# **Introducción a Python**

Parte de este capítulo se basa en tutoriales por Geek Girls Carrots (<http://django.carrots.pl/>).

¡Vamos a escribir algo de código!

## **El prompt de Python**

Para lectores en casa: esta parte está cubierta en el video [Python Basics: Integers, Strings, Lists, Variables and Errors](https://www.youtube.com/watch?v=MO63L4s-20U) (en Inglés).

Para empezar a jugar con Python, tenemos que abrir una *línea de comandos* en nuestra computadora. Deberías saber cómo hacerlo, lo aprendiste en el capítulo de Introducción a la línea de comandos.

Una vez que estés listo, sigue las siguientes instrucciones.

Queremos abrir una consola de Python, así que escribe python en Windows o python3 en Mac OS/Linux y pulsa intro.

$ python3  
Python 3.6.1 (...)  
Type "help", "copyright", "credits" or "license" for more information.  
>>>

## **¡Tu primer comando en Python!**

Después de ejecutar el comando de Python, el cursor cambia a >>>. Para nosotros esto significa que por ahora sólo podemos utilizar comandos en el lenguaje Python. No tienes que escribir el >>> - Python lo hará por ti.

Si deseas salir de la consola de Python en cualquier momento, simplemente escribe exit() o usa el atajo Ctrl + Z para Windows y Ctrl + D para Mac/Linux. Luego no verás más >>>.

Por ahora no queremos salir de la consola de Python. Queremos aprender más sobre ella. Empecemos con algo muy sencillo. Por ejemplo, intenta escribir algo matemático, como 2 + 3 y presiona intro.

>>> 2 + 3  
5

¡Bien! ¿Ves como salió la respuesta? ¡Python sabe matemáticas! Podrías intentar otros comandos como:

* 4 \* 5
* 5 - 1
* 40 / 2

Para realizar cálculos exponenciales, por ejemplo, 2 a la potencia de 3, escribimos:

>>> 2 \*\* 3  
8

Diviértete con esto por un momento y luego vuelve aquí :).

Como puedes ver, Python es una gran calculadora. Si te estás preguntando qué más puede hacer...

## **Cadenas de caracteres**

¿Cuál es tu nombre? Escribe tu nombre de pila entre comillas así:

>>> "Melissa"  
'Melissa'

¡Has creado tu primera cadena! Es una secuencia de caracteres que puede ser procesada por una computadora. Una cadena, "string" en inglés, siempre debe comenzar y terminar con el mismo carácter. Pueden ser comillas simples (') o dobles (") (¡no hay diferencia!) Las comillas le dicen a Python que lo que está dentro de ellas es una cadena.

Las cadenas pueden ser concatenadas. Prueba esto:

>>> "Hi there " + "Melissa"  
'Hi there Melissa'

También puedes multiplicar las cadenas por un número:

>>> "Melissa" \* 3  
'MelissaMelissaMelissa'

Si necesitas poner un apóstrofe dentro de la cadena, tienes dos formas de hacerlo.

Usar comillas dobles:

>>> "Runnin' down the hill"  
"Runnin' down the hill"

o escapar el apóstrofe con una barra invertida (\):

>>> 'Runnin\' down the hill'  
"Runnin' down the hill"

Bien, ¿eh? Para ver tu nombre en letras mayúsculas, simplemente escribe:

>>> "Melissa".upper()  
'MELISSA'

¡Acabas de usar la **función** upper sobre tu cadena! Una función (como upper()) es un conjunto de instrucciones que Python tiene que realizar sobre un objeto determinado ("Melissa") una vez que se llama.

Si quisieras saber el número de letras que contiene tu nombre, ¡también hay una **función** para eso!

>>> len("Melissa")  
7

Te preguntarás por qué a veces se llama a las funciones con un . al final de una cadena (como "Melissa".upper()) y a veces se llama a una función y colocas la cadena entre paréntesis. Bueno, en algunos casos las funciones pertenecen a objetos, como upper(), que sólo puede ser utilizado sobre cadenas (upper() es una función de los objetos string). En este caso, llamamos **método** a esta función. Otra veces, las funciones no pertenecen a ningún objeto específico y pueden ser usados en diferentes objetos, como len(). Esta es la razón de por qué estamos pasando "Melissa" como un parámetro a la función len.

