

Un poco más de Python

ExactasPrograma

Facultad de Ciencias Exactas y Naturales, UBA

Verano 2020

Asignación

```
Anastasio = 4  
Pedrito = 8  
Laura = 5  
Micaela = 10
```

Asignación

En `Python` podemos darle nombre a las *cosas* y asociarles un valor. Esto se llama **asignar** un valor a una **variable**.

`Anastasio` es la **variable** y 4 es el **valor**.

Estado de un programa

El estado de un programa en un momento de su ejecución está definido por el valor de **todas sus variables** en ese momento. Cuando se analiza **qué** es lo que hace el programa, nos interesa ver y entender **cómo** cambian los valores de las variables.

¿Cómo se ve la ejecución de un programa?

Python 3.6

```
1 Anastasio = 4  
→ 2 Pedrito = 8  
→ 3 Laura = 5  
4 Micaela = 10
```

[Edit this code](#)

Frames

Global frame

Anastasio	4
Pedrito	8

Step 3 of 4

- Este sitio permite ejecutar **paso a paso** nuestro programa.
- Nos permite ver el resultado de cada instrucción (**estado** del programa).

Listas

- En Python existen las **listas**, que sirven para almacenar valores:

```
Anastasio = []
Pedrito = []
Laura = []
Micaela = []
```

```
Anastasio.append(4)
Pedrito.append(8)
Laura.append(5)
Micaela.append(10)
```

```
Anastasio.append(6)
Pedrito.append(9)
Laura.append(6)
Micaela.append(13)
```

- ¿Qué pasa si ejecuto la línea `print("Anastasio:", Anastasio)`?

La salida me muestra: `Anastasio: [4, 6]`

Es una manera *linda* de ver el contenido de la lista... pero hay algo más ahí, ¿no?

Se puede definir una lista (por extensión) como:

```
milistita = [2, -1, 4, -2, 8, 17]
```

Cadenas de caracteres

Definición

Es una secuencia de caracteres definida por medio de comillas, es parecida a una lista, pero no es igual (es un tipo *immutable*):

```
'Hola, trencito'
```

Las operaciones básicas (algunas también funcionan con listas) son:

- **+**: concatenación. `'Hola' + ', trencito'` da `'Hola, trencito'`.
- **int**: convierte una cadena a número entero. `int('33')` da 33.
- **float**: convierte una cadena a número con coma. `float('4.5')` da 4.5.
- Al igual que con cualquier lista, dos de las funciones más usadas son:
 - **[]**: para acceder a los contenidos de posiciones individuales dentro de una cadena. Por ejemplo, `'Hola'[3]` da `'a'`.
 - **len**: devuelve la longitud de la cadena de caracteres. `len('abc')` devuelve 3.
- **lower**: devuelve la misma cadena pero en minúsculas. `'Hola'.lower()` da `'hola'`.
- **upper**: similar a la anterior, pero pasa a mayúsculas. `'Hola'.upper()` da `'HOLA'`.

Funciones en `Python`

- Son una construcción que permite *encerrar* un *pedacito* de programa.
- Así como `append`, hay muchísimas funciones que se pueden utilizar y aprovechar.
- Permiten definir cierto comportamiento interesante y no tener que volverlo a programar cada vez.
- Los lenguajes de programación tienen un mecanismo para definir funciones.
- Los valores que recibe una función se llamas **parámetros** o **argumentos**.

Tabulación

`Python` *sabe* donde termina la definición de una función por la tabulación: las instrucciones que componen la función están al menos 4 espacios hacia la derecha.

Tabulación, el retorno

Es **importante** usar una cantidad de espacios o tabulación, pero no mezclar, sino empiezan a aparecer errores muy raros de `Python`. **¡Sean prolijos!**

Ciclos

- El **while** permite repetir una serie de instrucciones **mientras se cumpla** una condición.
- Si, desde el principio, sabemos que el rango del ciclo es **fijo**, se puede usar **for**.
- Definamos la función `suma_elem`, que suma todos los elementos de una lista usando **for** y **while**:

Con **while**:

```
def suma_elem(listita):  
    suma = 0  
    i = 0  
    while i < len(listita):  
        suma = suma +  
        listita[i]  
        i = i + 1  
    return suma
```

Con **for**:

```
def suma_elem(listita):  
    suma = 0  
    for i in range(0, len(listita), 1):  
        suma = suma + listita[i]  
    return suma
```

- **range**(inf, sup, paso): devuelve una estructura que toma los números desde inf hasta sup de a paso. Si no se los escribe, inf vale 0 y paso vale 1.

