

CREACIÓN DE PROGRAMAS EN PYTHON

Grace Murray Hopper (1906-1992)

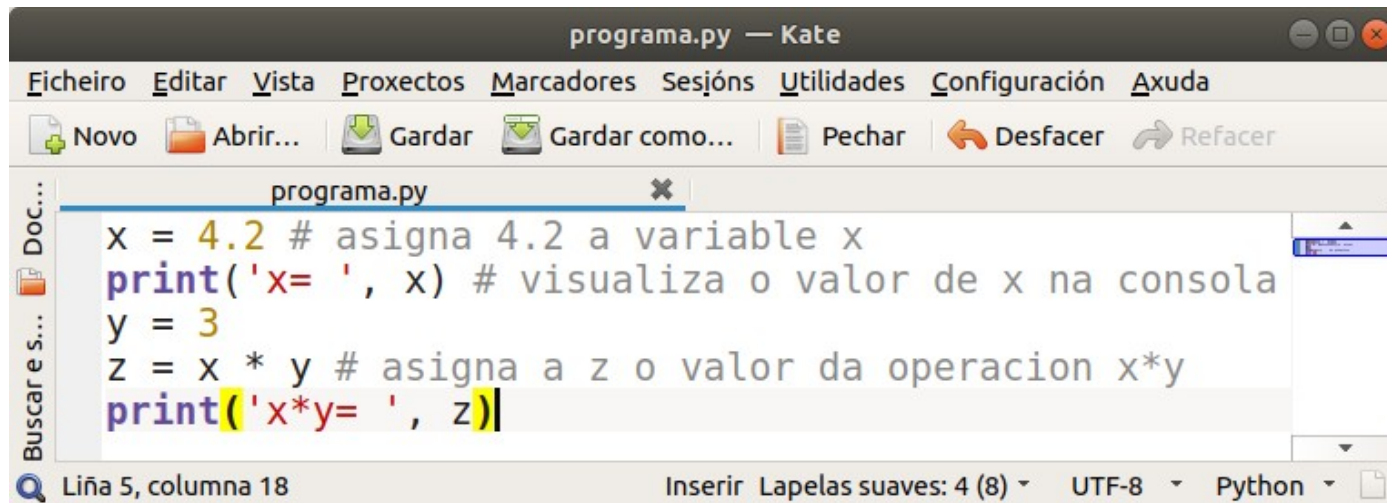


- Inventora de compiladores das linguaxes de programación A0 e B0, para o cálculo de nóminas
- Contra-almirante da US Navy
- Muller do ano en Informática (1969), primeira muller na British Computer Society (1973)

Scripts en Python

- A tradución literal sería guións, aínda que prefiro chamarlles **programas**.
- Un programa é un arquivo que contén código fonte en linguaxe Python.
- Os programas teñen extensión **.py**
- Para escribir os programas necesitas un editor de texto, mellor que teña resaltada a sintaxe de Python.
- Escribe un programa con comandos de Python, gardao con nome.py e executao desde o terminal de Linux con: **python3 nome.py**

