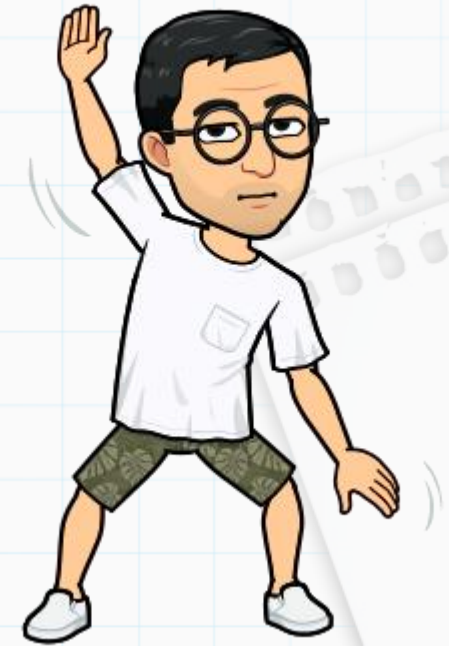
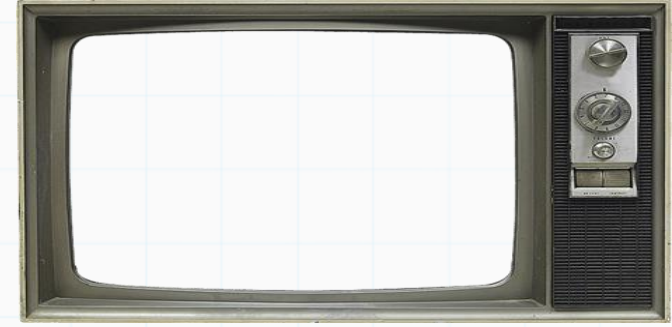


# Programação De Computadores

Professor : Yuri Frota

[www.ic.uff.br/~yuri/prog.html](http://www.ic.uff.br/~yuri/prog.html)

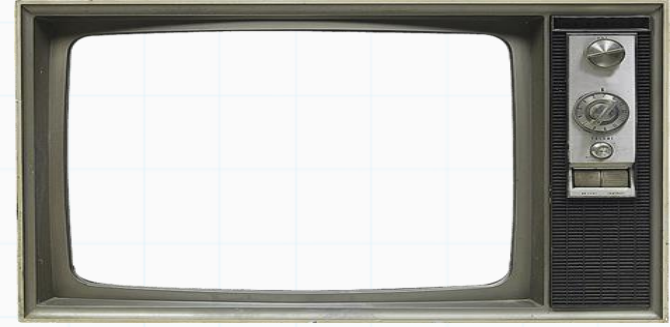
yuri@ic.uff.br



# Organização

Para começar em Python precisamos saber:

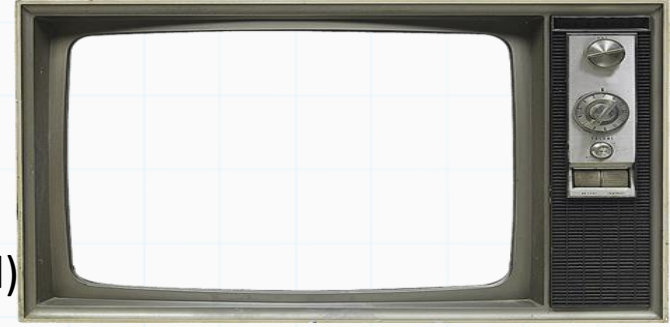
- 1) Como um programa está organizado ?
- 2) Como atribuir valores a variáveis ?
- 3) Quais os tipos de dados disponíveis ?
- 4) Como dar nomes as variáveis ?
- 5) Como fazer entrada e saída (básica) de dados ?



# 1) Como um programa está organizado

Instalando o IDE+Compilador

- Usaremos na aula o Thonny (leve e educativo) e o Pycharm (pesado e profissional)
- <https://www.jetbrains.com/pycharm/download/#section=windows>
- <https://thonny.org/>

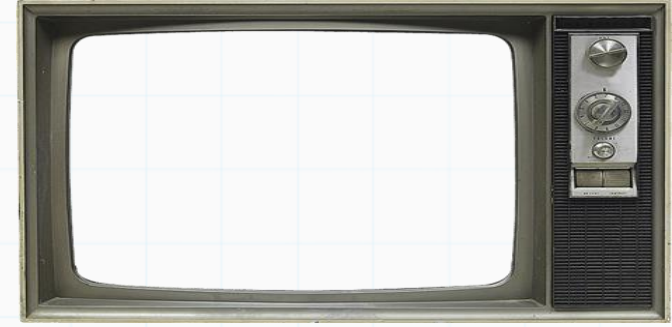
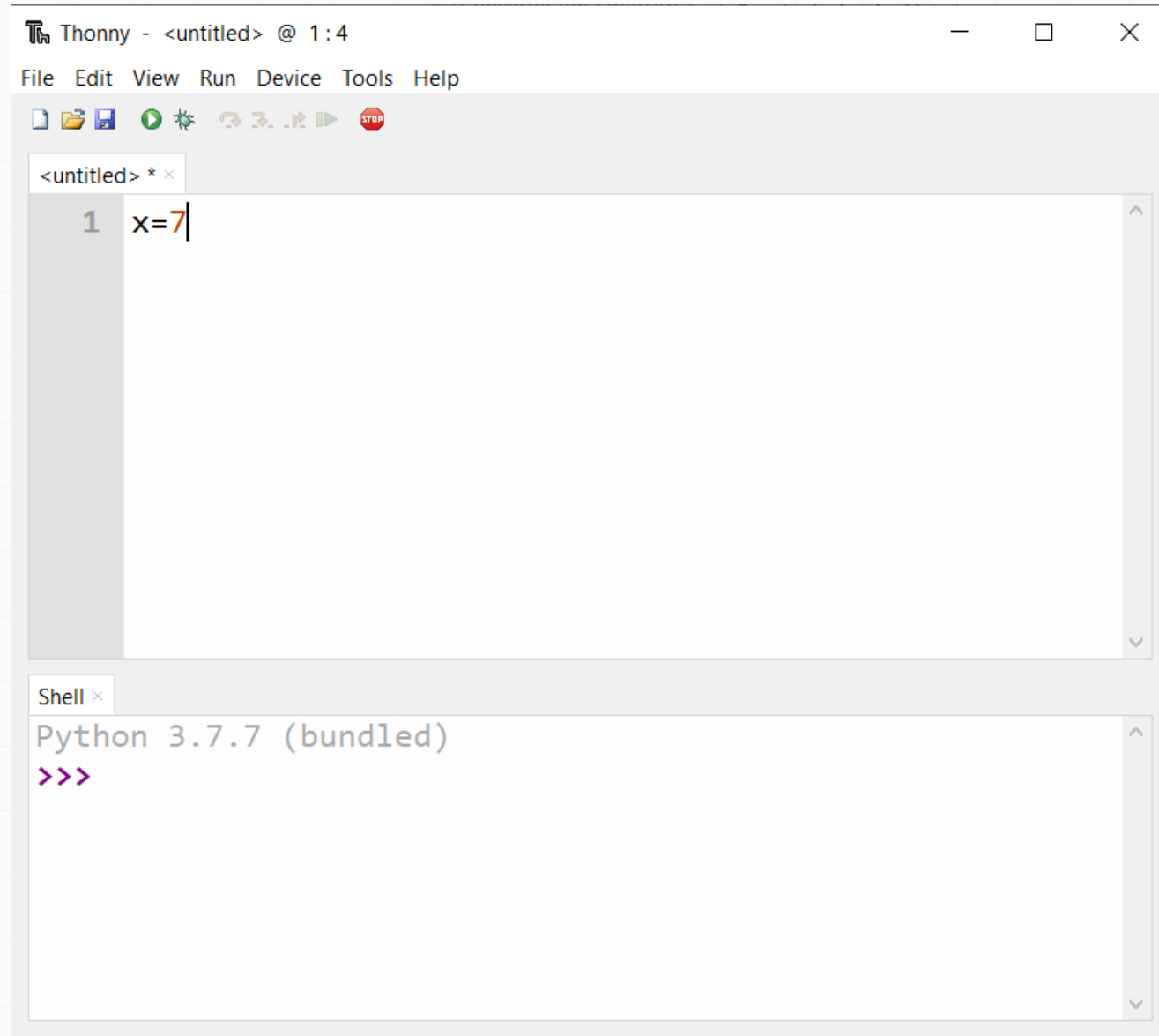


**Thonny**  
Python IDE for beginners



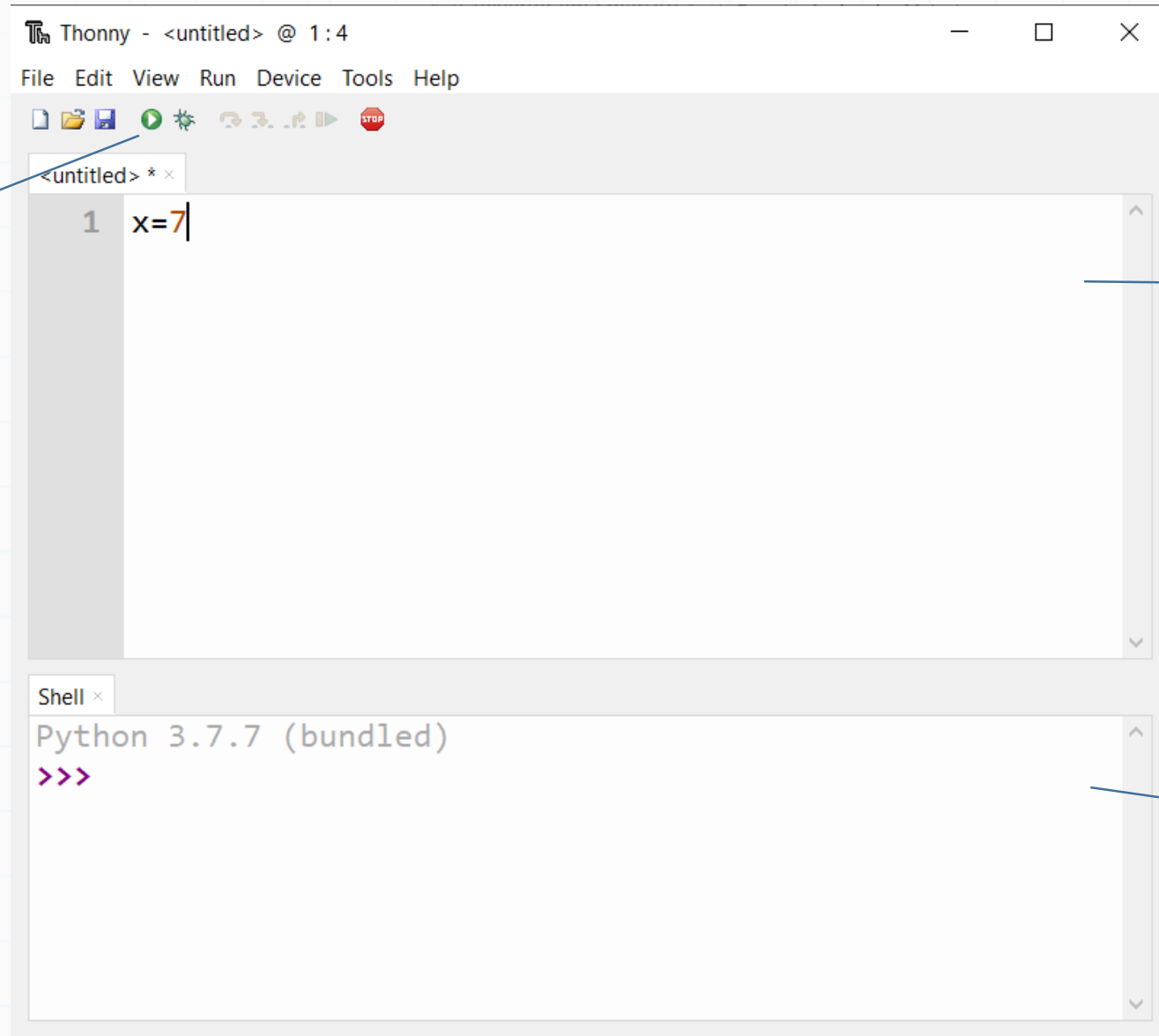
# 1) Como um programa está organizado

Usando o Thonny:



