

Primeros pasos en Python

ExactasPrograma

Facultad de Ciencias Exactas y Naturales, UBA

Verano 2020

Objetivos

- Buscamos descubrir que la computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Vamos a **amigarnos** con la computadora y empezar a ver que puede ser nuestra aliada aún cuando parezca que nos vamos a dedicar a cosas muy alejadas de la tecnología.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivocóquense y vuelvan a preguntar**. ¡Así funciona esto!

Python

- Open Source
- Cross-platform
- Tiene un fuerte desarrollo en una gran variedad de áreas:
 - Desarrollo de aplicaciones con interface gráfica (jueguitos por ejemplo).
 - Análisis de datos (ahora *Ciencia de Datos*)
 - Simulación numérica.
 - Desarrollo para Web e Internet.

*Es un lenguaje que se usa cada día más.
Aprender `Python` **es** una inversión segura.*

Pero... Python, ¿Estás seguro?

Most in-demand programming languages of 2019

Based on Indeed.com job postings in the USA - Feb 1, 2019

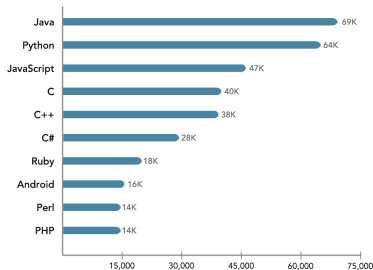
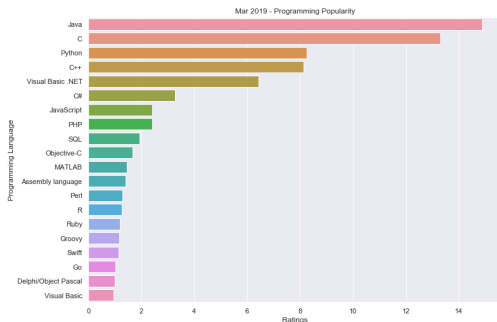


Image Source: CodingNomads



Sí, estamos seguros

No importa el ranking o el sitio que se mire, Python siempre está dentro de los primeros lugares de interés o crecimiento año a año.

Modelando un juego de cartas

- Una de las tareas que se hacen cuando se programa es representar algún aspecto del mundo real, se lo llama **modelar**.
- La computadora es una herramienta **fundamental** para esto.
- ¿Cómo hacemos para *modelar* que al jugador “Anastasio” le tocó un cuatro de picas en un juego de cartas? ¿Qué partes nos podrían interesar de este hecho?
- Vamos a suponer que, para lo que nos interesa **modelar** del juego, solo nos importa el valor.
- ¿Cómo hacemos ahora para decir algo de esto en Python?
- Mi primer programa en Python:
`Anastasio = 4`

Asignación

En Python podemos darle nombre a las cosas y asociarles un valor. A esto se lo llama **asignar** un valor a una **variable**.

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

Más jugadores

- Ahora queremos modelar que al siguiente jugador, “Pedrito”, le tocó un ocho de diamantes... ¿Cómo hacemos?
- Vamos a *alargar* nuestro programa. Vamos a escribir otra **instrucción**:

```
Anastasio = 4  
Pedrito = 8
```

- ¿Y con el resto de los jugadores?

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

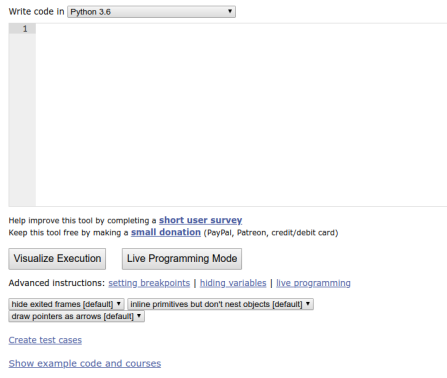
Ejecución de un programa

Aunque parezca sencillo, ya tenemos un programa en `Python` y podríamos *ejecutarlo*. Para hacerlo, necesitamos poder *escribir* el programa y tener instalado un *intérprete* de `Python` para poder ejecutarlo.