Exemplo de programa en Python

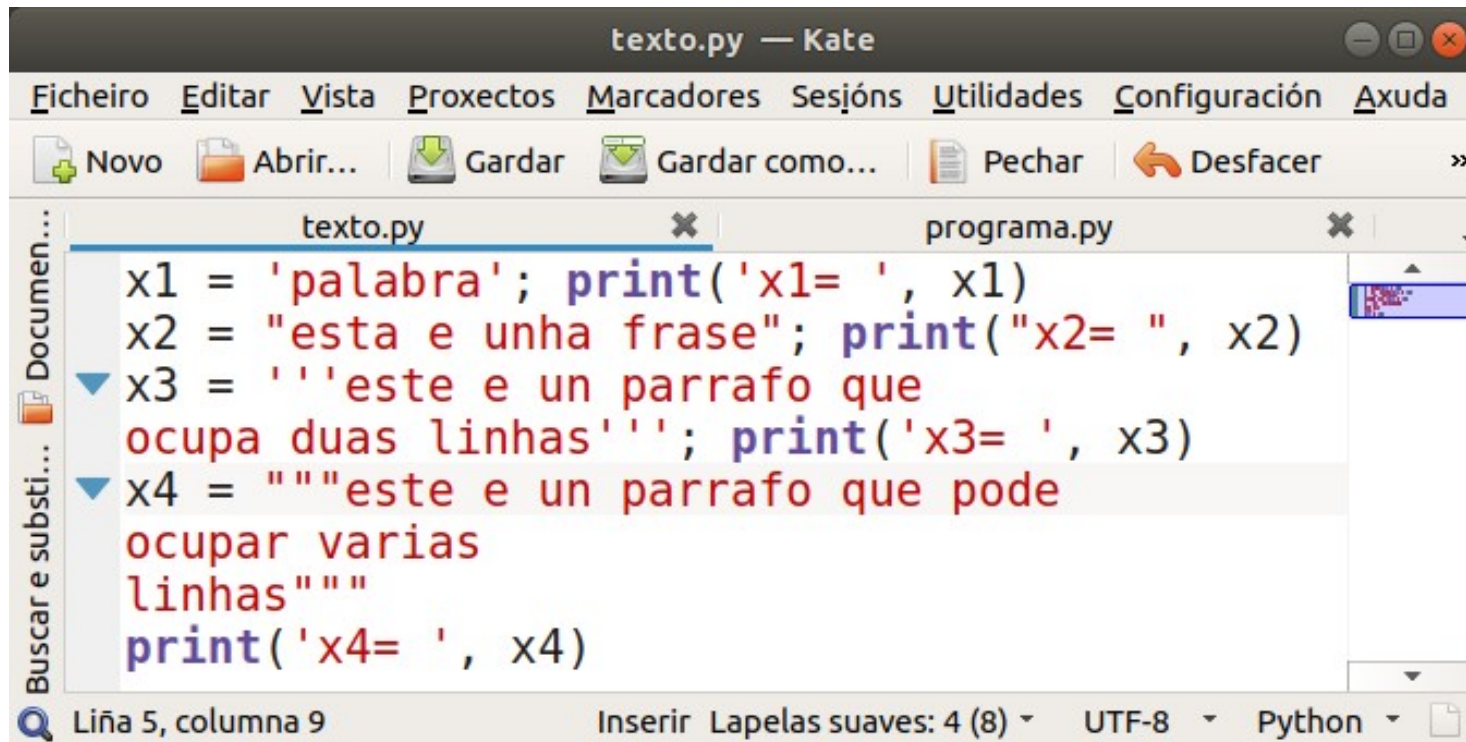


```
x = 4.2 # asigna 4.2 a variable x
print('x= ', x) # visualiza o valor de x na consola
y = 3
z = x * y # asigna a z o valor da operacion x*y
print('x*y= ', z)
```

- As sentencias en Python rematan co cambio de liña. Nembargantes, Python permite continuar unha sentencia na seguinte liña co carácter `\`.
 - `x = 4.2 + \`
 - `7.4 + \`
 - `8.4`
 - Equivalente a `x=4.2 + 7.4 + 8.4`
- Con sentencias que conteñan `[]`, `{}` ou `()` non se necesita usar o carácter de continuación de liña `\`. Por exemplo, na definición dunha lista:
 - `dias = ['luns', 'martes',`
 - `'mercores', 'xoves']`

Texto nos programas de Python

- Para poñer varias sentencias nunha liña, separalas por punto e coma (;).
- Python acepta comillas simples ('), dobres (") e triples (''' ou """) para especificar o comezo e final dun texto literal ou cadea de caracteres. Exemplos:



The screenshot shows the Kate text editor window titled 'texto.py — Kate'. The menu bar includes 'Ficheiro', 'Editar', 'Vista', 'Proxectos', 'Marcadores', 'Sesións', 'Utilidades', 'Configuración', and 'Axuda'. The toolbar contains icons for 'Novo', 'Abrir...', 'Gardar', 'Gardar como...', 'Pechar', and 'Desfacer'. The editor has two tabs: 'texto.py' (active) and 'programa.py'. The code in 'texto.py' is as follows:

```
x1 = 'palabra'; print('x1= ', x1)
x2 = "esta e unha frase"; print("x2= ", x2)
x3 = '''este e un parrafo que
ocupa duas linhas'''; print('x3= ', x3)
x4 = """este e un parrafo que pode
ocupar varias
linhas"""
print('x4= ', x4)
```

The status bar at the bottom indicates 'Liña 5, columna 9', 'Inserir', 'Lapelas suaves: 4 (8)', 'UTF-8', and 'Python'.