# 1) Como um programa está organizado

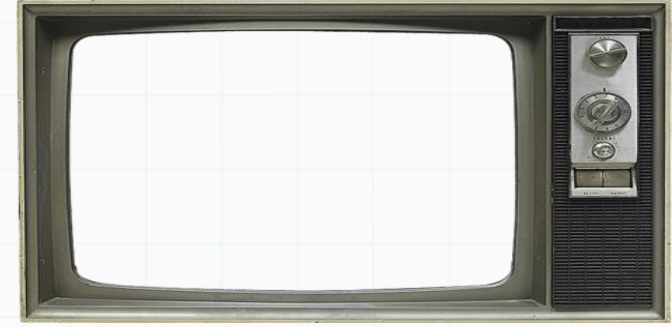
Usando o Thonny:



para executar o código

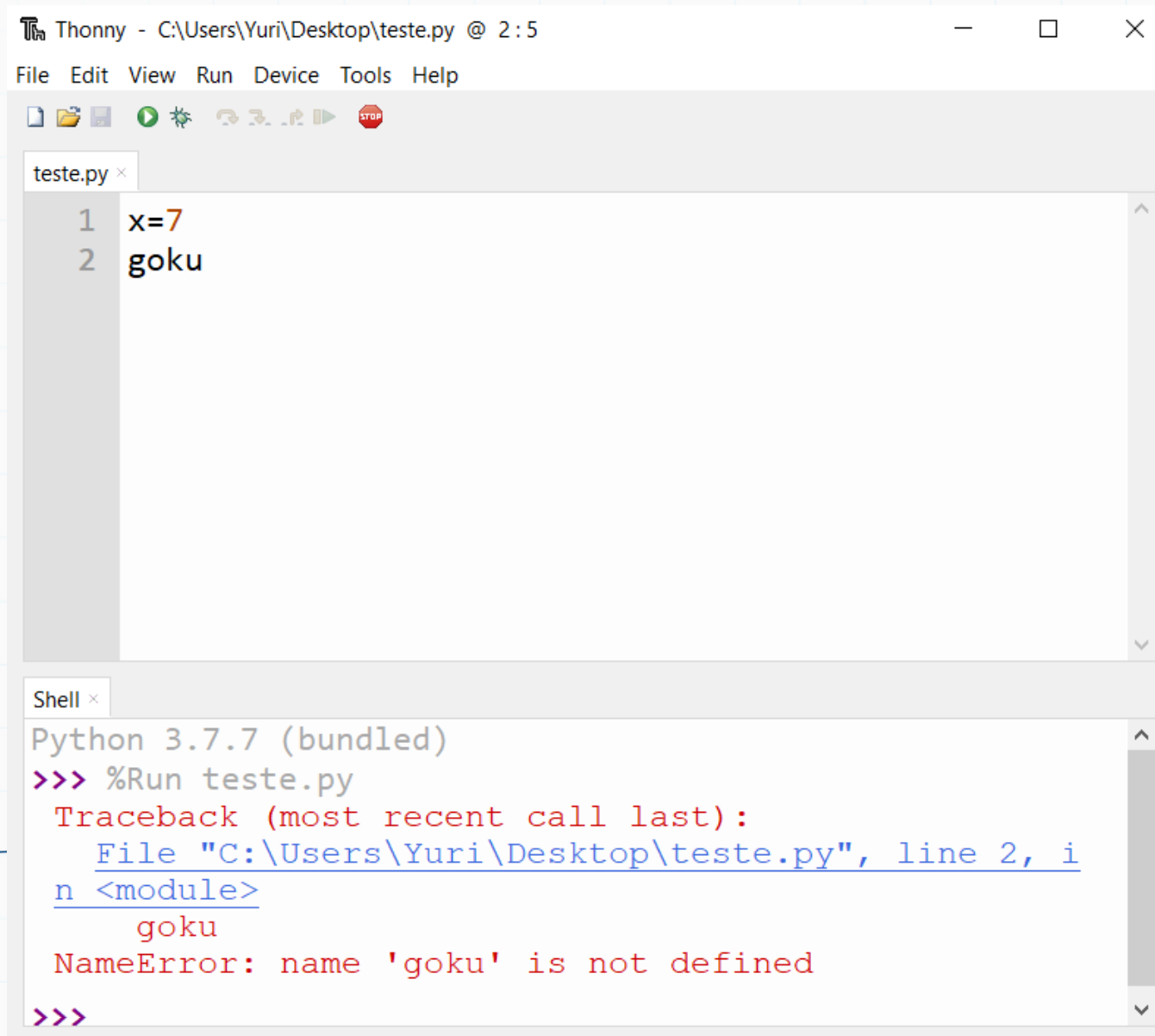
onde o código é escrito

onde a saída é mostrada



# 1) Como um programa está organizado

Usando o Thonny:



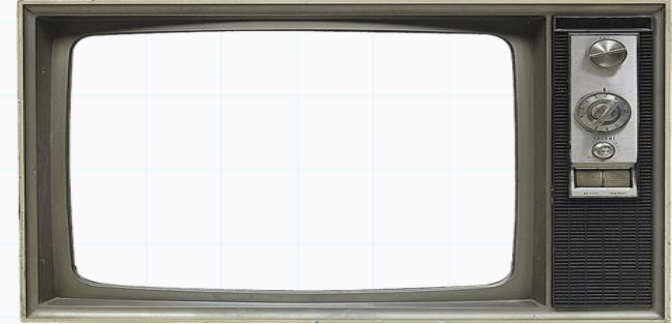
The screenshot shows the Thonny Python IDE interface. The title bar reads "Thonny - C:\Users\Yuri\Desktop\teste.py @ 2:5". The menu bar includes "File", "Edit", "View", "Run", "Device", "Tools", and "Help". The toolbar contains icons for file operations and execution. The editor window, titled "teste.py", contains the following code:

```
1 x=7
2 goku
```

The Shell window at the bottom shows the execution of the script:

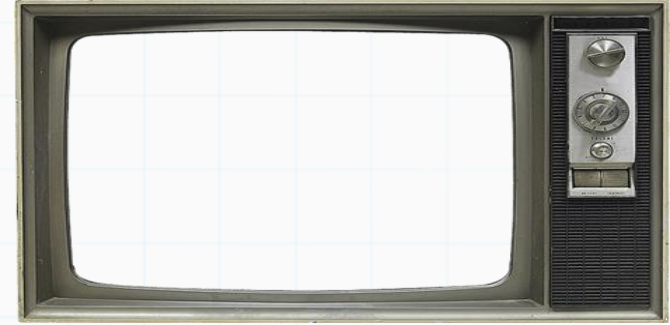
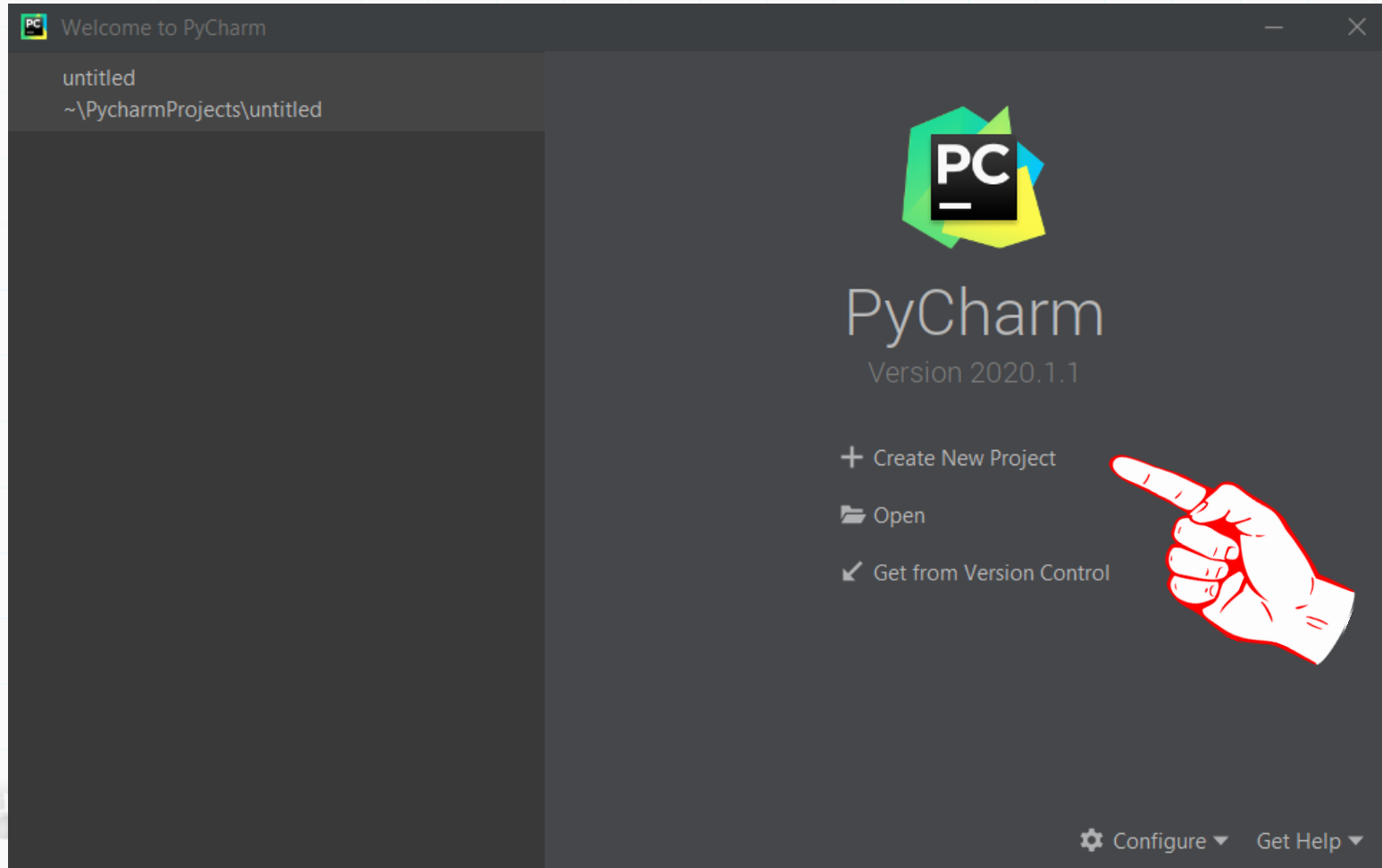
```
Python 3.7.7 (bundled)
>>> %Run teste.py
Traceback (most recent call last):
  File "C:\Users\Yuri\Desktop\teste.py", line 2, in <module>
    goku
NameError: name 'goku' is not defined
>>>
```

A blue callout box with the text "saída de erro" (error output) points to the error message in the Shell window.