### **Resumen**

Ok, suficiente sobre las cadenas. Hasta ahora has aprendido sobre:

* **la terminal** - teclear comandos (código) dentro de la terminal de Python resulta en respuestas de Python
* **números y strings** - en Python los números son usados para matemáticas y strings para objetos de texto
* **operadores** - como + y \*, combina valores para producir uno nuevo
* **funciones** - como upper() y len(), realizan opciones sobre los objetos.

Estos son los conocimientos básicos que puedes aprender de cualquier lenguaje de programación. ¿Listo para algo un poco más difícil? ¡Apostamos que lo estás!

## **Errores**

Vamos a intentar algo nuevo. ¿Podemos obtener la longitud de un número de la misma manera que pudimos averiguar la longitud de nuestro nombre? Escribe len(304023) y pulsa intro:

>>> len(304023)  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
TypeError: object of type 'int' has no len()

¡Tenemos nuestro primer error! Dice que los objetos de tipo "int" (números enteros) no tienen longitud. ¿Qué podemos hacer ahora? ¿Quizá podamos escribir el número como una cadena? Las cadenas tienen longitud, ¿verdad?

>>> len(str(304023))  
6

¡Funcionó! Hemos utilizado la función str dentro de la función len. str() que convierte todo en cadenas.

* La función str convierte cosas en cadenas, **strings**
* La función int convierte cosas en enteros, **integers**

Importante: podemos convertir números en texto, pero no podemos necesariamente convertir texto en números - ¿qué sería int('hello')?

## 

## **Variables**

Un concepto importante en programación son las variables. Una variable no es más que un nombre para alguno de forma que puedas usarlo más tarde. Los programadores usan estas variables para almacenar datos, hacer su código más legible y para no tener que recordar qué es cada cosa.

Supongamos que queremos crear una nueva variable llamada name:

>>> name = "Melissa"

¿Ves? ¡Es fácil! Es simplemente: name equivale a Melissa.

Como habrás notado, el programa no devuelve nada como lo hacía antes. ¿Cómo sabemos que la variable existe realmente? Simplemente escribe name y pulsa intro:

>>> name  
'Ola'

¡Genial! ¡Tu primera variable :)! Siempre puedes cambiar a lo que se refiere:

>>> name = "Sonja"  
>>> name  
'Sonja'

También puedes usarla dentro de funciones:

>>> len(name)  
5

Increíble, ¿verdad? Por supuesto, las variables pueden ser cualquier cosa, ¡también números! Prueba esto:

>>> a = 4  
>>> b = 6  
>>> a \* b  
24

Pero ¿qué pasa si usamos el nombre equivocado? ¿Puedes adivinar qué pasaría? ¡Vamos a probar!

>>> city = "Tokyo"  
>>> ctiy  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
NameError: name 'ctiy' is not defined

¡Un error! Como puedes ver, Python tiene diferentes tipos de errores y este se llama **NameError**. Python te dará este error si intentas utilizar una variable que no ha sido definida aún. Si más adelante te encuentras con este error, verifica tu código para ver si no has escrito mal una variable.

¡Juega con esto un rato y descubre qué puedes hacer!

## **La función print**

Intenta esto:

>>> name = 'Maria'  
>>> name  
'Maria'  
>>> print(name)  
Maria

Cuando sólo escribes name, el intérprete de Python responde con la *representación* en forma de cadena de la variable 'name', que son las letras M-a-r-i-a, rodeadas de comillas simples ''. Cuando dices print(name), Python va a "imprimir" el contenido de la variable a la pantalla, sin las comillas, que es más claro.

Como veremos después, print() también es útil cuando queremos imprimir cosas desde adentro de las funciones, o bien cuando queremos imprimir cosas en múltiples líneas.

