Taller de Python

Primer encuentro: Lo básico

FIFA

Federación Interestudiantil de Física Argentina

17 de abril de 2018

¿Qué es eso de Python? ¿Con qué se come?

Python es un *lenguaje de programación*: una forma versátil de darle instrucciones a la computadora para que ella haga las cosas que nosotros queremos.

Instalación de Python

- Si tenés Linux, ya tenés instalado Python ;)
  Pero posiblemente quieras instalar el paquete de Anaconda que viene con muchos chiches.
- Si tenés Windows, no tenés Python por defecto pero lo podés descargar de `https://www.anaconda.com/download/`

Usando Python

Para este taller, vamos a usar el IDE (Interactive Development Environment) llamado Spyder, un programa que nos ejecuta la consola de Python y nos permite escribir archivos .py desde el mismo lugar, con coloreado de palabras especiales y otros chiches. Para los que lo conocen, es parecido al entorno de MATLAB.

¿Pero qué es hablar con la computadora?

La computadora ejecuta programas, que no son más que recetas:

1. Moje el cabello,
2. Coloque champú,
3. Masajee suavemente y deje actuar por 2 min.,
4. Enjuague, y
5. Repita el procedimiento (desde 1.-).

Podemos ver que los pasos de toda receta sólo pueden hacer dos cosas:

- Transforman datos (o estados)
- Cambian el flujo de las operaciones

1. Moje el cabello.
2. Coloque champú,
3. Masajee suavemente y deje actuar por 2 min.,
4. Enjuague, y
5. Repita el procedimiento (desde 1.-).

Podemos ver que los pasos de toda receta sólo pueden hacer dos cosas:

- Transforman datos (o estados)
- Cambian el flujo de las operaciones

1. Moje el cabello.
2. Coloque champú,
3. Masajee suavemente y deje actuar por 2 min.,
4. Enjuague, y
5. Repita el procedimiento (desde 1.-).

Nuestras herramientas

Utilizaremos Spyder como entorno para trabajar en Python. En las compus del laboratorio está disponible. Si trajiste tu compu, andá instalándotelo si no lo tenés.

Como guía de trabajo, utilizaremos la disponible en *http://goo.gl/B2q73R*

Empecemos con lo básico: las palabras

Como todo lenguaje, Python tiene un vocabulario de 31 palabras *claves*

`and`	`del`	`from`	`not`	`while`
`as`	`elif`	`global`	`or`	`with`
`assert`	`else`	`if`	`pass`	`yield`
`break`	`except`	`import`	`print`	
`class`	`exec`	`in`	`raise`	
`continue`	`finally`	`is`	`return`	
`def`	`for`	`lambda`	`try`	

Con esto se puede hablar y escribir Python

¿Qué es realmente un dato?

Los datos representan valores o cantidades de la vida real, como ser cantidad de manzanas que llevo en un canasto, o cantidad de monedas que puedo gastar al comprar un caramelo.

Un dato tiene un *valor* numérico (binario), ya que la computadora debe guardarlo de alguna forma, pero si le defino un *tipo* también se que es realmente y que representa de la vida real (un número, una palabra, etc).

¿Y qué tipo de datos puedo usar?

Los tipos de datos básicos son

- Valores lógicos de verdad o valores *booleanos* (`False` y `True`)
- Enteros (1, 2, 5443, etc)
- Reales con punto decimal flotante (o coma flotante) (1.2, 5.61×10^2, etc)
- Cadenas (o *strings*) de caracteres de texto (u'Hola Mundo',u"Ñoño"',u'# Números' etc). Son un tipo especial de *lista*, con métodos especiales.
- Listas de todos los anteriores sin necesidad de ser homogéneos ([1,2,3], ['H','o','l','a'], [True,False,0,1])

¿Y qué tipo de datos puedo usar?

Los tipos de datos básicos son

- Valores lógicos de verdad o valores *booleanos* (`False` y `True`)
- Enteros (1, 2, 5443, etc)
- Reales con punto decimal flotante (o coma flotante) (1.2, 5.61×10^2, etc)
- Cadenas (o *strings*) de caracteres de texto (u'Hola Mundo',u"Ñoño"',u'# Números' etc). Son un tipo especial de *lista*, con métodos especiales.
- Listas de todos los anteriores sin necesidad de ser homogéneos ([1,2,3], ['H','o','l','a'], [True,False,0,1])

¿Y qué tipo de datos puedo usar?

Los tipos de datos básicos son

- Valores lógicos de verdad o valores *booleanos* (`False` y `True`)
- Enteros (1, 2, 5443, etc)
- Reales con punto decimal flotante (o coma flotante) (1.2, 5.61×10^2, etc)
- Cadenas (o *strings*) de caracteres de texto (u'Hola Mundo',u"Ñoño"',u'# Números' etc). Son un tipo especial de *lista*, con métodos especiales.
- Listas de todos los anteriores sin necesidad de ser homogéneos ([1,2,3], ['H','o','l','a'], [True,False,0,1])

¿Y qué tipo de datos puedo usar?