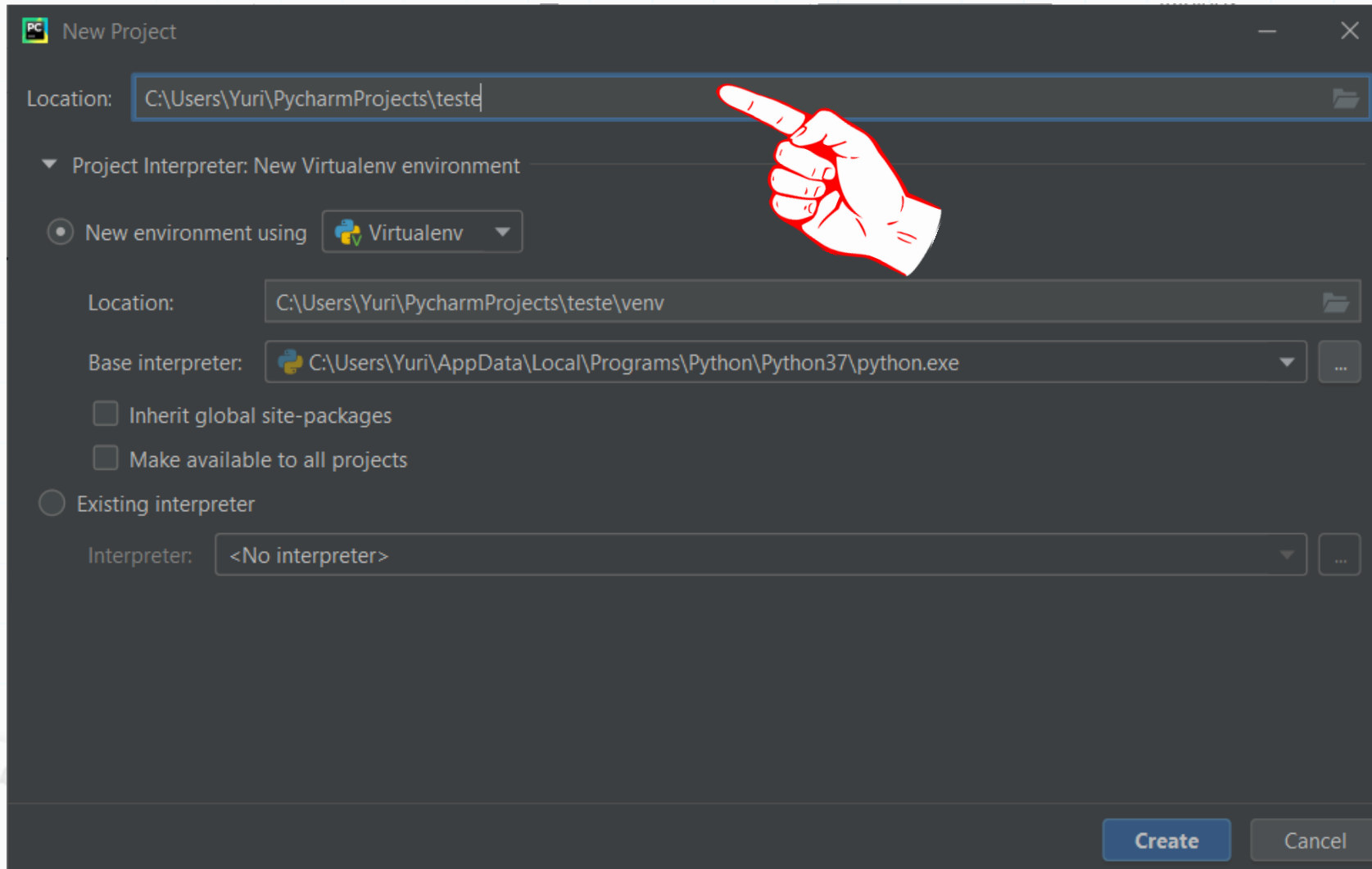
# 1) Como um programa está organizado

Usando o Pycharm: Criando um projeto novo



# 1) Como um programa está organizado

Usando o Pycharm: Nome do projeto

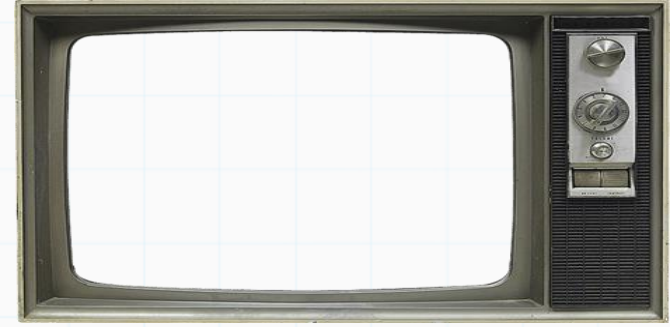


escolha do  
ambiente/compilador,  
deixa como está

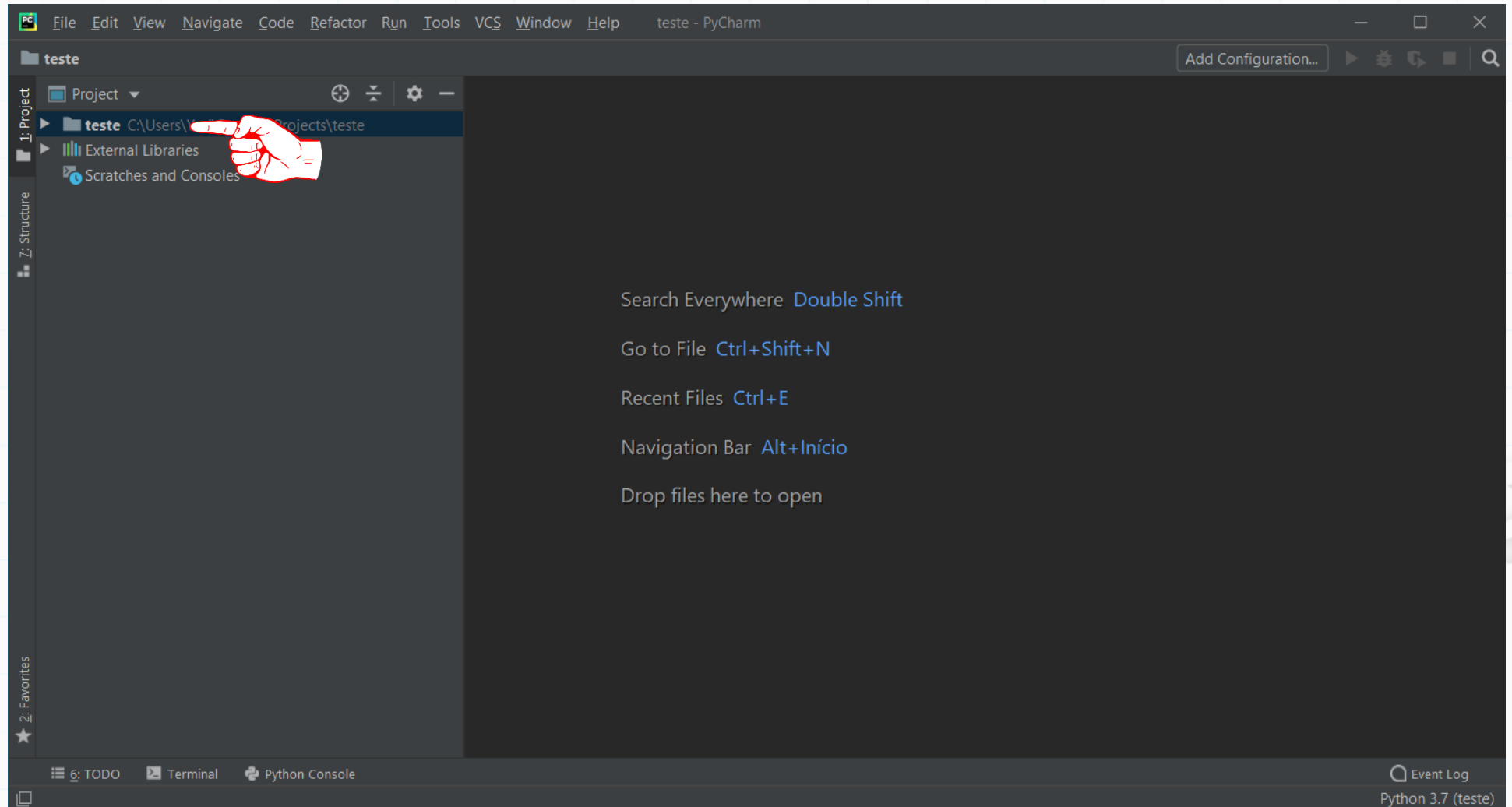


# 1) Como um programa está organizado

Usando o Pycharm: Criando arquivo python



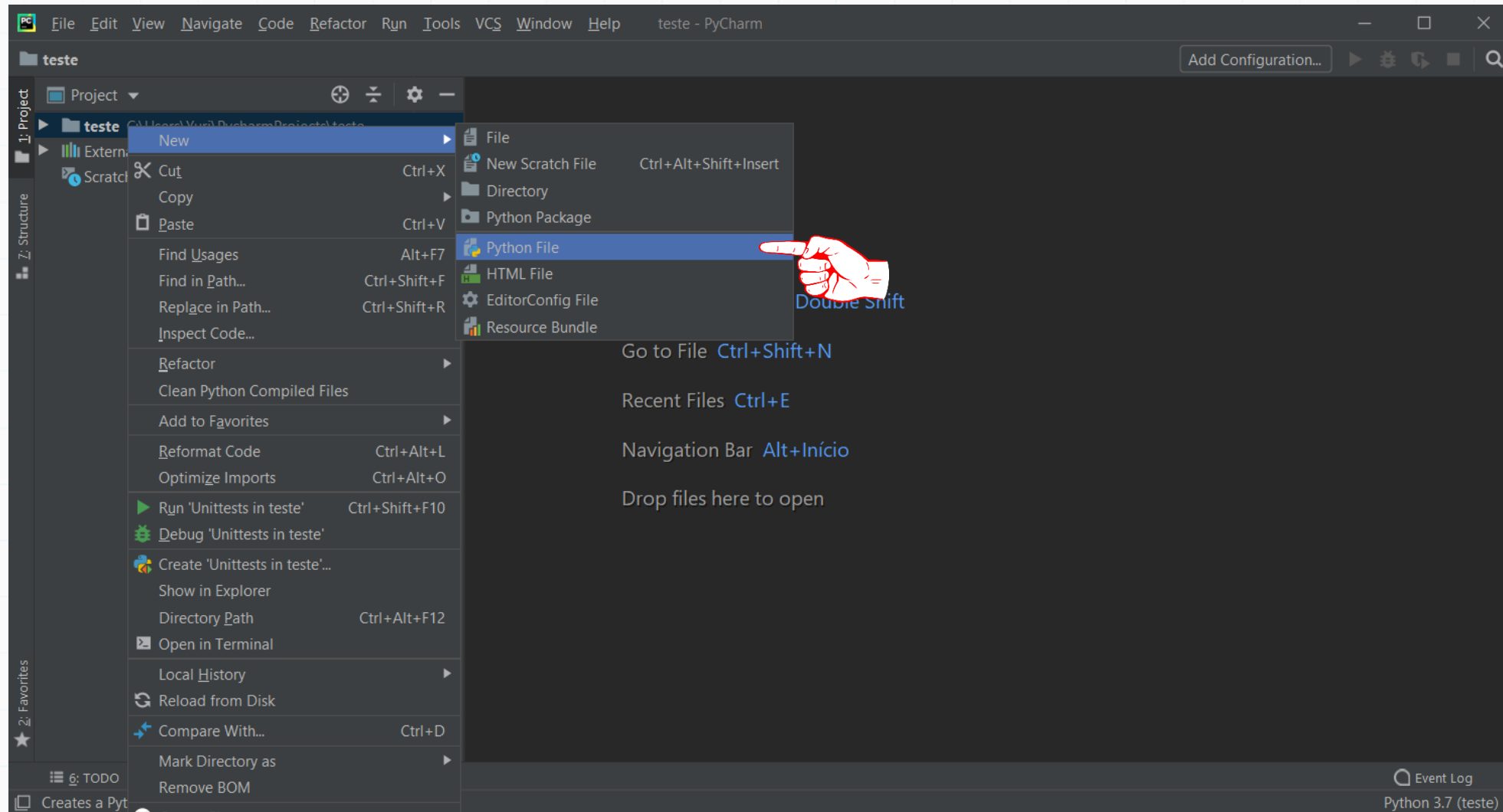
clique com botão  
direito no nome do  
projeto



# 1) Como um programa está organizado

Usando o Pycharm: Criar arquivo python dentro do projeto

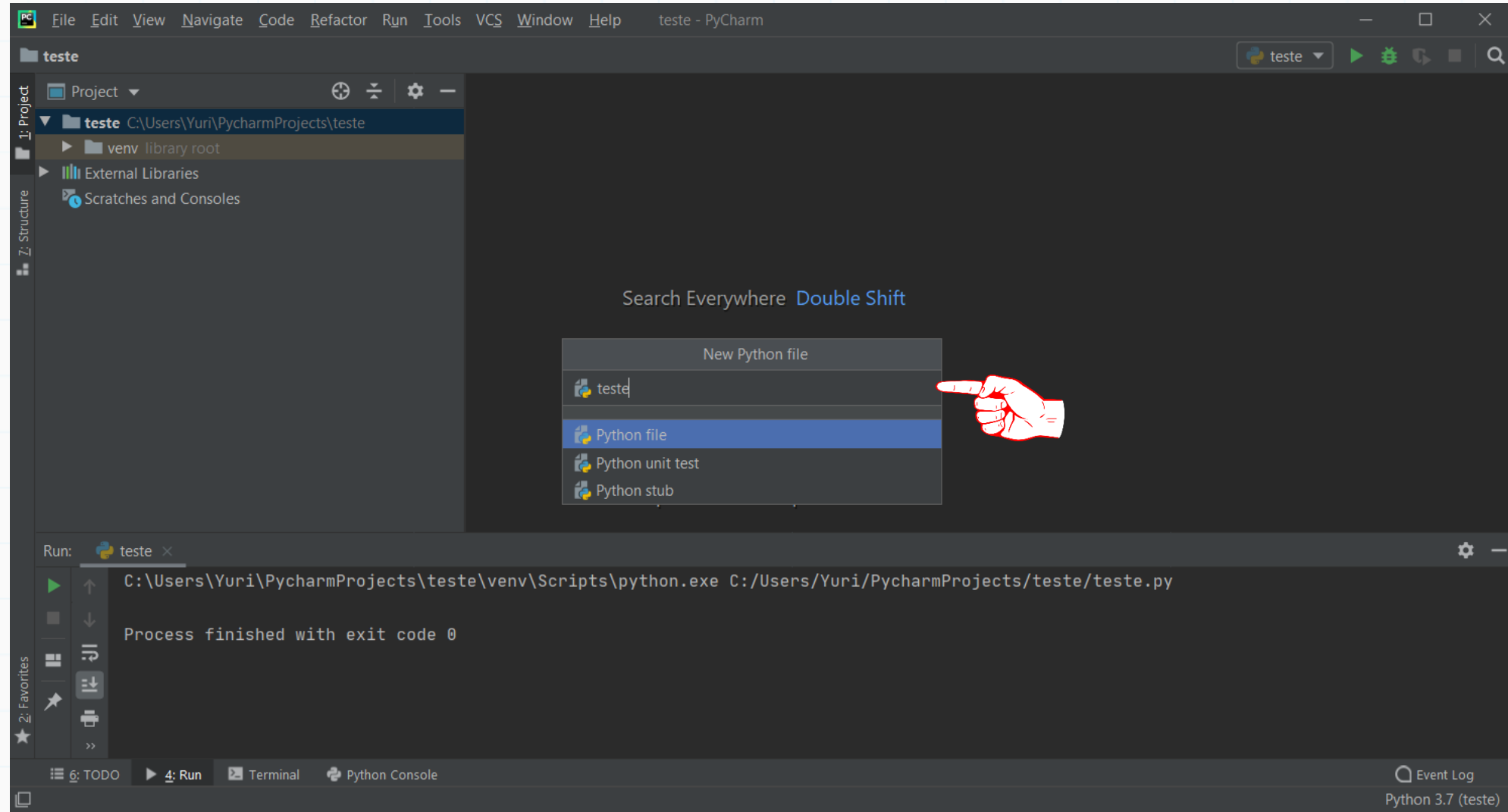
selecionar  
New/Python File



# 1) Como um programa está organizado

Usando o Pycharm: Criar arquivo python dentro do projeto

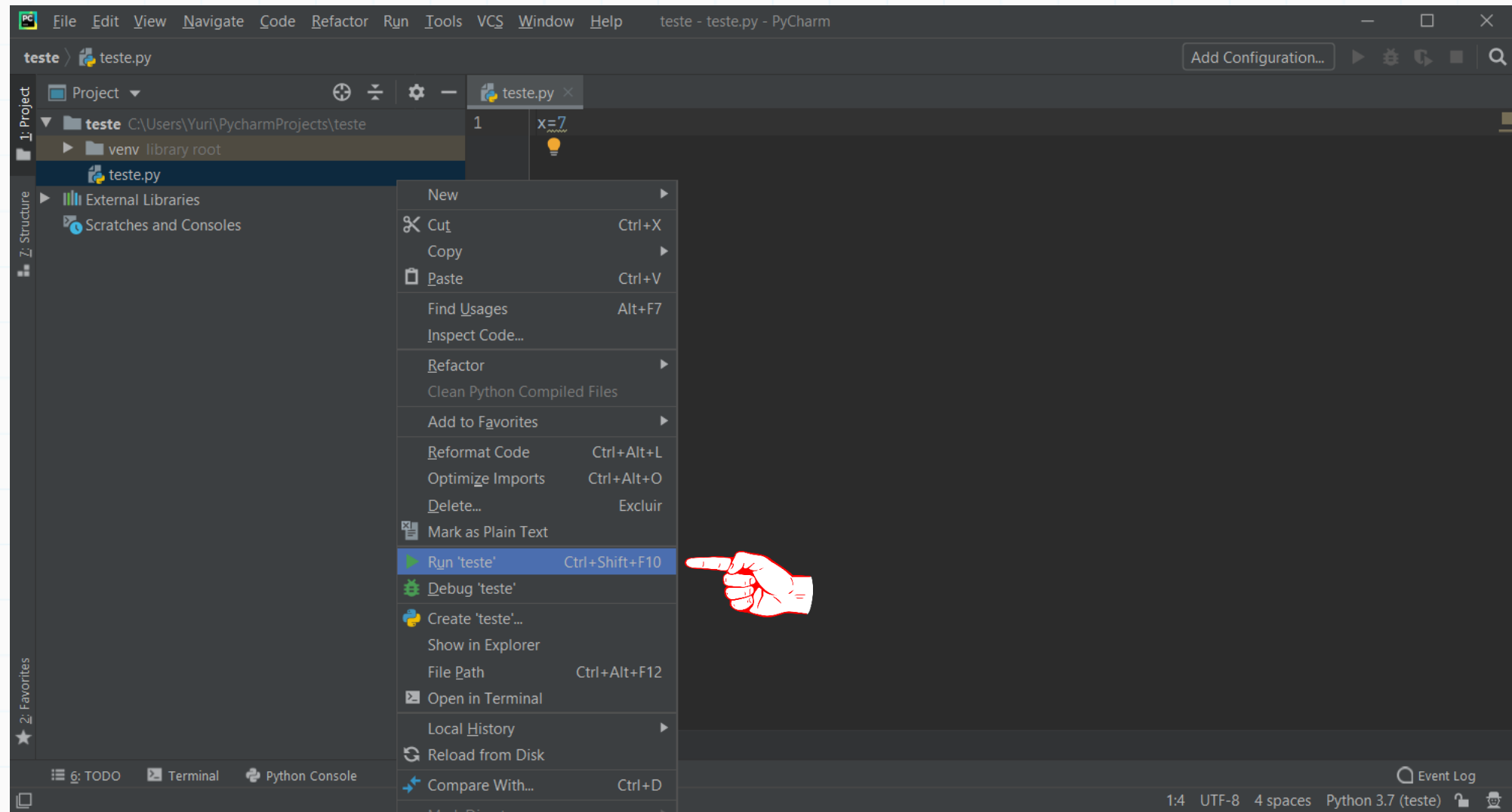
coloque  
nome



# 1) Como um programa está organizado

Usando o Pycharm: Criar arquivo python dentro do projeto

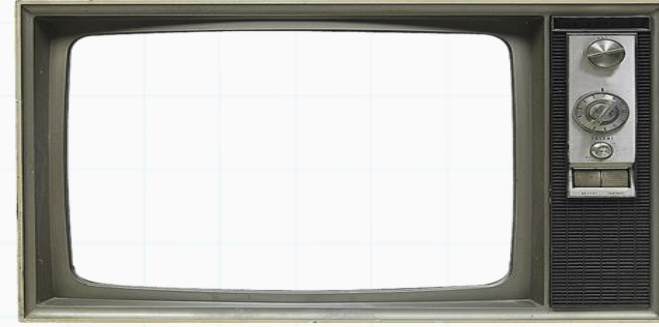
para executar,  
clique com botão  
direito em cima  
do nome do  
arquivo e clique  
em "Run"