Otra estructura de control: `if`

- Permite ejecutar una serie de instrucciones si se cumple cierta condición.
- Supongamos que queremos usar la función `proc_jugadas` cuando la lista de jugadas no esté vacía o si lo estuviera, que el resultado fuera `-1`:

```
if Anastasio!=[]:  
    suma_Anastasio = proc_jugadas(Anastasio)  
else:  
    suma_Anastasio = -1
```

```
if Pedrito!=[]:  
    suma_Pedrito = proc_jugadas(Pedrito)  
else:  
    suma_Pedrito = -1
```

```
if Laura!=[]:  
    suma_Laura = proc_jugadas(Laura)  
else:  
    suma_Laura = -1
```

```
if Micaela!=[]:  
    (...)
```

- Los dos puntos (`:`) son obligatorios, ¡No olvidarse!

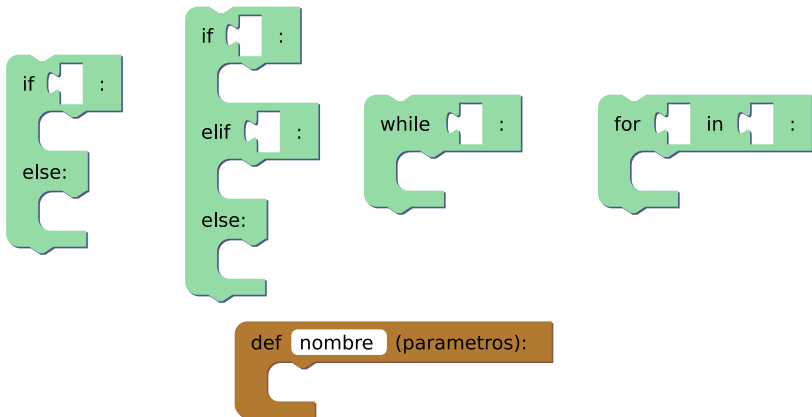
Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
if a>x and a<y:
    c = x*x+y*y
else:
    c = 2*x*y
```

```
if a*a>x or a<y*y:
    c = a*a-y*y
else:
    c = x*y/2
```

Resumen de estructuras



Resolución de `cant_e`

```
def cant_e ( palabra ):  
    i = 0  
    count = 0  
    while i < len(palabra) :  
        if palabra[i] == "e" :  
            count = count + 1  
        i = i + 1  
    return( valor a retornar )
```

Módulos para usar otras funciones

- Si bien `Python` tiene muchas funciones que se pueden usar *directamente*, hay muchas otras que están disponibles como **módulos**.
- Un **módulo** es una colección de funciones que alguien (o una comunidad) desarrollaron y empaquetaron para que estén disponibles para todo el mundo.
- Para que las funciones estén disponibles para ser utilizadas en mi programa, tengo que usar la instrucción `import`.

Módulos para usar otras funciones

- Si quiero generar números aleatorios, que están en el módulo `random`, tengo que escribir:

```
import random
random.seed(...COMPLETEN CON UN NUMERO...)
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
```

- ¿Cómo sé que funciones o módulos hay? **!!!Google!!!**

Existe vida más allá del `pythontutor`: `spyder`

- open source
- cross-platform
- integrated development environment (IDE)
- incluye un editor de texto que remarca las palabras clave del lenguaje
- tiene soporte para distintas versiones de `Python`
- permite escribir programas y probarlos de manera muy sencilla
- En las máquinas de los laboratorios, ya está instalado y listo para usarse (tipear `spyder3` o `spyder` como comando).
- Si tienen Linux con Debian o Ubuntu: `sudo apt-get install spyder3`.
- Para aquellos que tienen máquina Windows:
`https://www.anaconda.com/distribution` y bajar `Python 3.7 version` para Windows (64 bits para máquinas nuevas, 32 bits si tienes una medio viejita).



Editor - /home/mlopez/.config/spyder-py3/mi_prim... Variable explorer

mi_primer_script.py