Execución de Scripts en Windows

- Os arquivos **.py** xa están asociados ó interprete de Python, polo que con premer dúas veces sobre o arquivo xa se executa o programa.
- Os resultados da execución visualízanse na consola. Se a execución é moi rápida pode que non de tempo a vela.
- Truco: engadir ó arquivo co código unha liña que agarde a entrada dalgún dato por teclado: **input()**
- Así a consola agarda a que se introduza algo por teclado.

Máis sobre a execución de scripts en linux

- Executar un programa no terminal de Linux:

```
python3 nome.py
```

- Desde o entorno interactivo executar:

```
>>> run nome.py
```

- Execución implícita do arquivo nome.py:

- Introducir na primeira liña do arquivo **#!/usr/bin/python3** (esta é a ruta onde está instalado python).

- Darlle permisos de execución ó arquivo nome.py con:

```
chmod +x nome.py
```

- Executar o arquivo no terminal con:

```
./nome.py
```

Entornos de ejecución para Python

- Instalación dun entorno de desenvolvemento IDE (Interface Development Environment) no ordenador. O IDE máis popular é SPIDER (<https://www.spyder-ide.org/>).
- Execución de comandos ou programas online desde un navegador (non necesitar ter instalado Python no ordenador). Algúns exemplos podes atopalos no enlace:
<https://quintagroup.com/cms/python/online-interpreter>.

Execución desde o entorno SPYDER

The screenshot displays the Spyder Python IDE interface. The main editor window shows a Python script named `temp.py` with the following code:

```
6
7 import pylab
8 from numpy import cos, linspace, pi, sin, random
9 from scipy.interpolate import splprep, splev
10
11 # %% Generate data for analysis
12
13 # Make ascending spiral in 3-space
14 t = linspace(0, 1.75 * 2 * pi, 100)
15
16 x = sin(t)
17 y = cos(t)
18 z = t
19
20 # Add noise
21 x += random.normal(scale=0.1, size=x.shape)
22 y += random.normal(scale=0.1, size=y.shape)
23 z += random.normal(scale=0.1, size=z.shape)
24
25
26 # %% Perform calculations
27
28 # Spline parameters
29 smoothness = 3.0 # Smoothness parameter
30 k_param = 2 # Spline order
31 nests = -1 # Estimate of number of knots needed (-1 = maximal)
32
33 # Find the knot points
34 knot_points, u = splprep([x, y, z], s=smoothness, k=k_param, nests=-1)
35
36 # Evaluate spline, including interpolated points
37 xnew, ynew, znew = splev(linspace(0, 1, 400), knot_points)
38
39
40 # %% Plot results
41
42 # TODO: Rewrite to avoid code smell
43 pylab.subplot(2, 2, 1)
44 data, = pylab.plot(x, y, 'bo-', label='Data with X-Y Cross Section')
45 fit, = pylab.plot(xnew, ynew, 'r-', label='Fit with X-Y Cross Section')
46 pylab.legend()
47 pylab.xlabel('x')
48 pylab.ylabel('y')
```

The Variable explorer shows the following variables:

Name	Type	Size	Value
bars	container.BarContainer	20	BarContainer object of matplotlib.cont...
df	DataFrame	(3, 2)	Column names: bools, ints
filename	str	1	C:\ProgramData\Anaconda3\lib\site-pack...
list_test	list	2	[Dataframe, Numpy array]
nrows	int	1	344
r	float64	1	7.611082589334796
radii	float64	(20,)	Min: 0.4983036630525687 Max: 9.856848974942551
region	tuple	2	(slice, slice)
rgb	float64	(45, 45, 4)	Min: 0.0 Max: 1.0
series	Series	(1,)	Series object of pandas.core.series mo...
test_none	NoneType	1	NoneType object of builtin module

The Python console shows the following output:

```
...: # in the rgb colors of the shaded surface calculated from shade ...
...: rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_mode='soft')
...: surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, facecolors=rgb,
...: linewidth=0, antialiased=False, shade=False)
...:
...: plt.show()
```

The console also displays a 3D surface plot and a corresponding polar plot. The 3D plot shows a surface with a color gradient from blue to red. The polar plot shows a circular distribution of data points with a color gradient from blue to red.