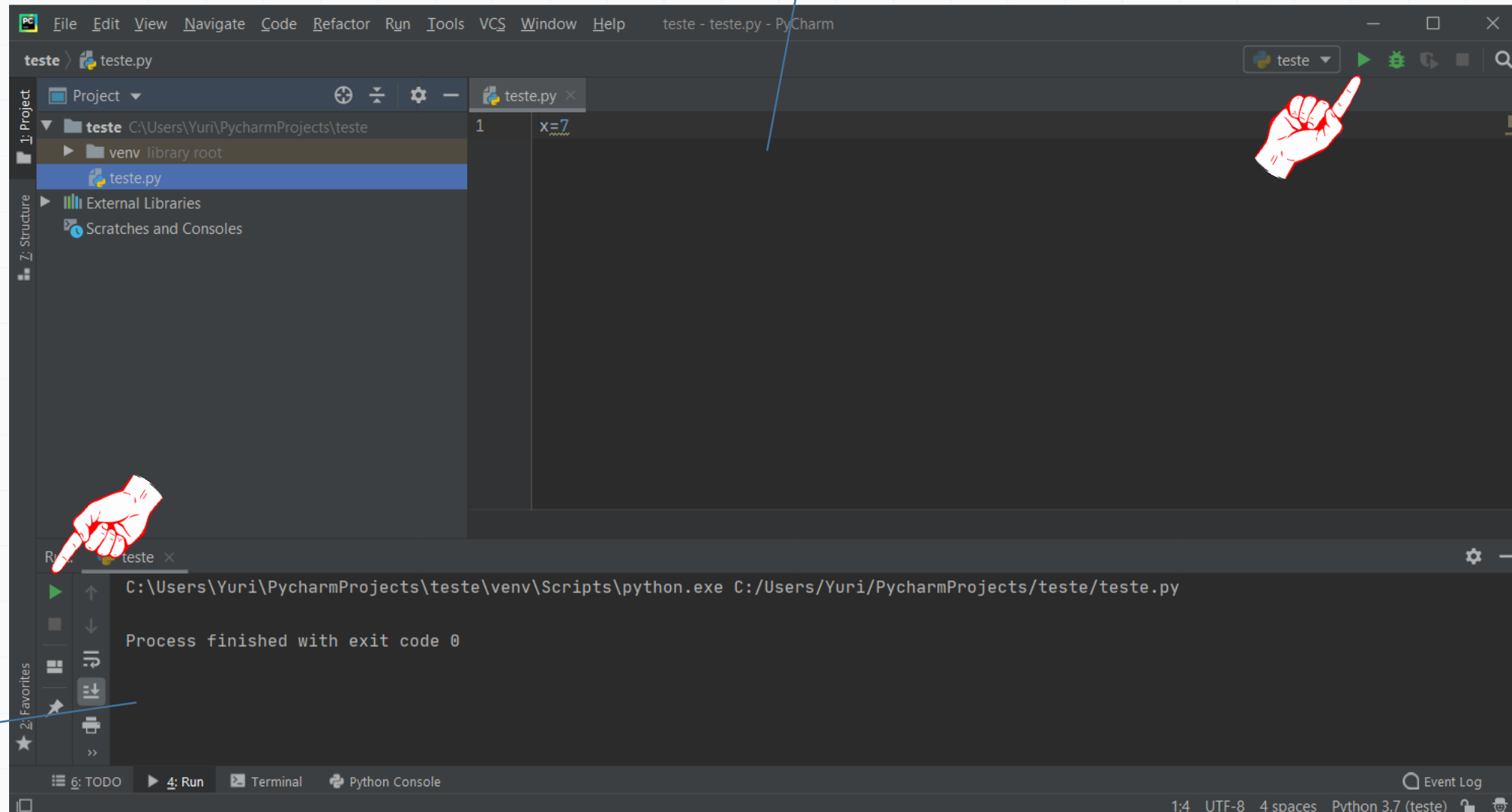
# 1) Como um programa está organizado

Usando o Pycharm: Criar arquivo python dentro do projeto

onde o código é  
escrito



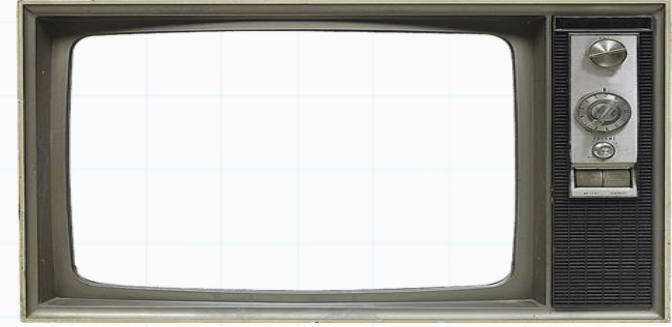
depois da  
primeira  
execução,  
podemos usar o  
botão



onde a saída é  
mostrada

# 1) Como um programa está organizado

- Sequência dos comandos é importante
  - Blocos devem ser criados usando indentação (com espaços ou tab).
- Ex: determinar os comandos dentro de uma repetição

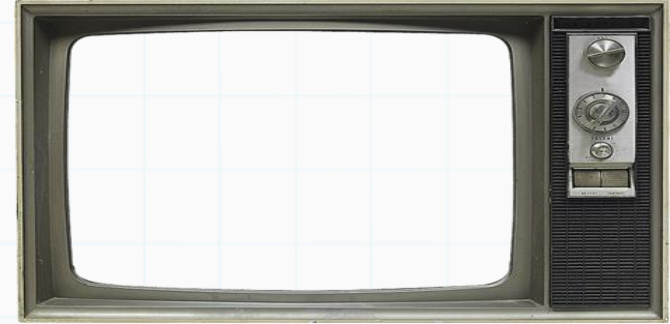
A screenshot of the Thonny Python IDE. The window title is "Thonny - C:\Users\Yuri\Desktop\teste.py @ 3:1". The menu bar includes "File", "Edit", "View", "Run", "Device", "Tools", and "Help". Below the menu is a toolbar with icons for opening files, saving, running, and stopping. The main editor area shows a file named "teste.py" with the following code:

```
1 RAIO=15
2 DIAMETRO=2*RAIO
3
```

The bottom panel is labeled "Shell" and shows the Python 3.7.7 (bundled) interpreter with a prompt ">>> |".

# 1) Como um programa está organizado

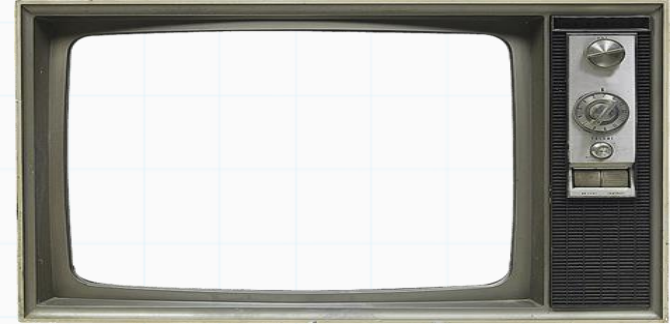
- Comentários são trechos do programa voltados para a leitura por humanos, e ignorados pelo compilador
- Começam com o símbolo **#**. Tudo na linha após **#** é ignorado pelo compilador.

A screenshot of the Thonny Python IDE. The window title is "Thonny - C:\Users\Yuri\Desktop\teste.py @ 3 : 29". The menu bar includes File, Edit, View, Run, Device, Tools, and Help. The toolbar contains icons for opening files, saving, running, and stopping. The editor shows a file named "teste.py" with the following code:

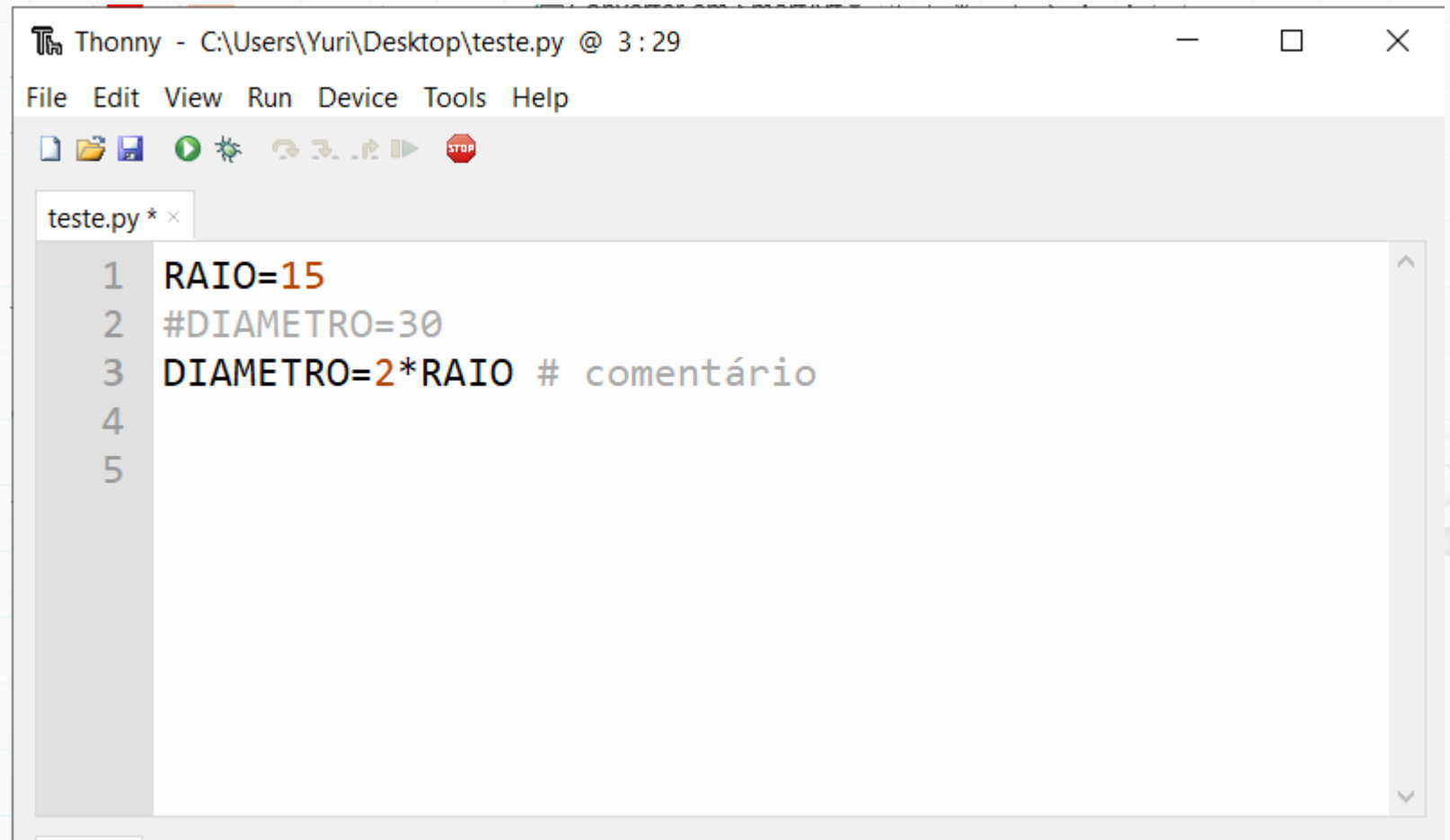
```
1 RAIO=15
2 #DIAMETRO=30
3 DIAMETRO=2*RAIO # comentário
4
5
```

# 1) Como um programa está organizado

- Comentários são trechos do programa voltados para a leitura por humanos, e ignorados pelo compilador
- Começam com o símbolo **#**. Tudo na linha após **#** é ignorado pelo compilador.