Los tipos de datos básicos son

- Valores lógicos de verdad o valores *booleanos* (`False` y `True`)
- Enteros (1, 2, 5443, etc)
- Reales con punto decimal flotante (o coma flotante) (1.2, 5.61×10^2, etc)
- Cadenas (o *strings*) de caracteres de texto (u'Hola Mundo',u"Ñoño"',u'# Números' etc). Son un tipo especial de *lista*, con métodos especiales.
- Listas de todos los anteriores sin necesidad de ser homogéneos ([1,2,3], ['H','o','l','a'], [True,False,0,1])

¿Y qué tipo de datos puedo usar?

Los tipos de datos básicos son

- Valores lógicos de verdad o valores *booleanos* (`False` y `True`)
- Enteros (1, 2, 5443, etc)
- Reales con punto decimal flotante (o coma flotante) (1.2, 5.61×10^2, etc)
- Cadenas (o *strings*) de caracteres de texto (u'Hola Mundo',u"Ñoño"',u'# Números' etc). Son un tipo especial de *lista*, con métodos especiales.
- Listas de todos los anteriores sin necesidad de ser homogéneos ([1,2,3], ['H','o','l','a'], [True,False,0,1])

Ahora necesitamos las *variables*. Escriban esto en la consola de Python

```
>>> a = 5
>>> type(a)
<class 'int'>
```

Hicimos un entero, prueben con `True`, '5', 1.2 y [2, 3, 4].
Por ejemplo:

```
>>> a = '5'
>>> type(a)
```

Ahora un poco de control al asunto

Ejecuten el siguiente comando

```
>> print('Hola mundo')
Hola mundo
```

y ahora quiero repetirlo **10** veces. ¿Cómo lo hago?

- Método mecánico

```
print('Hola mundo')
print('Hola mundo')
...
print('Hola mundo')
```

- Que la computadora sepa que tiene que repetir 10 veces

Ahora un poco de control al asunto

Ejecuten el siguiente comando

```
>> print('Hola mundo')
Hola mundo
```

y ahora quiero repetirlo **10** veces. ¿Cómo lo hago?

- Método mecánico

```
print('Hola mundo')
print('Hola mundo')
...
print('Hola mundo')
```

- Que la computadora sepa que tiene que repetir 10 veces

El entorno *for*

¿Cómo puede saber la computadora eso? Para eso existen bucles (loops en inglés).

```
>>> for i in range(10):
...     print('Hola mundo')
...
Hola mundo
Hola mundo
...
Hola mundo
```

Ahora veamos qué es cada cosa...

Nota: **Acuérdense de revisar sintaxis y tabulado**

El entorno *for*

```
>>> range(10)
[0,1,2,3,4,5,6,7,8,9]
```

Genera una lista del 0 a 9 (uno menos que el valor que ingresamos). **Tiene 10 elementos**.

Nota: Prestar atención a que el primer elemento de las listas es el 0. Las listas tienen *desde 0 hasta n-1 elementos*.

El entorno *for*

Entonces con

```
>>> for i in range(10):
...     print('Hola mundo')
...
```

la computadora sabe *literalmente* lo que dice, en inglés: por cada elemento `i` de la lista `range(10)`, haz `print('Hola mundo')`.

El entorno *for*

Entonces con

```
>>> for i in range(10):
...     print('Hola mundo')
...
```

la computadora sabe *literalmente* lo que dice, en inglés: por cada elemento `i` de la lista `range(10)`, haz `print('Hola mundo')`.

Es *exactamente* lo que queríamos que la computadora hiciese

El entorno *for*

Otro ejemplo, más ilustrativo de "recorrer una lista"

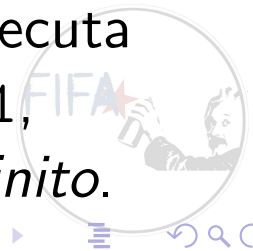
```
>>> for i in range(10):
...     print(i)
...
0
1
2
...
9
```

El entorno *while*

Además del `for`, existe otra estructura de bucle

```
>>> i = 0
>>> while i < 10:
...     print(i)
...     i = i+1       -->  también pueden escribir
...                        i += 1 que es lo mismo
0
1
...
9
```

El bucle al entrar verifica que `i < 10` sea verdadero y luego ejecuta lo que viene abajo. Si no existiese el último comando `i = i+1`, nunca cambiaría el contador y nunca terminaría. Un bucle *infinito*.



