a/c/00 python3 heranca.py
```

```
eu estou falando
```

```
tarcisio ~ /D/a/c/00 □
```

Polimorfismo

**# Classe filha
Recebendo mais
Parametros no
Construtor**

```
1 class Animal():
2     def __init__(self, idade, nome):
3         self.idade = idade
4         self.nome = nome
5
6     def falar(self):
7         print('eu estou falando')
8
9     def comer(self):
10        print("estou comendo")
11
12 class Cachorro(Animal):
13     def __init__(self, idade, nome, raca):
14         super().__init__(idade, nome)
15         self.raca = raca
16         # adicionando mais parametros no construtor
17
18     def falar(self):
19         print("au au")
20
21 beagle = Cachorro(10, "apolo", "beagle")
22
23 beagle.falar()
24
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

1: file

tarcisio ~/D/a/c/00 python3 classes.py

au au

tarcisio ~/D/a/c/00

Fim!





Valeu galera !

<https://docs.python.org/3/tutorial/datastructures>

<https://docs.python.org/3/whatsnew/3.6.html>



Valeu galera !

Contato:

Email : tarcisio_marinho09@hotmail.com

fb: <https://www.facebook.com/OneDayUmay>

github: <https://github.com/tarcisio-marinho>