Use comentários  
para documentar  
seu código e fazer  
com que ele seja  
fácil de entender  
por outras pessoas



```
Thonny - C:\Users\Yuri\Desktop\teste.py @ 3 : 29
File Edit View Run Device Tools Help
teste.py * x
1 RAIO=15
2 #DIAMETRO=30
3 DIAMETRO=2*RAIO # comentário
4
5
```



## 2) Como atribuir valores a variáveis

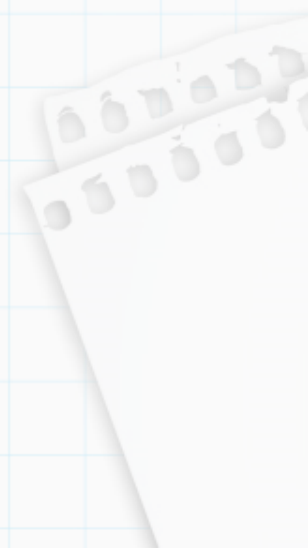
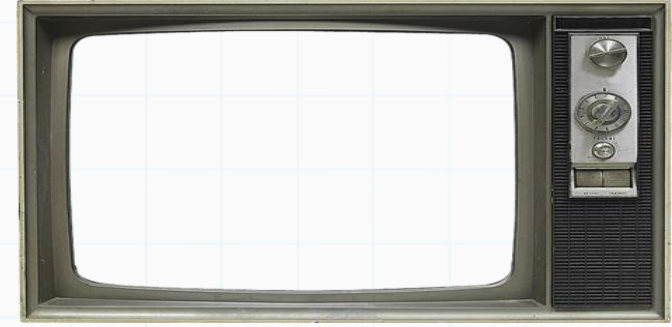
- Usamos o símbolo “=” para atribuir valores

formato:

<variável> = <expressão>



```
1 a=1
2 x=5
3 nome='Seya'
4 x=x-1
5
```



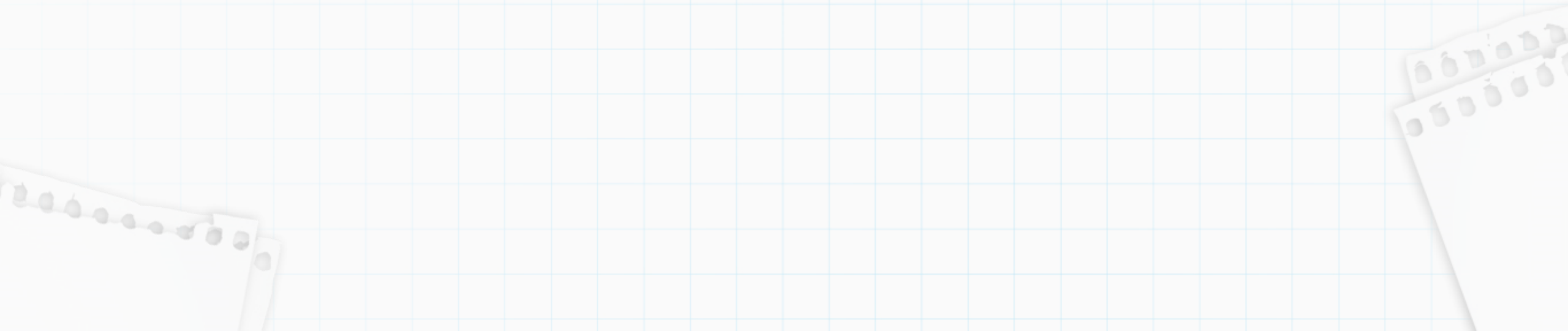
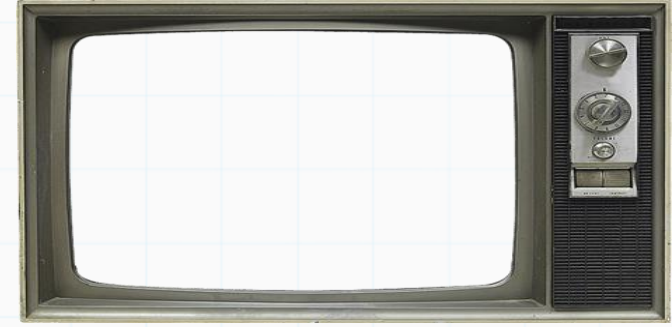
## 2) Como atribuir valores a variáveis

- Atribuição em linha

```
1 a=b=c=d=0
```



```
1 a=0  
2 b=0  
3 c=0
```



## 2) Como atribuir valores a variáveis

- Atribuição em linha

```
1 a=b=c=d=0
```



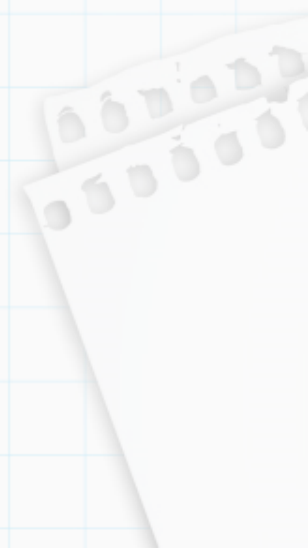
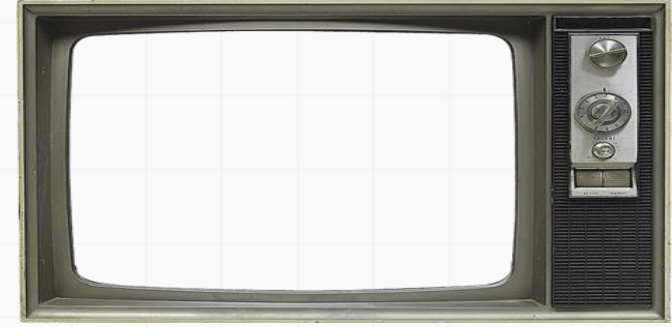
```
1 a=0  
2 b=0  
3 c=0
```

- Atribuição em lista

```
1 x,y,z=1,2,3|
```

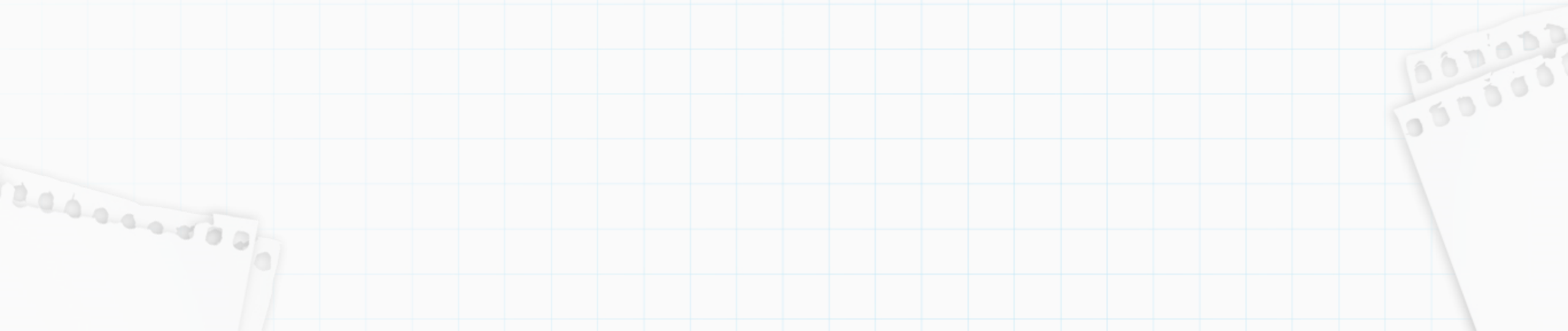
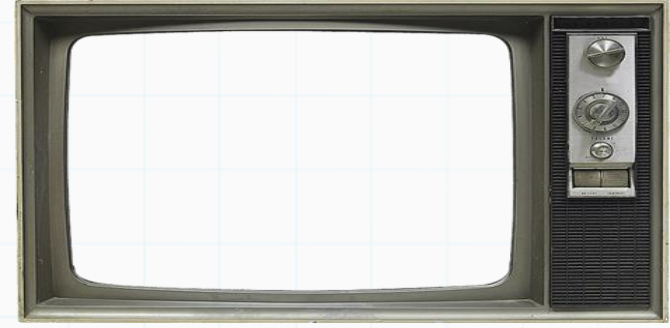


```
1 x=1  
2 y=2  
3 z=3|
```



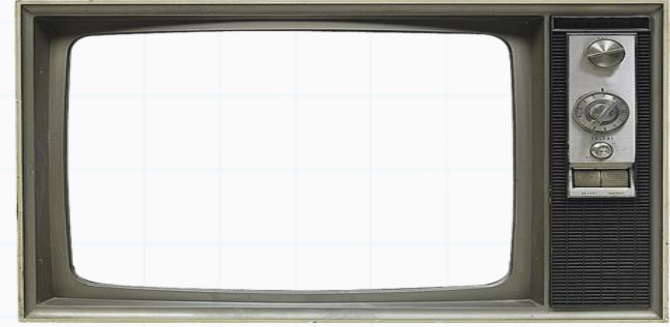
### 3) Quais os tipos de dados disponíveis

- Em Python, toda variável tem um tipo, com isso, o computador pode saber quais operações são permitidas.



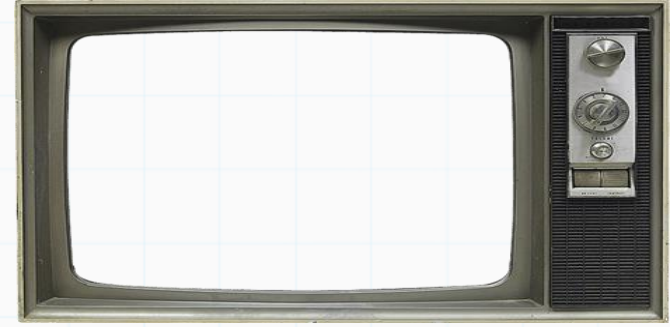
### 3) Quais os tipos de dados disponíveis

- Em Python, toda variável tem um tipo, com isso, o computador pode saber quais operações são permitidas.
- Os tipos podem ser divididos em três grupos:
  - Tipos numéricos (inteiro e float)
  - Tipos textuais (caractere e string)
  - Tipo lógico (booleano)



### 3) Quais os tipos de dados disponíveis

- Em Python, toda variável tem um tipo, com isso, o computador pode saber quais operações são permitidas.
- Os tipos podem ser divididos em três grupos:
  - Tipos numéricos (inteiro e float)
  - Tipos textuais (caractere e string)
  - Tipo lógico (booleano)
- Os tipos são definidos dinamicamente, pelo próprio Python  
Não é preciso dizer de que tipo é cada variável

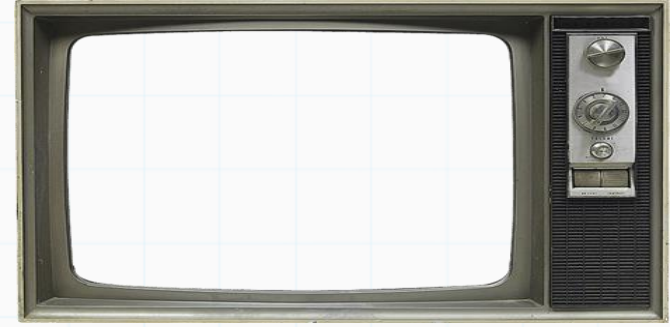


### 3) Quais os tipos de dados disponíveis

- Tipo Boolean:

```
1 x=True
2 y=False
3 |
```

no momento da  
atribuição x vira  
uma variável  
lógica

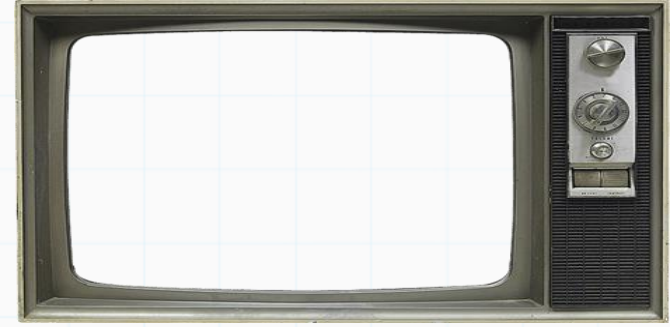


### 3) Quais os tipos de dados disponíveis

#### - Tipo Boolean:

```
1 x=True
2 y=False
3 |
```

no momento da  
atribuição x vira  
uma variável  
lógica



#### - Tipo String:

```
1 nome='Naruto'
2 sobrenome='Uzumaki'
3 letra='A'
4
5
```

o valor de uma  
string tem que  
estar entre aspas  
ou apóstrofes



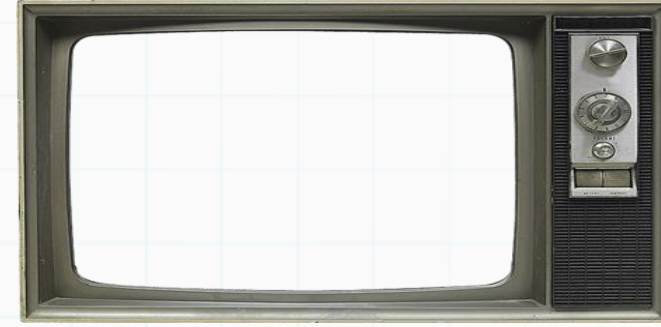


### 3) Quais os tipos de dados disponíveis

#### - Tipo Boolean:

```
1 x=True
2 y=False
3 |
```

no momento da  
atribuição x vira  
uma variável  
lógica



#### - Tipo String:

```
1 nome='Naruto'
2 sobrenome='Uzumaki'
3 letra='A'
4
5
```

o valor de uma  
string tem que  
estar entre aspas  
ou apóstrofes

#### - Tipos Numéricos:

```
1 a=-1
2 b=19
3 c=2.5
4 d=0.4e-3
5
```

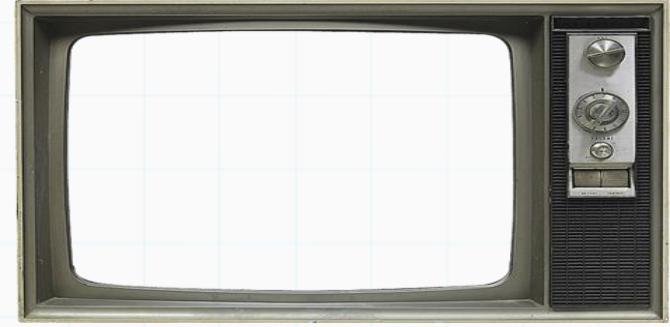
o “.” separa a  
parte inteira da  
fracionária