```
1 #-*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8
9 Anastasio = []
10 Pedrito = []
11 Laura = []
12 Micaela = []
13
14 Anastasio.append(4)
15 Pedrito.append(8)
16 Laura.append(5)
17 Micaela.append(10)
18
19 Anastasio.append(6)
20 Pedrito.append(9)
21 Laura.append(6)
22 Micaela.append(13)
23
24 print("Anastasio: ", Anastasio)
```

Name	Type	Size	Value
Anastasio	list	2	[4, 6]
Laura	list	2	[5, 6]
Micaela	list	2	[10, 13]
Pedrito	list	2	[8, 9]

Variable explorer

File explorer

Help

IPython console

Console 1/A

```
In [3]: runfile('/home/mlopez/.config/spyder-py3/
mi_primer_script.py', wdir='/home/mlopez/.config/spyder-py3')
Anastasio: [4, 6]
```

```
In [4]:
```

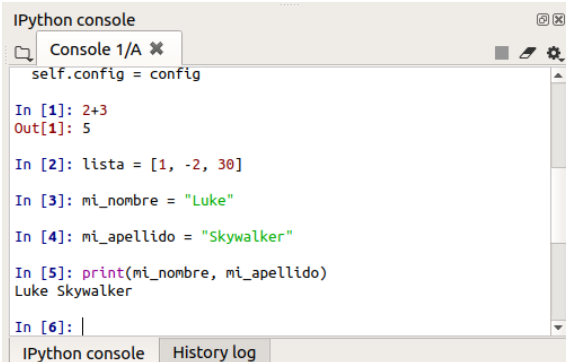
IPython console

History log

¿Cómo usamos cada parte de Spyder?

• Consola

- para hacer pruebas con comandos
- explorar valores de variables
- *jugar en la consola*



```
IPython console
Console 1/A ✕
self.config = config

In [1]: 2+3
Out[1]: 5

In [2]: lista = [1, -2, 30]

In [3]: mi_nombre = "Luke"

In [4]: mi_apellido = "Skywalker"

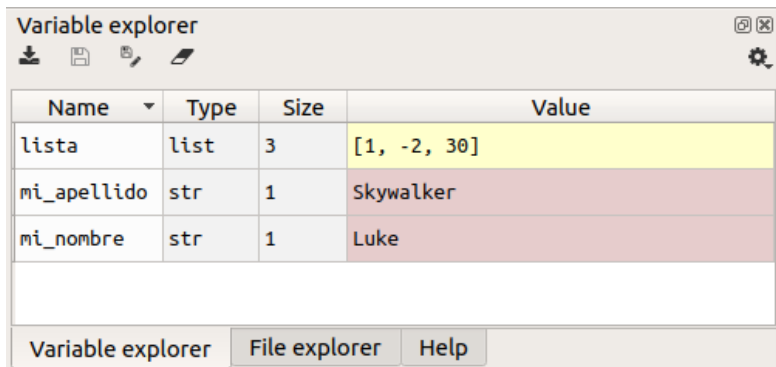
In [5]: print(mi_nombre, mi_apellido)
Luke Skywalker

In [6]: |
```

IPython console History log

¿Cómo usamos cada parte de Spyder?

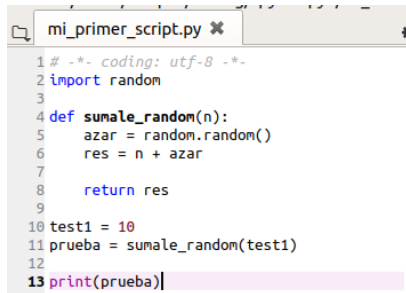
- Explorador de variables: donde podemos ver las variables que existen (similar al *global frame* de Python Tutor)



Name ▾	Type	Size	Value
lista	list	3	[1, -2, 30]
mi_apellido	str	1	Skywalker
mi_nombre	str	1	Luke

¿Cómo usamos cada parte de Spyder?

- Script, donde escribimos nuestro programa que guardaremos para el futuro.
Sugerencia de organización de los archivos:
 - arriba de todo escribimos los **imports** que necesitamos
 - después vienen las funciones
 - finalmente código que usa las funciones y que nos sirve para tener *modelos construidos*