## **Listas**

Además de cadenas y enteros, Python tiene toda clase de tipos de objetos diferentes. Ahora vamos a introducir uno llamado **list**. Las listas son exactamente lo que piensas que son: objetos que son listas de otros objetos :)

Anímate y crea una lista:

>>> []  
[]

Sí, esta lista está vacía. No es muy útil, ¿verdad? Vamos a crear una lista de números de lotería. No queremos repetir todo el tiempo, así que la pondremos también en una variable:

>>> lottery = [3, 42, 12, 19, 30, 59]

Muy bien, ¡tenemos una lista! ¿Qué podemos hacer con ella? Vamos a ver cuántos números de lotería hay en la lista. ¿Tienes alguna idea de qué función deberías usar para eso? ¡Ya lo sabes!

>>> len(lottery)  
6

¡Sí! len() puede darte el número de objetos en una lista. Útil, ¿verdad? Tal vez la ordenemos ahora:

>>> lottery.sort()

No devuelve nada, sólo ha cambiado el orden en que los números aparecen en la lista. Vamos a imprimirla otra vez y ver que ha pasado:

>>> print(lottery)  
[3, 12, 19, 30, 42, 59]

Como puedes ver, los números de tu lista ahora están ordenados de menor a mayor. ¡Felicidades!

¿Te gustaría invertir ese orden? ¡Vamos a hacerlo!

>>> lottery.reverse()  
>>> print(lottery)  
[59, 42, 30, 19, 12, 3]

Fácil, ¿no? Si quieres agregar algo a tu lista, puedes hacerlo escribiendo este comando:

>>> lottery.append(199)  
>>> print(lottery)  
[59, 42, 30, 19, 12, 3, 199]

Si deseas mostrar sólo el primer número, puedes hacerlo mediante el uso de **indexes** (en español, índices). Un índice es el número que te dice dónde en una lista aparece un ítem. Los programadores prefieren comenzar a contar desde 0, por lo tanto el primer objeto en tu lista está en el indice 0, el próximo está en el 1, y así sucesivamente. Intenta esto:

>>> print(lottery[0])  
59  
>>> print(lottery[1])  
42

Como puedes ver, puedes acceder a diferentes objetos en tu lista utilizando el nombre de la lista y el índice del objeto dentro de corchetes.

Para borrar algo de tu lista necesitas usar **indices** como aprendimos anteriormente y el método pop(). Vamos a ver un ejemplo y reforzar lo que aprendimos anteriormente; vamos a borrar el primer número de nuestra lista.

>>> print(lottery)  
[59, 42, 30, 19, 12, 3, 199]  
>>> print(lottery[0])  
59  
>>> lottery.pop(0)  
59  
>>> print(lottery)  
[42, 30, 19, 12, 3, 199]

¡Funcionó de maravilla!

Para diversión adicional, prueba algunos otros índices: 6, 7, 1000, -1, -6 ó -1000. A ver si se puedes predecir el resultado antes de intentar el comando. ¿Tienen sentido los resultados?

Puedes encontrar una lista de todos los métodos disponibles para listas en este capítulo de la documentación de Python: <https://docs.python.org/3/tutorial/datastructures.html>

## **Diccionarios**

Para lectores en casa: esta parte está cubierta en el video [Python Basics: Dictionaries](https://www.youtube.com/watch?v=ZX1CVvZLE6c) (en Inglés).

Un diccionario es similar a una lista, pero accedes a valores usando una clave en vez de un índice. Una clave puede ser cualquier cadena o número. La sintaxis para definir un diccionario vacío es:

>>> {}  
{}

Esto demuestra que acabas de crear un diccionario vacío. ¡Hurra!

Ahora, trata escribiendo el siguiente comando (intenta reemplazando con propia información):

>>> participant = {'name': 'Ola', 'country': 'Poland', 'favorite\_numbers': [7, 42, 92]}

Con este comando, acabas de crear una variable participant con tres pares clave-valor:

* La clave name apunta al valor 'Ola' (un objeto string),
* country apunta a 'Poland' (otro string),
* y favorite\_numbers apunta a [7, 42, 92] (una list con tres números en ella).