Tipo é determinado  
**automaticamente**  
pelo Python no  
momento de  
criação da variável

0.4x10<sup>-3</sup>

### 3) Quais os tipos de dados disponíveis

-Uma vez que uma variável tenha um valor de um tipo, ele não pode ser usado como se fosse de outro tipo



```
1 x=5
2 y='10'
3 z= x+y
4
5
6
```

Shell x

```
>>> %Run teste.py
```

```
Traceback (most recent call last):
```

```
  File "C:\Users\Yuri\Desktop\teste.py", line 3, i
n <module>
```

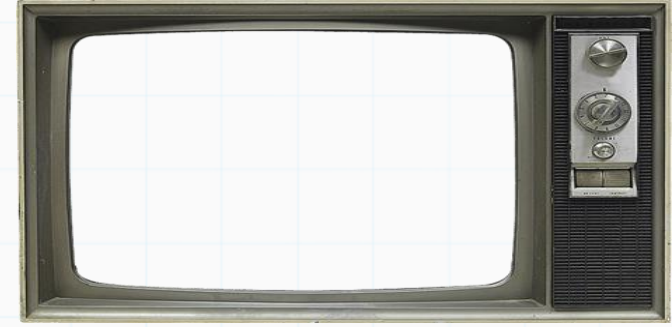
```
    z= x+y
```

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

```
>>>
```

### 3) Quais os tipos de dados disponíveis

-Uma vez que uma variável tenha um valor de um tipo, ele não pode ser usado como se fosse de outro tipo, só se você trocar !



```
1 x=5
2 y='10'
3 y=10
4 z=x+y
5 print(z)
```

Shell x

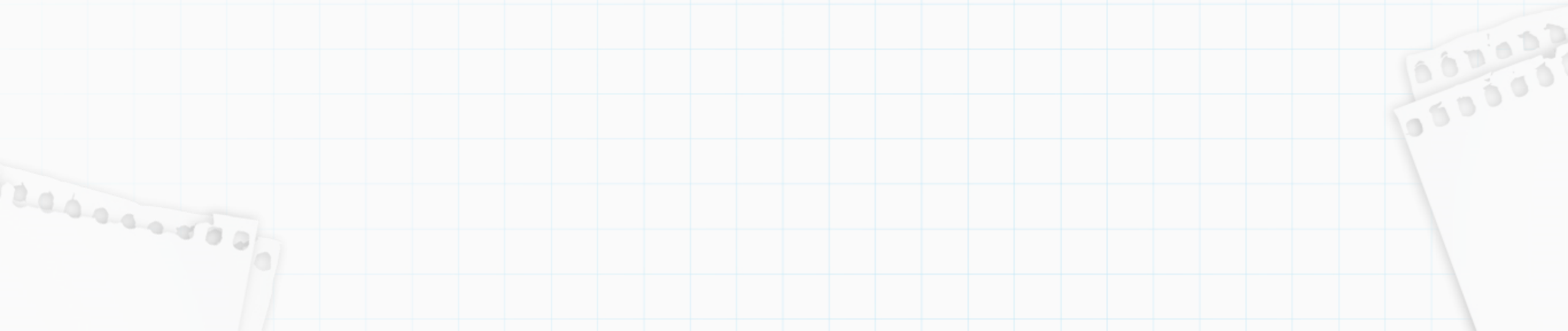
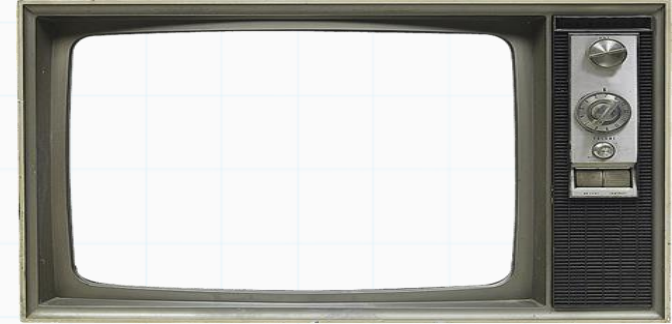
>>>

>>> %Run teste.py

15

## 4) Como dar nomes as variáveis

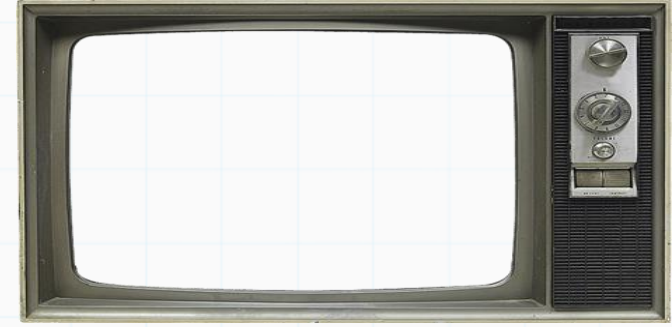
- São sensíveis a caixa
- Podem ter tamanho ilimitado (mas evite abusos)
- Devem começar com letra ou underline ( \_ )
- Outros caracteres podem ser letras, números ou underline
- Não podem ter espaço nem acentos
- Não podem ser uma palavra reservada da linguagem



## 4) Como dar nomes as variáveis

- São sensíveis a caixa
- Podem ter tamanho ilimitado (mas evite abusos)
- Devem começar com letra ou underline ( \_ )
- Outros caracteres podem ser letras, números ou underline
- Não podem ter espaço nem acentos
- Não podem ser uma palavra reservada da linguagem

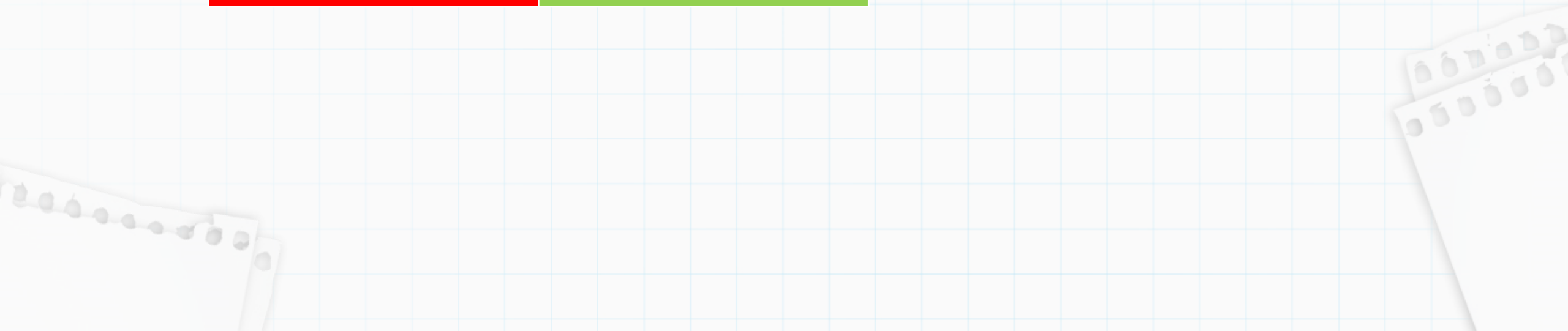
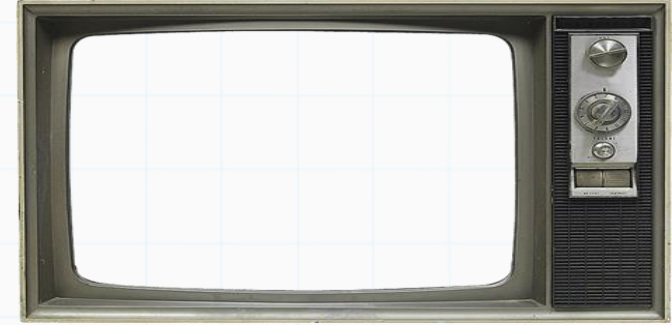
x	7dias_Samara
saldo\$	Largura-comprimento
_ÁREA	Taxa



## 4) Como dar nomes as variáveis

- São sensíveis a caixa
- Podem ter tamanho ilimitado (mas evite abusos)
- Devem começar com letra ou underline ( \_ )
- Outros caracteres podem ser letras, números ou underline
- Não podem ter espaço nem acentos
- Não podem ser uma palavra reservada da linguagem

x	7dias_Samara
saldo\$	Largura-comprimento
_ÁREA	Taxa



## 5) Como fazer entrada e saída (básica) de dados

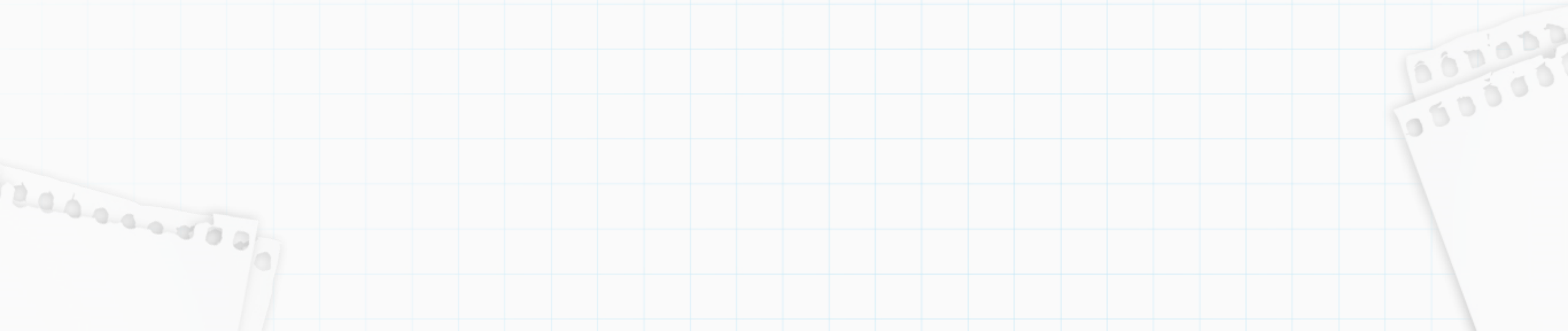
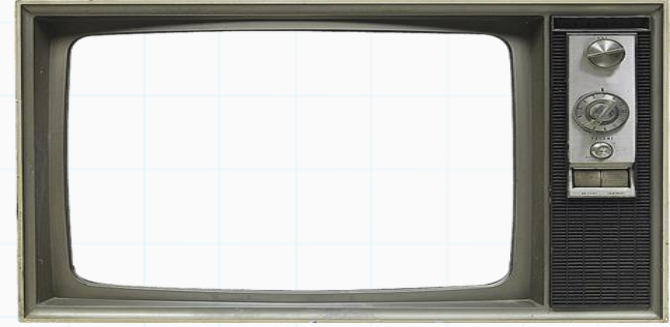
Entrada: Comando **input**

formato:

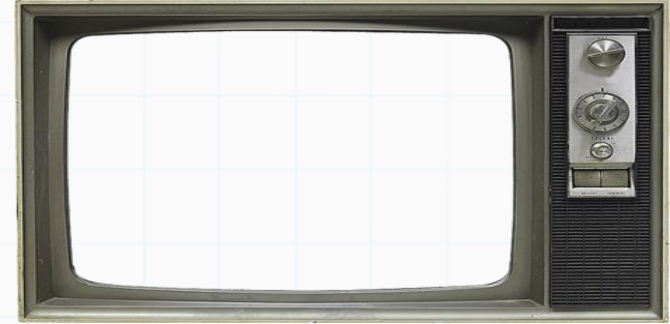
<variável> = input()

ou

<variável> = input('texto')



## 5) Como fazer entrada e saída (básica) de dados



Entrada: Comando **input**

formato:

<variável> = input()

ou

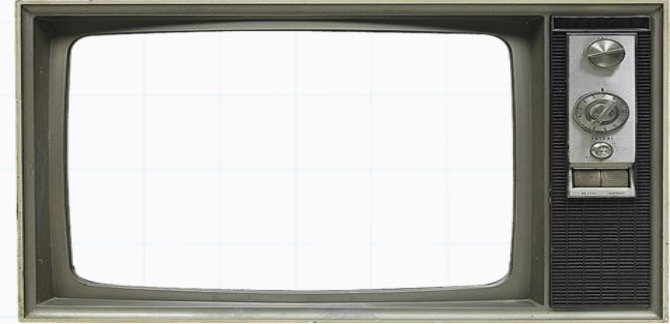
<variável> = input('texto')

```
1 nome=input()  
2  
3
```

```
1 nome=input("Digite seu nome|")  
2  
3  
4
```



## 5) Como fazer entrada e saída (básica) de dados



Entrada: Comando **input**

formato:

<variável> = input()

ou

<variável> = input('texto')

```
1 nome=input()  
2  
3
```

```
1 nome=input("Digite seu nome")  
2  
3  
4
```

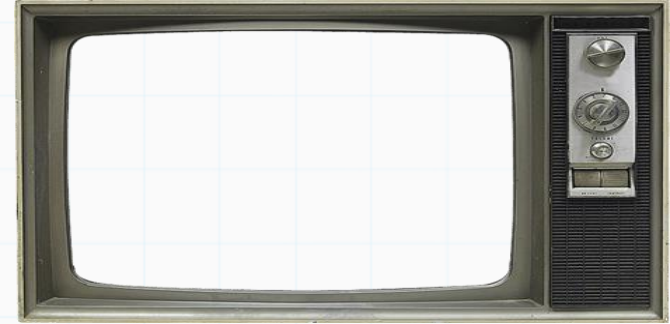
Usar int() ou float() para fazer o Python ler variáveis de tipo numérico

```
1 idade=int(input("Digite sua idade"))  
2 peso=float(input("Digite seu peso"))  
3
```

```
1 raio=float(input("Digite o raio"))  
2 diametro=raio*2  
3
```



## 5) Como fazer entrada e saída (básica) de dados



Entrada: Comando **input**

formato:

<variável> = input()

ou

<variável> = input('texto')

```
1 nome=input()
```

```
2
```

```
3
```

```
1 nome=input("Digite seu nome")
```

```
2
```

```
3
```

```
4
```

Usar int() ou float() para fazer o Python ler variáveis de tipo numérico

```
1 num = int(input("idade? "))
2 print(num)
```

Shell ×

idade? yuri

Traceback (most recent call last):

File "C:\Users\Yuri\Desktop\teste.py", line 1  
, in <module>

num = int(input("idade? "))

ValueError: invalid literal for int() with base 10: 'yuri'

...