Ejecutando mi primer programa en Python

- `http://pythontutor.com`
- Vamos a usar un sitio para hacer nuestros primeros programas en Python.
- Elegir Visualize your code and get live help now de la pantalla principal.
- Del menú al lado de Write code in, asegurarnos que dice Python 3.6 (ahora es el default).
- Escribir el programa:


```
Anastasio = 4
Pedrito = 8
Laura = 5
Micaela = 10
```
- Hacer click en Visualize Execution.



¿Cómo se ve la ejecución?

[How do I use this?](#)

Python 3.6

```

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

[Edit code](#) | [Live programming](#)

→ line that has just executed
 → next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

Step 3 of 4

Keep this tool free for everyone by [making a small donation](#)

Help us improve this tool by clicking below whenever you learn something:

I just cleared up a misunderstanding! I just fixed a bug in my code!

Frames	Objects
Global frame	
Anastasio	4
Pedrito	8

Generate permanent link

Generate shortened link

Click the button above to create a permanent link to your visualization. To report a bug, paste the link along with a brief error description in an email addressed to philip@pgbovine.net

Generate embed code

To embed this visualization in your webpage, click the 'Generate embed code' button above and paste the resulting HTML code into your webpage. Adjust the height and width parameters and change the link to <https://> if needed.

- Este sitio permite ejecutar **paso a paso** nuestro programa.
- Podemos **ver** el resultado de cada instrucción.

¿Cómo hago para modelar una segunda ronda?

- Para modelar que dimos otra carta a cada jugador, podemos definir nuevas variables:

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

```
Anastasio2 = 6  
Pedrito2 = 9  
Laura2 = 6  
Micaela2 = 13
```

- Pero si ahora quisiéramos simular la tercera, ¿Vamos a definir más variables?
¿Con la cuarta ronda?
- ¿Si no sabemos cuántas rondas puede tener el juego?

Así no va...

Necesitamos un mecanismo que nos permita **almacenar** las distintas cartas que le fueron tocando a cada jugador y que permita guardar la cantidad que necesite.

Listas

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

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

En estas líneas, asignamos a cuatro variables el mismo valor: una lista vacía (no contiene ningún elemento).

- Veamos ahora cómo hacer para agregar la carta que recibe cada jugador:

```
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)
```

¿Y si lo ejecuto?

Escribir el programa anterior y ejecutarlo en python tutor.

Python 3.6

```

1 Anastasio=[]
2 Pedrito=[]
3 Laura=[]
4 Micaela=[]
5 Anastasio.append(4)
6 Pedrito.append(8)
7 Laura.append(5)
8 Micaela.append(10)
9 Anastasio.append(6)
10 Pedrito.append(9)
11 Laura.append(6)
12 Micaela.append(13)

```

[Edit code](#) | [Live programming](#)

→ line that has just executed
 → next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First < Back Step 11 of 12 Forward > Last >>

Keep this tool free for everyone by [making a small donation](#)

Help us improve this tool by clicking below whenever you learn something:

I just cleared up a misunderstanding! I just fixed a bug in my code!

Frames

Global frame

Anastasio

Pedrito

Laura

Micaela

Objects

list

0	1
4	6

list

0	1
8	9

list

0	
5	

list

0	
10	

Una función en Python

La función `append` agrega un elemento al final de la lista.

¿Qué vendría a ser un programa?

Vamos a comenzar con esta definición de *entre casa*:

Un programa es una receta de instrucciones que se siguen desde la primera hasta la última y están escritas de una manera particular.

Hay dos puntos importantes:

- **Receta:** da la idea de una serie de instrucciones que deben seguirse al pie de la letra y que **no** son infinitas.
- **Escritas:** la forma de dar las instrucciones sigue una *sintaxis* o forma de escribir pautada. Es lo que se conoce como **lenguaje de programación**.

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa.
- En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).
 - **Definición:** es dónde se indica qué hace la función. En la receta, sería el lugar donde se explica paso a paso qué es batir una clara a punto nieve.
 - **Uso:** Cada vez que se usa o *llama* a la función. En las recetas, sería cada vez que nos indican “batir una clara a punto nieve”, “batir dos claras a punto nieve”, etc.
- Si bien se llama **igual** que el objeto matemático que todos deberíamos conocer, en el universo computacional, forzamos un poco las cosas.
- Así como `append`, hay **muchas** funciones que se pueden utilizar.
- Todos los lenguajes de programación **tienen** un mecanismo para definir funciones.
- Los valores que recibe una función se llaman **parámetros** o **argumentos**:

```
def suma_a_mult_b(a, b, c):
    x = c + a
    y = c * b
    r = x*x - y*y
    return r
res = suma_a_mult_b(2, 4, 7)
print(res)
res = suma_a_mult_b(5, 10, 2)
print(res)
```

Funciones: a probarlo

```
def suma_a_mult_b(a, b, c):  
    x = c + a  
    y = c * b  
    r = x*x - y*y  
    return r  
res = suma_a_mult_b(2, 4, 7)  
print(res)  
res = suma_a_mult_b(5, 10, 2)  
print(res)
```

Tabulación

Python *sabe* donde termina la definición de una función por la **tabulación**. En el caso de la función `suma_a_mult_b` las instrucciones que componen la función están cuatro espacios hacia la izquierda.

Funciones: ¿Si lo ejecuto?

Vamos a escribir algo parecido al programa anterior y a ejecutarlo.

Python 3.6

```

→ 1 def suma_a_mult_b(a, b, c):
2     x = c + a
3     y = c * b
4     r = x*x - y*y
5     return r
→ 6 r = suma_a_mult_b(2, 4, 7)
7 print(r)
8 r = suma_a_mult_b(5, 10, 2)
9 print(res)

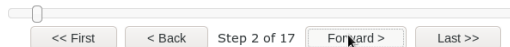
```

[Edit this code](#)

→ line that has just executed

→ next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.



Help improve this tool by clicking whenever you learn something:

I just cleared up a misunderstanding!

I just fixed a bug in my code!

Print output (drag lower right corner to resize)



Frames

Objects

Global frame

suma_a_mult_b

function

suma_a_mult_b(a, b, c)

A esta altura de la ejecución del programa, Python solo *sabe* que existe una función llamada `suma_a_mult_b` que recibe tres argumentos.

¡Sigamos ejecutando!

Python 3.6

```

→ 1 def suma_a_mult_b(a, b, c):
→ 2     x = c + a
  3     y = c * b
  4     r = x*x - y*y
  5     return r
  6 r = suma_a_mult_b(2, 4, 7)
  7 print(r)
  8 r = suma_a_mult_b(5, 10, 2)
  9 print(res)

```

[Edit this code](#)

→ line that has just executed

→ next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

Help improve this tool by clicking whenever you learn something:

Print output (drag lower right corner to resize)

Frames

Objects

Global frame

suma_a_mult_b

function

suma_a_mult_b(a, b, c)

suma_a_mult_b

a

2

b

4

c

7

En este paso, Python necesita obtener el resultado de la función `suma_a_mult_b` con los argumentos 2,4 y 7.

¡Sigamos ejecutando!

Python 3.6

```

1 def suma_a_mult_b(a, b, c):
2     x = c + a
3     y = c * b
4     r = x*x - y*y
5     return r
6 r = suma_a_mult_b(2, 4, 7)
7 print(r)
8 r = suma_a_mult_b(5, 10, 2)
9 print(res)

```

[Edit this code](#)

→ line that has just executed

→ next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First
 < Back
 Step 7 of 17
 Forward >
 Last >>

Help improve this tool by clicking whenever you learn something:

I just cleared up a misunderstanding!

I just fixed a bug in my code!

Print output (drag lower right corner to resize)

Frames

Objects

Global frame

suma_a_mult_b

function

suma_a_mult_b(a, b, c)

suma_a_mult_b

a	2
b	4
c	7
x	9
y	28
r	-703

Acá, Python está a punto de devolver el **resultado** de la ejecución de la función. La instrucción `return` le indica a Python que el valor de la expresión que sigue es el resultado de la función.

Y la ejecución termina así...

Python 3.6