Puedes verificar el contenido de claves individuales con esta sintaxis:

>>> print(participant['name'])  
Ola

Mira, es similar a una lista. Pero no necesitas recordar el índice - sólo el nombre.

¿Qué pasa si le pedimos a Python el valor de una clave que no existe? ¿Puedes adivinar? ¡Pruébalo y verás!

>>> participant['age']  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
KeyError: 'age'

¡Mira, otro error! Este es un **KeyError**. Python te ayuda y te dice que la llave 'age' no existe en este diccionario.

¿Cuando deberías usar un diccionario o una lista? Bueno, es un buen punto para reflexionar. Simplemente ten una solución en mente antes de buscar una respuesta en la siguiente línea.

* ¿Sólo necesitas una secuencia ordenada de elementos? Usa una lista.
* ¿Necesitas asociar valores con claves, así puedes buscarlos eficientemente (usando las claves) más adelante? Utiliza un diccionario.

Los diccionarios, como las listas, son *mutables*, lo que quiere decir que pueden ser modificados después de ser creados. Puedes agregar nuevos pares clave/valor a un diccionario luego de crearlo, como:

>>> participant['favorite\_language'] = 'Python'

Como las listas, usando el método len() en los diccionarios devuelve el número de pares clave-valor en el diccionario. Adelante escribe el comando:

>>> len(participant)  
4

Espero tenga sentido hasta ahora. :) ¿Listo para más diversión con los diccionarios? Salta a la siguiente línea para algunas cosas sorprendentes.

Puedes utilizar el método pop() para borrar un elemento en el diccionario. Por ejemplo, si deseas eliminar la entrada correspondiente a la clave 'favorite\_numbers', sólo tienes que escribir el siguiente comando:

>>> participant.pop('favorite\_numbers')  
[7, 42, 92]  
>>> participant  
{'country': 'Poland', 'favorite\_language': 'Python', 'name': 'Ola'}

Como puedes ver en la salida, el par de clave-valor correspondiente a la clave 'favorite\_numbers' ha sido eliminado.

Además de esto, también puedes cambiar un valor asociado a una clave ya creada en el diccionario. Teclea:

>>> participant['country'] = 'Germany'  
>>> participant  
{'country': 'Germany', 'favorite\_language': 'Python', 'name': 'Ola'}

Como puedes ver, el valor de la clave 'country' ha sido modificado de 'Poland' a 'Germany'. :) ¿Emocionante? ¡Hurra! Has aprendido otra cosa asombrosa.

### **Resumen**

¡Genial! Sabes mucho sobre programación ahora. En esta última parte aprendiste sobre:

* **errors** - ahora sabes cómo leer y entender los errores que aparecen si Python no entiende un comando que le has dado
* **variables** - nombres para los objetos que te permiten codificar más fácilmente y hacer el código más legible
* **lists** - listas de objetos almacenados en un orden determinado
* **dictionaries** - objetos almacenados como pares clave-valor

¿Emocionado por la siguiente parte? :)

## **Compara cosas**

Para lectores en casa: esta parte está cubierta en el video [Python Basics: Comparisons](https://www.youtube.com/watch?v=7bzxqIKYgf4) (en Inglés).

Una gran parte de la programación incluye comparar cosas. ¿Qué es lo más fácil para comparar? Números, por supuesto. Vamos a ver cómo funciona:

>>> 5 > 2  
True  
>>> 3 < 1  
False  
>>> 5 > 2 \* 2  
True  
>>> 1 == 1  
True  
>>> 5 != 2  
True

Le dimos a Python algunos números para comparar. Como puedes ver, Python no sólo puede comparar números, sino que también puede comparar resultados de método. Bien, ¿eh?

¿Te preguntas por qué pusimos dos signos igual == al lado del otro para comparar si los números son iguales? Utilizamos un solo = para asignar valores a las variables. Siempre, **siempre** es necesario poner dos == si deseas comprobar que las cosas son iguales entre sí. También podemos afirmar que las cosas no son iguales a otras. Para eso, utilizamos el símbolo !=, como mostramos en el ejemplo anterior.