CUIDADO!

## 5) Como fazer entrada e saída (básica) de dados

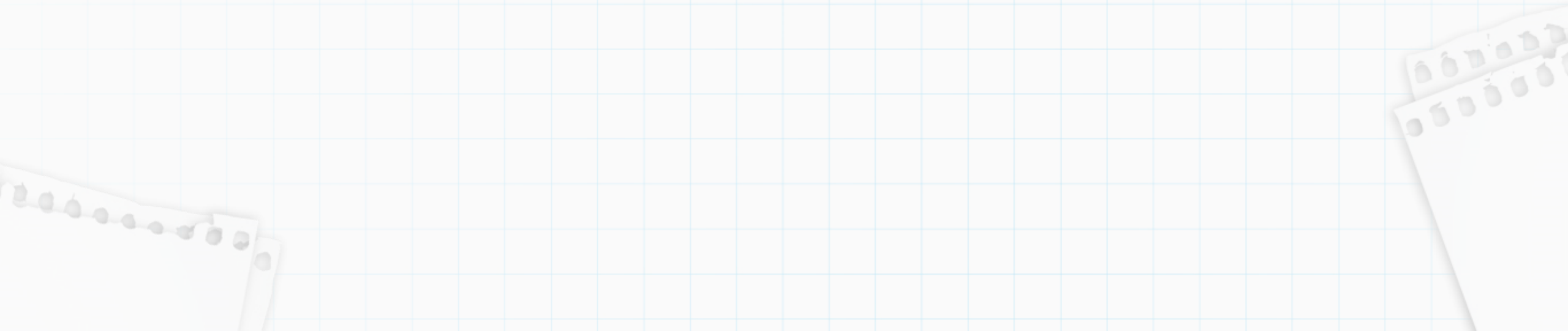
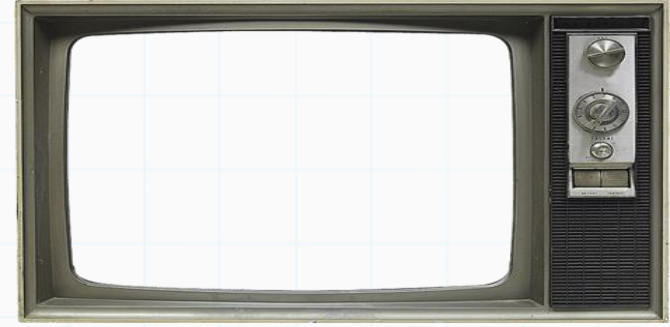
Saida: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

os argumentos podem ser:

- texto
- números
- variáveis



## 5) Como fazer entrada e saída (básica) de dados

Saída: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

os argumentos podem ser:

- texto
- números
- variáveis

```
1 print("Que pokemon é esse?")
```

```
2  
3  
4
```

Shell ×

```
>>>
```

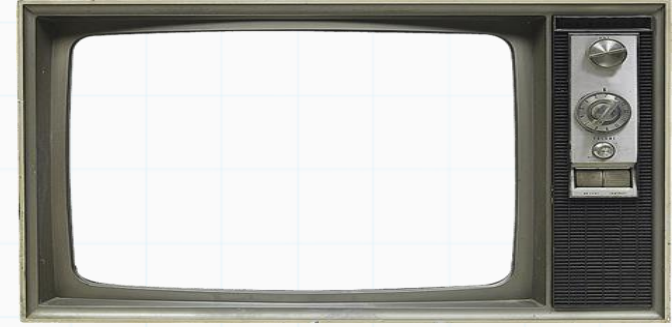
```
>>>
```

```
>>>
```

```
>>>
```

```
>>> %Run teste.py
```

```
Que pokemon é esse?
```



## 5) Como fazer entrada e saída (básica) de dados

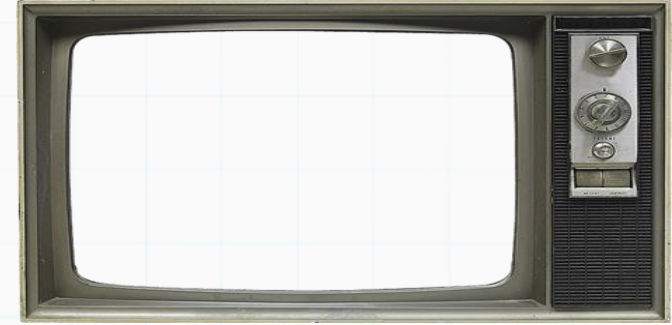
Saida: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

os argumentos podem ser:

- texto
- números
- variáveis



```
1 print("Eu peguei",3," charmander")
```

```
2
```

```
3
```

```
4
```

Shell ×

```
>>>
```

```
>>>
```

```
>>>
```

```
>>>
```

```
>>> %Run teste.py
```

```
Eu peguei 3 charmander
```



## 5) Como fazer entrada e saída (básica) de dados

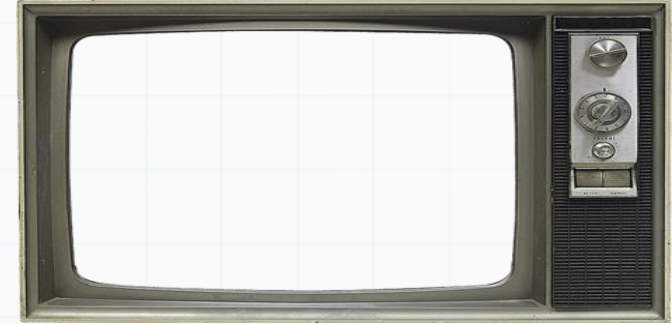
Saída: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

os argumentos podem ser:

- texto
- números
- variáveis



```
1 squirtle = int(input("quantos skirtles você tem ?"))
2 print("Ele tem ",squirtle," squirtles")
3
4
5
```

Shell ×

>>>

>>>

>>>

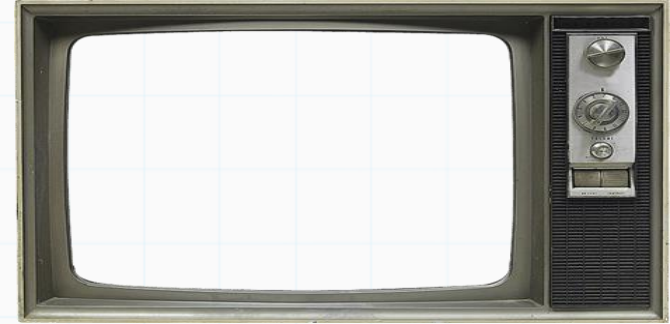
>>> %Run teste.py

quantos skirtles você tem ? 73

Ele tem 73 squirtles



## 5) Como fazer entrada e saída (básica) de dados



Saída: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

- A função print sempre pula uma linha ao final da impressão.
- Se você não quiser que pule uma linha, inclua como último argumento `end=""`

argumento `end=""` indica qual texto será colocado no final do `print`

```
1 print('3, ', end='')
2 print('4, ', end='')
3 print('5 ', end='')
4
5
6
7
```

Shell ×

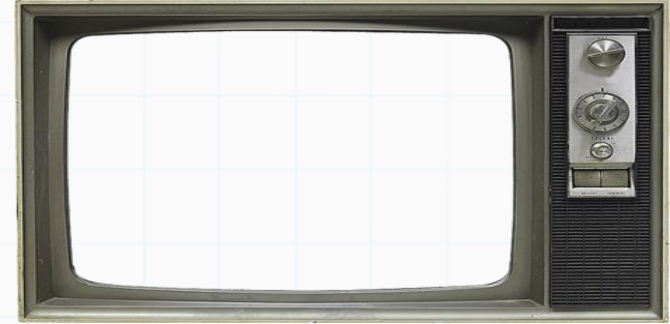
>>>

>>>

>>> %Run teste.py

3, 4, 5

## 5) Como fazer entrada e saída (básica) de dados



Saida: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

- A função print sempre pula uma linha ao final da impressão.
- Se você não quiser que pule uma linha, inclua como último argumento `end=""`
- Podemos especificar o número de casas decimais que deve ser impresso em um número ponto flutuante usando `%.Nf`, onde N especifica o número de casas decimais.

impressão básica

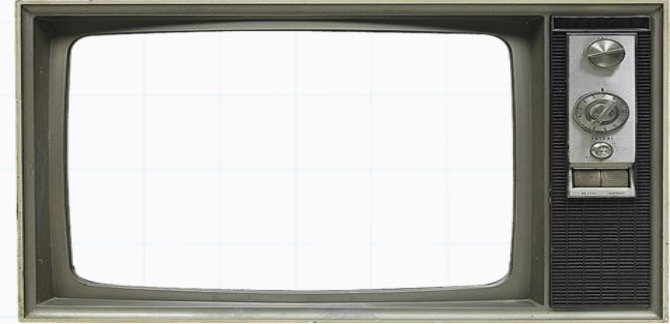
```
1 pi      = 3.1415
2 raio    = 7
3 area    = pi*raio*raio
4
5 print('área = ',area)
6
7
8
9
```

Shell ×

```
>>>
>>>
>>> %Run teste.py
      área = 153.9335
```



## 5) Como fazer entrada e saída (básica) de dados



Saída: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

- A função print sempre pula uma linha ao final da impressão.
- Se você não quiser que pule uma linha, inclua como último argumento `end=""`
- Podemos especificar o número de casas decimais que deve ser impresso em um número ponto flutuante usando `%.Nf`, onde N especifica o número de casas decimais.

impressão  
formatada

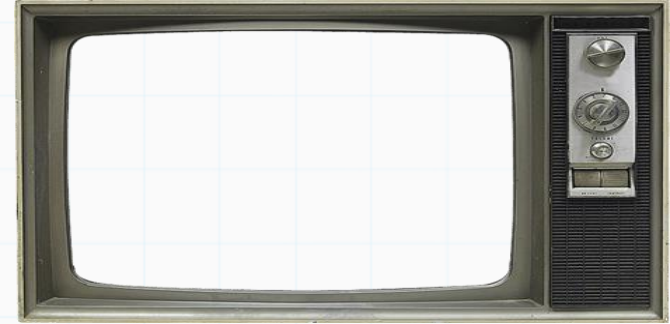
```
1 pi      = 3.1415
2 raio    = 7
3 area    = pi*raio*raio
4
5 print('raio = %.2f'%raio)
6 print('área = %.2f'%area)
7
8
9
10
```

Shell ×

```
>>>
>>> %Run teste.py

raio = 7.00
área = 153.93
```

## 5) Como fazer entrada e saída (básica) de dados



Saída: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

- A função print sempre pula uma linha ao final da impressão.
- Se você não quiser que pule uma linha, inclua como último argumento `end=""`
- Podemos especificar o número de casas decimais que deve ser impresso em um número ponto flutuante usando `%.Nf`, onde N especifica o número de casas decimais.

Múltiplos valores formatados

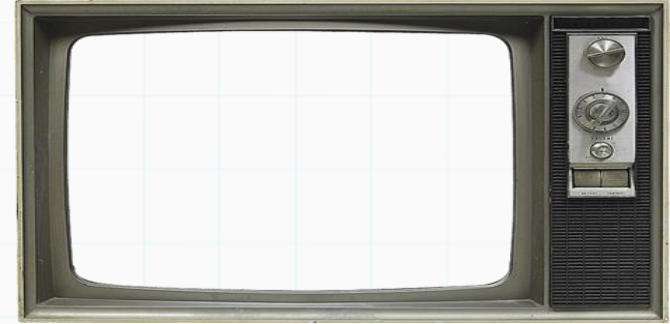
impressão  
formatada

```
1 a = 5.2345753
2 b = 2.9764897
3 c = 0.2346732
4
5 print("a=%.3f b=%.3f c=%.3f" %(a,b,c))
```

Shell ×

a=5.235 b=2.976 c=0.235

## 5) Como fazer entrada e saída (básica) de dados



Saída: Comando **print**

formato:

`print(arg1, arg2, arg3, ....)`

- A função print sempre pula uma linha ao final da impressão.
- Se você não quiser que pule uma linha, inclua como último argumento `end=""`
- Podemos especificar o número de casas decimais que deve ser impresso em um número ponto flutuante usando `%.Nf`, onde N especifica o número de casas decimais.

impressão  
formatada

Impressão de inteiros também com `%d`

```
1 a = 5.2345753
2 b = 15
3 c = 0.2346732
4
5 print("a=%.3f b=%d c=%.3f" %(a,b,c))
```

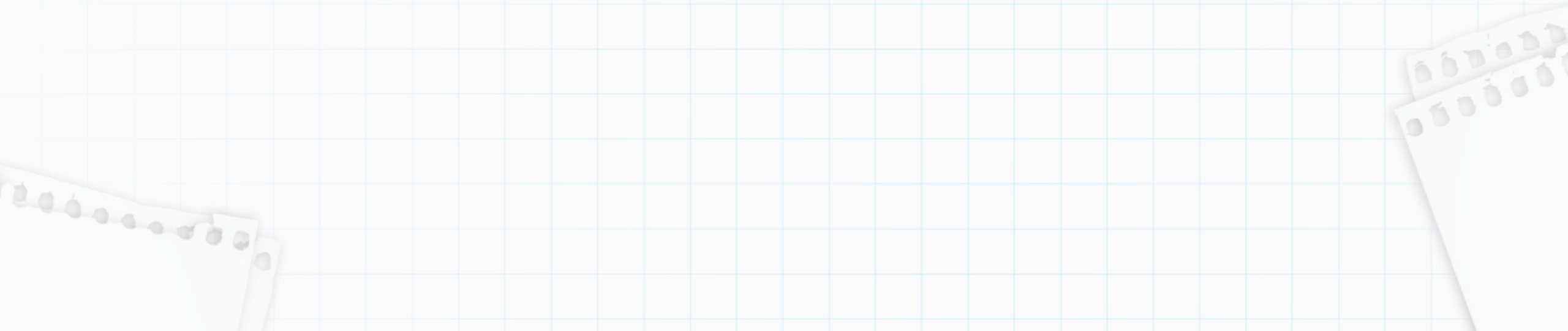
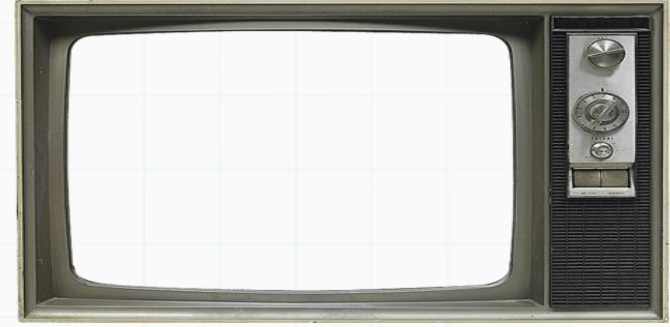
Shell ×

```
a=5.235 b=15 c=0.235
```