```

1 def suma_a_mult_b(a, b, c):
2     x = c + a
3     y = c * b
4     r = x*x - y*y
5     return r
6 r = suma_a_mult_b(2, 4, 7)
7 print(r)
8 r = suma_a_mult_b(5, 10, 2)
9 print(res)

```

[Edit this code](#)

→ line that has just executed

→ next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.



Help improve this tool by clicking whenever you learn something:

I just cleared up a misunderstanding!

I just fixed a bug in my code!

Print output (drag lower right corner to resize)

-703

Frames

Objects



El valor que *devolvió* la función `suma_a_mult_b` fue asignada a una variable `r`, que si bien se llama igual que la definida dentro de la función... **¡es otra!**

Las variables que se definen dentro de una función, solo **existen** dentro de esa función. A esto se lo conoce como **alcance** de una definición.

Strings: listas raras de caracteres

- Vimos algunos ejemplos de listas con números enteros. Veamos ahora que podemos definir también estructuras con caracteres.
- Para definir un *string* (cadena de caracteres), podemos hacer esto:

```
nombre = "Marianito"
```

Esto define la variable `nombre` que contiene una cadena de caracteres.

- Podemos imprimir el string:

```
print(nombre)
```

- Pero podemos hacerlo mucho más divertido:

```
print("Mi nombre es:", nombre)
```

El resultado de esto es que, en pantalla, vamos a ver el contenido de dos cadenas... ¿Cuáles? ¡Probémoslo!

- Una vez que lo probemos y si no cambiamos nada, lo que salió en pantalla fue: `Mi nombre es: Marianito`. Noten que **print** combinó una cadena de caracteres constante con el contenido de una variable para mostrar todo en pantalla.

Metiéndose con las cadenas

Hay algunas operaciones interesantes para hacer con cadenas (también con listas en general):

- `len()`: es una función que recibe una cadena de caracteres (o una lista) y te devuelve la cantidad de elementos que tiene (viene de “length”, longitud). Por ejemplo la podemos usar `len(lista_de_las_compras)` y nos dirá el largo de la lista que le pasamos.

`len([])` da...¡0!

- `[]`: es un *operador* para acceder a cualquier elemento de una cadena o de una lista:

```
nombre = "Marianito"
print(nombre[0])
print(nombre[5])
```

Hay que tener cuidado con acceder a los elementos fuera de rango, aparecen errores feos.

Metiéndose con las cadenas

Más cosas:

- También se pueden crear listas usando `[]` y dando los elementos:

```
lista_numeros = [1,2,3,10,99]
```

```
lista_caracteres = ['1','2','3','10','99']
```

- `+`: concatenación, toma dos strings (o listas) y los *pega*:

```
apellido = "Quito"
```

```
primer_nombre = "Armando"
```

```
segundo_nombre = "Esteban"
```

```
print("Hola!!, quiero hablar con " + apellido + ", " + primer_nombre  
      + segundo_nombre + apellido + " por favor")
```

¿Funciona bien? ¿Se puede mejorar?

Los strings no son listas, son parecidos

- Si bien los strings son *casí* como listas, son **inmutables**.
- Esto significa que, una vez definidos, no se pueden cambiar:

```
nombre = "Marianito"
nombre[8] = 'a'
```

Esto **no** funciona da un error que dice

`TypeError: 'str' object does not support item assignment.`
Significa que estamos queriendo cambiar algo que **no** se puede cambiar.

- La manera para trabajar esto es usando una lista de verdad:

```
nombre = "Marianito"
milista = list(nombre)
milista[8] = 'a'
print(milista)
print(nombre)
```

```
milista[len(milista)-1] = 'a' # Alternativa para lo mismo
#(esto es un comentario)
```

Ciclo, triciclo, ciclotímico: ciclo que te ciclo y no reciclo

Quiero imprimir en pantalla los números del 1 al 10:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
print(6)
print(7)
print(8)
print(9)
print(10)
```

¡Qué gran programa!