Pongamos condiciones a este programa

¿Qué pasa si tengo esto:

```
>>> for i in range(10):
...     print(i)
...
0
1
2
...
9
```

pero quiero que imprima solamente los números pares entre 3 y 8 (inclusive), sin cambiar la lista que se "recorre"? (en general, no vas a poder hacer esto o no querés).

El entorno *if*

Para lo anterior tengo la siguiente estructura:

```
>>> for i in range(10):
...     if i <= 8 and i >= 3:
...         if i % 2 == 0:
...             print(i)
...
```

El comando `if` ejecuta lo que viene a continuación sólo si la condición es verdadera (en este caso que `i` sea mayor que 3 y menor que 8). Las condiciones verdaderas dan valores booleanos `True`.
Vean que puedo tener `if` dentro de `if`, lo que se llama *anidar*.

Nota: El comando %, llamado *módulo*, da el resto de la división de `i` en 2, que es 0 si `i` es par.

El entorno *if*

Prueben usar

```
>>> i = 3
>>> i < 8
True
>>> i > 3
False
>>> i % 2 == 0
False
```

Para igualar expresiones en el sentido matemático usamos ==.

El entorno *if*

Resumiendo, el `if`

```
>>> if CONDICION:
...     ejecuto si es verdadero
```

El entorno *if*

Resumiendo, el `if`

```
>>> if CONDICION:
...     ejecuto si es verdadero
```

¿Que pasa si quiero ejecutar algo si es falsa la condición?

El entorno *if*

Agrego un `else`

```
>>> a = 3
>>> if a < 5:
...     print(True)
... else:
...     print(False)
...
```

El entorno *if*

Si no es verdadera la condición inicial, podemos preguntarnos si hay una condición que si sea verdadera, como en el caso anterior

```
>>> a = 3
>>> if a < 5:
...     print(True)
... elif a == 5:
...     print('Iguales')
... else:
...     print(False)
...
```

Primero verifica la primera, después verifica la segunda condición y si ninguna es verdadera ejecuta lo que está dentro de `else`.

Reutilizando la receta

Imaginate que tenés que ejecutar una operación de forma seguida pero no de forma regular, como por ejemplo

```
>>> a = 2
>>> b = 5
>>> c = 3
>>> d = a + b + c
>>> d
10
```

Reutilizando la receta

Algo tan simple como eso lo queremos hacer modular, queremos una estructura que nos de posibilidad de sumar 3 números en cualquier lugar.

Reutilizando la receta

Algo tan simple como eso lo queremos hacer modular, queremos una estructura que nos de posibilidad de sumar 3 números en cualquier lugar.

¿Cómo lo hacemos?

Reutilizando la receta

Algo tan simple como eso lo queremos hacer modular, queremos una estructura que nos de posibilidad de sumar 3 números en cualquier lugar.

¿Cómo lo hacemos?

Funciones

Funciones

```
>>> def Suma(a,b,c):
...     d = a + b + c
...     return d
>>> Suma(2,5,3)
10
```

Prueben transformar en funciones todo lo que escribieron hasta ahora.

`a,b,c` son *argumentos* de la función Suma y con `return` la función devuelve un resultado, como una función matemática.

Funciones que nos resuelven todo

¿Se acuerdan de `range(10)`? Bueno, es una función de una biblioteca o *biblioteca* básica de Python.
Las funciones básicas más usadas

`abs()`	`bin()`	`bool()`	`chr()`
`divmod()`	`float()`	`format()`	`help()`
`input()`	`open()`	`print()`	`len()`
`list()`	`map()`	`max()`	`min()`
`range()`	`type()`		

De las funciones básicas, la más importante para el recién empezado es la función `help()` a la cual le podemos pasar el nombre de cualquier función e imprimirá la ayuda escrita previamente. Por ejemplo, escriban `help(list)` (pueden salir presionando *q*)

Más funciones que nos resuelven todo

En caso de querer usar funciones matemáticas necesitamos usar

```
>>> import math
>>> math.sin(math.pi)
-1
```

Con `import` le decimos al intérprete de Python que traiga el paquete math y ahí vos lo podés usar.
Como ya vimos antes, los *paquetes* son programas y utilidades organizadas para el uso posterior, en particular los programas están organizados en funciones ya que son *el método* usado para reutilizar programas.

Más funciones que nos resuelven todo

Otro ejemplo puede ser el paquete `os`, que son funciones de sistema operativo

```
>>> import os
>>> os.urandom(10)
```

que nos da una cadena aleatoria de 10 bytes.
Hasta ahora hablamos de paquetes básicos de la instalación. En próximos encuentros hablaremos profundamente de las bibliotecas científicas NUMPY, SCIPY y MATPLOTLIB, que pueden ir bajando e instalado.