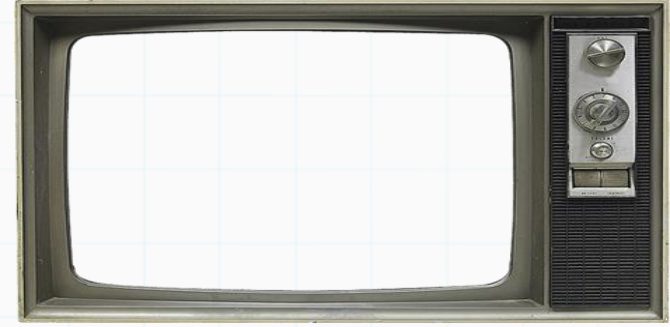
## 5) Como fazer entrada e saída (básica) de dados

Qual a saída do programa ?

```
1  x = 1.0
2  y = 2.0
3  z = 3.0
4
5  x = -x
6  y = y - 1
7  z = z + x
8  z = z + x - y
9  print("x =", x, ", y =", y, ", z =", z)
10
```



## 5) Como fazer entrada e saída (básica) de dados

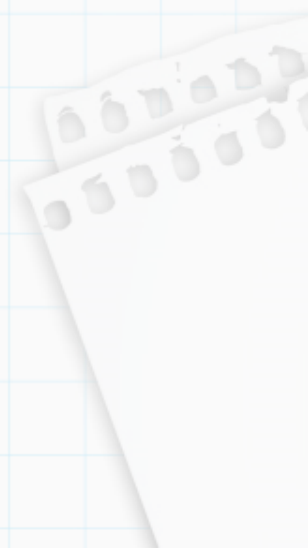


Qual a saída do programa ?

```
1  x = 1.0
2  y = 2.0
3  z = 3.0
4
5  x = -x
6  y = y - 1
7  z = z + x
8  z = z + x - y
9  print("x =", x, ", y =", y, ", z =", z)
10
```

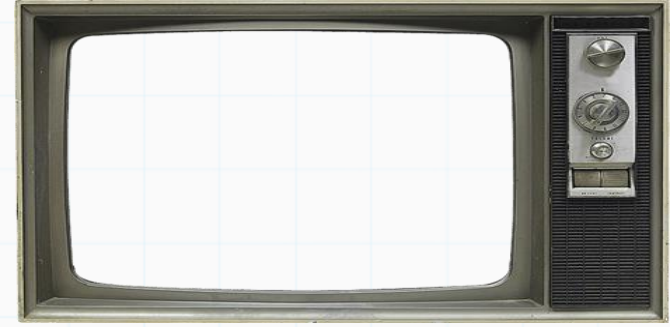
```
>>> %Run teste.py
```

```
x = -1.0 , y = 1.0 , z = 0.0
```

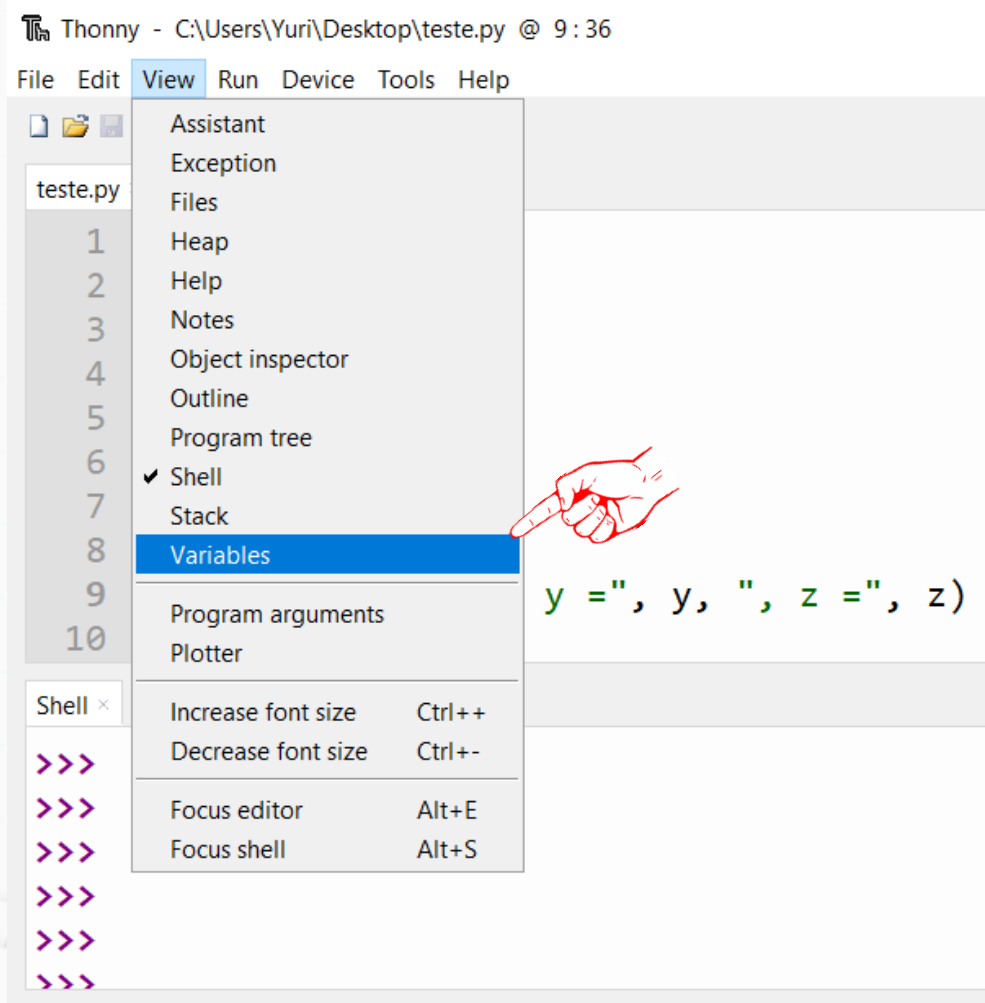


# Depuração

É possível executar o programa passo a passo para acompanhar sua execução (muito bom para encontrar erros)

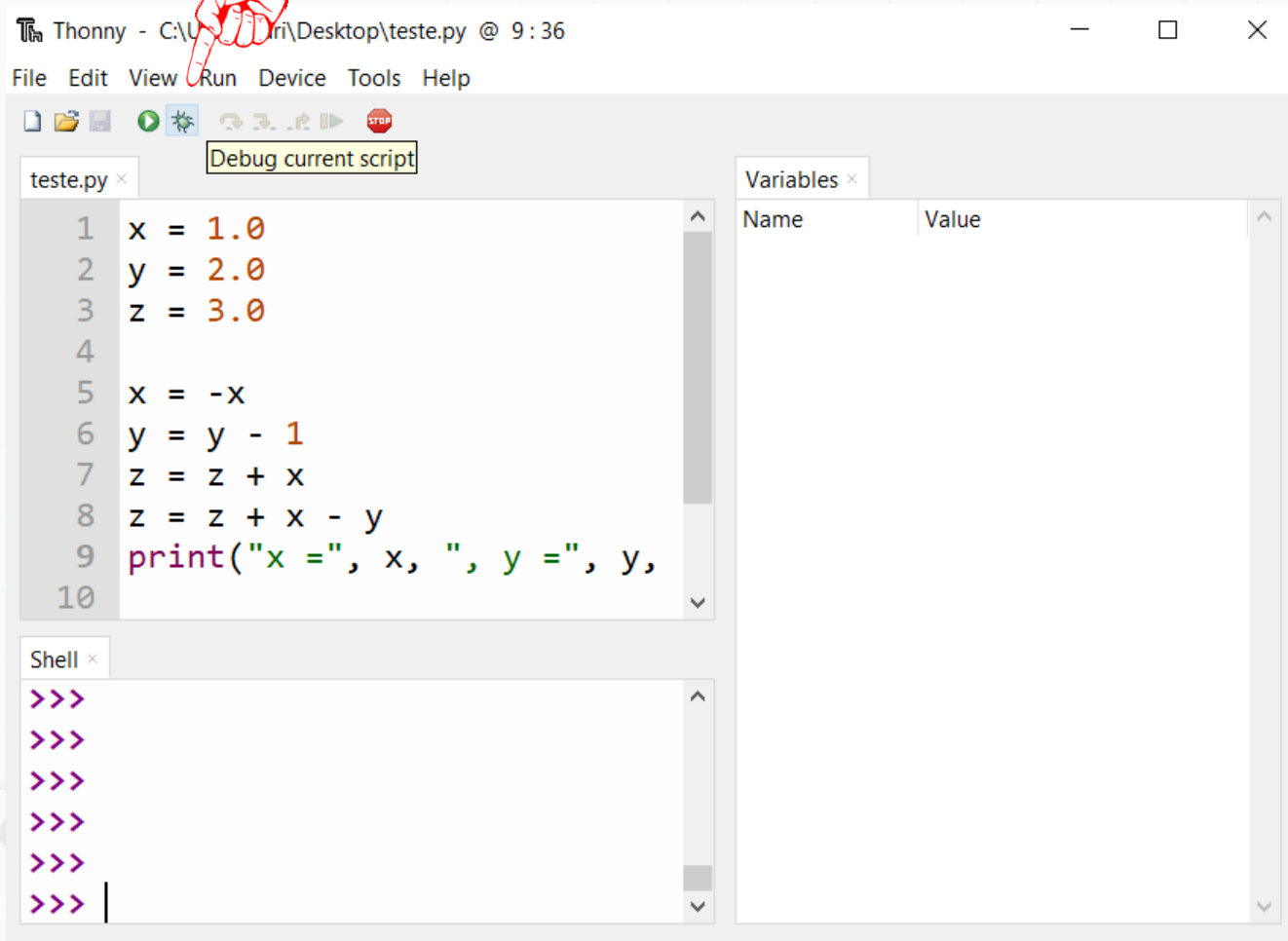
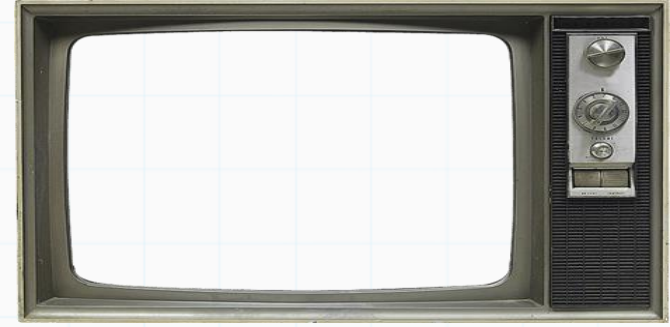


Vamos primeiro abrir  
a janela com os  
valores das variáveis



# Depuração

É possível executar o programa passo a passo para acompanhar sua execução (muito bom para encontrar erros)

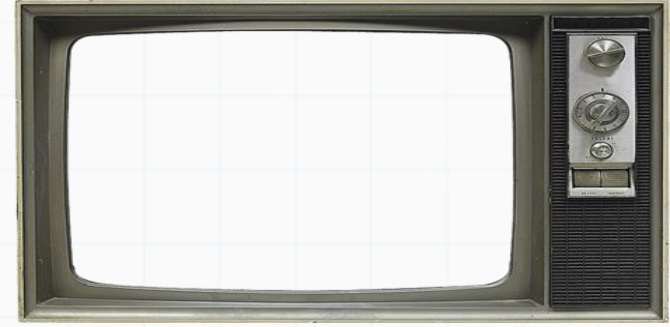


Vamos executar o  
depurador



# Depuração

É possível executar o programa passo a passo para acompanhar sua execução (muito bom para encontrar erros)



Thonny - C:\Users\Yuri\Desktop\teste.py @ 9:36

File Edit View Run Device Tools Help

```
teste.py x
3  z = 3.0
4
5  x = -x
6  y = y - 1
7  z = z + x
8  z = z + x - y
9  print("x =", x, ", y =", y,
10
11
12
```

Variables x

Name	Value
x	-1.0
y	1.0
z	0.0

Shell x

```
>>>
>>>
>>> %Debug teste.py
x = -1.0 , y = 1.0 , z = 0.0
>>>
```

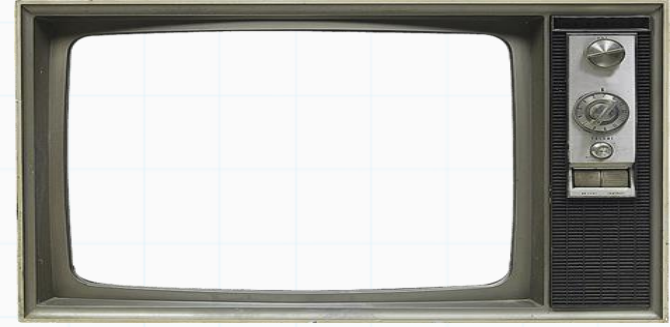
podemos acompanhar os valores das variáveis no decorrer da execução

Vamos testar esse exemplo no Thonny  
Exemplo !!!





Até a próxima



Slides baseados no curso de Vanessa Braganholo