Execución desde o entorno SPYDER

- Desde o intérprete executar (como en Linux):

```
>>> run nome.py
```

- Pulsar o botón run na interface
- En ambos casos, necesítase arrancar o entorno para executar os programas. As opcións das primeiras transparencias permiten executar os programas sen entrar no entorno SPYDER.

Mensaxes de erro

- Ó executar un programa o resultado pode non ser correcto por:
 - **Erros na sintaxe de Python:** son máis ou menos doados de correxir a partires das trazas de rastreo que proporciona Python (liña e arquivo nos que se produce o erro). Máis información en: <http://python.org.ar/MensajesExcepcionales>
 - **Erros de execución:** o programa remata sen erros pero o resultado non é correcto. Utilizar o depurador.
 - Erros causados por excepcións non capturadas.

Entrada/saída de datos no programa

- Entrada de datos desde o teclado:
 - **input("texto")**: detén o programa, amosa na pantalla o texto que lle pasemos a función e agarda a que se introduza algo por teclado. A función devolve o que se introduce por teclado como unha cadea de caracteres que coas función **int()** ou **float()** podemos converter a números.
- Saída de datos por pantalla:
 - **print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)**: visualiza datos na consola. O argumento **objects** son os obxectos a visualizar, **sep** é o separador a utilizar entre os obxectos (espacio por defecto), **end** indica o que facer cando se executa **print()** (por defecto cambia de liña '\n'), se **flush** é True forza a que se visualice instantaneamente e **file** é un fluxo que por defecto representa a consola.
 - **print()**: cambia de liña.

Saída de datos con formato

- Saída de datos con formato por pantalla:
 - `print(cadea % (var1, var2, var3, ...))`: onde `cadea` é unha cadea de caracteres con formatos intercalados, tantos como número de variables poñamos.
- Exemplos:
 - `print('Visualiza enteiros x=%d e y=%d' % (34, 40)) #`
devolve Visualiza enteiros x=34 e y=40
 - `print("Visualiza reais x=%7.2f e y=%.3f" % (45.4321, 40.2)) #` Visualiza reais x= 45.43 e y=40.200
 - `print('Hola %s' % ('Python')) #` devolve Hola Python
- Formatos: `%d` (enteiro), `%f` (real), `%e` (real en formato exponencial), `%g` (real, elixe entre `%f` e `%e`), `%s` (cadea de caracteres), `%c` (caracter).
- Formatos: `%m.nf` (`m` significa o ancho do campo e `n` o número de decimais).

Entrada/saída: arquivos de texto

- Pasos a realizar: abrir arquivo, ler/escribir, cerrar arquivo.
- Abrir arquivo: **f=open(nome_arquivo, modo)**
 - nome_arquivo é a ruta do arquivo a ler/escribir.
 - Modo especifica o uso que se vai facer do arquivo: **'r'** (so para lectura), **'w'** (so para escribir desde o principio do arquivo), **'a'** (para escribir ó final dun arquivo) e **'r+'** (para ler e escribir no arquivo).
 - f é o obxecto de tipo **file** devolto por **fopen**.
- Pechar arquivo: **f.close()**
- Métodos do obxecto file:
 - **f.read(tamanho)**: devolve unha cadea de caracteres cos tamaño bytes do arquivo f. Se non especificas tamaño le todo o arquivo.
 - **f.readline()**: devolve unha liña do arquivo (ate que atope o carácter fin de liña (**'\n'**)).

Entrada/saída: arquivos de texto

- Métodos do obxecto **file**:
 - **f.write(cadea)**: escribe unha cadea de caracteres no arquivo.
 - **f.writelines(secuencia)**: escribe unha secuencia de cadeas de caracteres no arquivo (cada elemento da secuencia nunha liña).
 - **f.tell()**: devolve un enteiro coa posición actual de lectura/escritura no arquivo.
 - **f.seek(bytes)**: move a posición actual de lectura/escritura a bytes desde o inicio do arquivo.