Bueno, bueno, dejen de tirar cosas y abuchear, lo podemos hacer así:

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

Si quisiéramos sumar 1 a todos los elementos de una lista:

```
def cambiar_lista(lis):
    i = 0
    while i < len(lis):
        lis[i] = lis[i] + 1
        i = i + 1
    return lis
```

Escribamos los dos programas en el tutor y veamos cómo funcionan. En el segundo caso, no se olviden de **llamar** a la función con algún parámetro **razonable** para probarla.

Otra estructura de control: `if`

- Permite ejecutar una serie de instrucciones si se cumple cierta condición.
- Supongamos que queremos cambiar un valor dependiendo si vale o no 2:

```
mivalor = 2
if mivalor==2:
    mivalor = 5
else:
    mivalor = 8

print(mivalor)
```

Los dos puntos (:) son obligatorios,

¡No olvidarse!

- Veamos un ejemplo más divertido:

```
def multiploRaro(a):
    if a == 2:
        res = a * a + 1
    elif a == 3:
        res = a + a - 1
    elif a == 7:
        res = a * a + 5
    elif a == 11:
        res = a * (a - 2) + 9
    else:
        res = a * a - 2
    return res
```

Y ahora a probarlo:

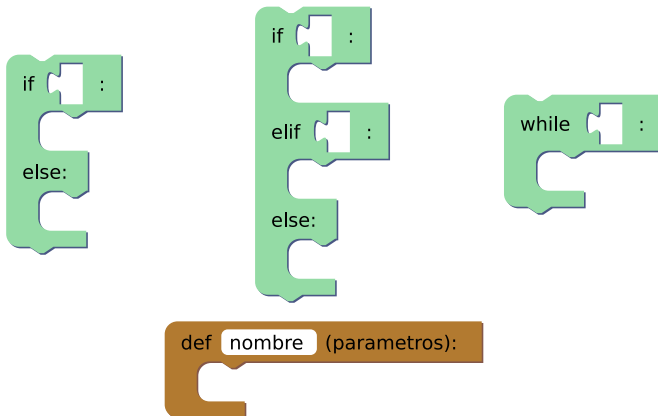
```
prueba=multiploRaro(8)
print(prueba)
```


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
```

Estructuras



Ejercicios

- 1 Defina una función que tome dos cadenas de caracteres como parámetro y devuelva la de mayor longitud. Complete el siguiente programa (la cantidad de guiones no indica la cantidad de caracteres a completar):

```
def mas_larga(l1 , l2):
    if -----:
        res = ---
    else---
        res ---
    return res
```

Probarlo con estos y otros casos:

```
nombre1="Pepe"
nombre2="Chirizo "
res=mas_larga-----
print(res)
```

- 2 Defina una función que recibe una cadena de caracteres y devuelve la cantidad de letras e que contiene.

```
def cant_e ----:
    i=0
    count=0
    while i<----
        if ---[i]=='e' ----
            count=count---
        i----1
    return -----
```

- 3 Defina una función que tome una lista y cambie todas las vocales por -.
- 4 Defina la función `mezclar` que tome dos listas y devuelva una lista que sea el resultado de intercalas elemento a elemento. Por ejemplo: si intercalamos `Pepe` con `Jose` daría `PJeopsee`.

Links útiles

En el Campus de la facultad: `https://campus.exactas.uba.ar` en:

 Dto de Computacion

 |-> 2020

 |-> Verano

 |-> ExactasPrograma-2020-V

van a encontrar el material del curso.

¡Ya hay guías de ejercicios para complementar si necesitan practicar!