Gráficos

Trabajaremos con una de estas para graficar (MATPLOTLIB) y con otra para trabajar numéricamente (NUMPY)

```
>>> from matplotlib import pyplot as plt
>>> import numpy as np
```

Probaremos graficar una función elemental.

Inventemos un dominio y una función imagen. Son supuestas mediciones así que aportemos ruido.

```
>>> x = np.linspace(-5, 5)
>>> y = x**2 -3

>>> ruido = np.random.rand(len(y))*0.8
>>> y = y+ruido
```

Ahora pidamos que grafique.

```
>>> plt.scatter(x, y)
>>> plt.xlabel('Variable independiente')
>>> plt.ylabel('x^2')
>>> plt.grid()

>>> plt.show() #(que nos muestre el gráfico)
```

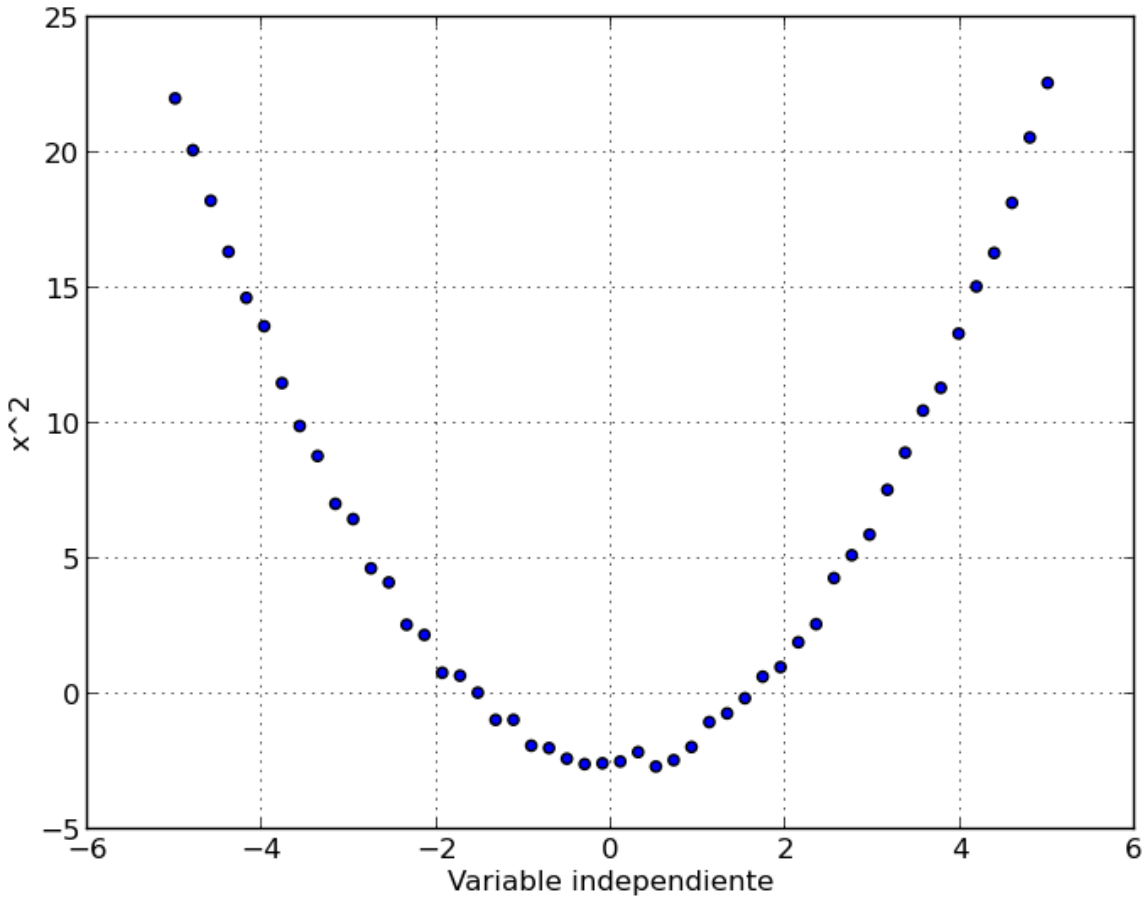


Figura: Gráfico de $f(x) = x^2 - 3$

Para seguir profundizando

Con esto vimos lo básico de programación en Python.
Para seguir buscando tenemos
`http://python.org.ar/`
que tiene muchas páginas y libros para buscar. También recomendamos el tutorial en
`http://www.learnpython.org` y el libro, con muchos ejemplos y exigiendo nada al lector:
Lutz, M (2008). *Learning Python*. 3era Ed. O'Really

Para la próxima

- Ajustes lineales
- Interpolaciones
- Derivación numérica
- Integración numérica
- Estadística básica

¡Gracias por venir!