```

1 # -*- coding: utf-8 -*-
2 import random
3
4 def sumale_random(n):
5     azar = random.random()
6     res = n + azar
7
8     return res
9
10 test1 = 10
11 prueba = sumale_random(test1)
12
13 print(prueba)
  
```

- Para que python lo ejecute hay que darle al botón de *Run all*



```

In [6]: runfile('/home/mlopez/.config/spyder-py3/
mi_primer_script.py', wdir='/home/mlopez/.config/spyder-py3')
10.637429270389042
  
```

¿Cómo usamos cada parte de Spyder?

- Una vez que hicimos *Run all* de nuestro script podemos probar las funciones en la consola y ver las variables.

```
In [6]: runfile('/home/mlopez/.config/spyder-py3/  
mi_primer_script.py', wdir='/home/mlopez/.config/spyder-py3')  
10.637429270389042
```

```
In [7]: a = sumale_random(15)
```

```
In [8]: print(a)  
15.864804305756609
```

- Cada vez que hacemos *Run all* en el “mundo python” se crean las funciones y variables de nuestro script. Para estar seguros de que no nos quedan variables *viejas* o funciones en versiones que no funcionaban podemos reiniciarlo desde el menú `Consoles -> Restart kernel`.

FIN DE LAS PRESENTACIONES

Nano Jack: como el Blackjack, pero con problemitas



- Es uno de los juegos que habitualmente se encuentran en los casinos.
- Se juega entre varios jugadores (más de dos).
- Se usan las cartas francesas (las de poker).
- Cada jugador pide cartas tratando de que sus valores *sumen* 21.
- Si te pasas de 21, perdiste.

Nano Jack

Vamos a tomar el *espíritu* del Blackjack para armar un juego que podamos implementar con un programa, así que las reglas van a estar relajadas (ya las vamos a ir viendo).

Empecemos con cartas de poker



- Cada mazo de cartas tiene cuatro palos.
- Dos palos son rojos, dos negros.
- Las cartas van del 1 al 10 y tres figuras: J, Q, K.
- A las figuras se les asigna valores: J vale 11, Q vale 12 y K vale 13.
- Las cartas se entregan de a una
- Cada jugador recibe cartas hasta que gana o pierde

Pensemos entre todos

Nuestra tarea será hacer un programa de computadora que *simule* varios jugadores en una partida de `Nano Jack`. ¿Cómo encaramos esto?

Modelando el juego

- Cuando se enfrenta una situación así, lo mejor es revisar cómo es el juego y cuáles son sus principales características.
- Si todo fue como lo planeado, tendremos algunos mazos de cartas para jugar un poco entre nosotros.
- Junto con divertirnos y conocernos, tratemos de buscar qué sería lo que un programa tendría que ir haciendo para jugar.

¿Qué esperan?

¡A armar los grupos y a jugar un ratito!

¡A trabajar!

- Con todo lo que vimos, podemos empezar a solucionar el problema planteado en el primer taller.
- Recuerden que los docentes estamos para ayudar, aprovechen a consultar **todo**, no se traben.
- El material de las clases y las actividades del curso van a estar en el campus virtual de la facultad: <https://campus.exactas.uba.ar> en:

Dto de Computacion

|-> 2020

|-> Verano

|-> ExactasPrograma-2020-V

- Si no tiene cuenta, deberían poder acceder directamente anónimamente (como *guest* o invitado).

Importante

Aprender a programar se basa en **equivocarse** y aprender de los errores, ¡No tengan miedo de experimentar y preguntar!

Deben subir la tarea a: <http://bit.do/entregas-v2020> .