Da dos tareas más a Python:

>>> 6 >= 12 / 2  
True  
>>> 3 <= 2  
False

> y < son fáciles, pero ¿qué significa > = y < =? Se leen así:

* x > y significa: x es mayor que y
* x < y significa: x es menor que y
* x <= y significa: x es menor o igual que y
* x >= y significa: x es mayor o igual que y

¡Genial! ¿Quieres hacer uno más? Intenta esto:

>>> 6 > 2 and 2 < 3  
True  
>>> 3 > 2 and 2 < 1  
False  
>>> 3 > 2 or 2 < 1  
True

Puedes darle a Python todos los números para comparar que quieras, y siempre te dará una respuesta. Muy inteligente, ¿verdad?

* **and** - si utilizas el operador and, ambas comparaciones deben ser True para que el resultado de todo el comando sea True
* **or** - si utilizas el operador or, sólo una de las comparaciones tiene que ser True para que el resultado de todo el comando sea True

¿Has oído la expresión "comparar manzanas con naranjas"? Vamos a probar el equivalente en Python:

>>> 1 > 'django'  
Traceback (most recent call last):  
 File "<stdin>", line 1, in <module>  
TypeError: unorderable types: int() > str()

Aquí verás que al igual que en la expresión, Python no es capaz de comparar un número (int) y un string (str). En cambio, muestra un **TypeError** y nos dice que los dos tipos no se pueden comparar.

## **Boolean**

Por cierto, acabas de aprender acerca de un nuevo tipo de objeto en Python. Se llama un **Boolean** (o booleano en español), y es probablemente el tipo más simple que existe.

Hay sólo dos objetos booleanos:

* True
* False

Pero para que Python entienda esto, siempre necesitas escribirlos como 'True' (la primera letra en mayúscula, con el resto de las letras en minúscula). **true, TRUE, tRUE no funcionarán -- solo True es correcto.** (Lo mismo aplica para 'False', por supuesto.)

Los valores booleanos pueden ser variables, también. Ve el siguiente ejemplo:

>>> a = True  
>>> a  
True

También puedes hacerlo de esta manera:

>>> a = 2 > 5  
>>> a  
False

Practica y diviértete con los booleanos ejecutando los siguientes comandos:

* True and True
* False and True
* True or 1 == 1
* 1 != 2

¡Felicidades! Los booleanos son una de las funciones más geniales en programación y acabas de aprender cómo usarlos.

# **¡Guárdalo!**

Hasta ahora hemos escrito todo nuestro código Python en el intérprete, lo cual nos limita a ingresar una línea de código a la vez. Normalmente los programas son guardados en archivos y son ejecutados por el **intérprete** o **compilador** de nuestro lenguaje de programación. Hasta ahora, hemos estado corriendo nuestros programas de a una línea por vez en el **intérprete** de Python. Necesitaremos más de una línea de código para las siguientes tareas, entonces necesitaremos hacer rápidamente lo que sigue:

* Salir del intérprete de Python
* Abrir el editor de texto de nuestra elección
* Guardar algo de código en un nuevo archivo de Python
* ¡Ejecutarlo!

Para salir del intérprete de Python que hemos estado usando, simplemente escribe la función exit():

>>> exit()  
$

Esto te llevará de vuelta a la línea de comandos.

Anteriormente, elegimos un editor de código en la sección de **Editor de código**. Tendremos que abrir el editor ahora y escribir algo de código en un archivo nuevo:

print('Hello, Django girls!')

Obviamente, ahora eres una desarrollador Python muy experimentado, así que sientete libre de escribir algo del código que has aprendido hoy.

Ahora tenemos que guardar el archivo y asignarle un nombre descriptivo. Vamos a llamar al archivo **python\_intro.py** y guardarlo en tu escritorio. Podemos nombrar el archivo como queramos, pero la parte importante es asegurarse de que termina en **.py**. La extensión **.py** le dice a nuestro sistema operativo que es un **archivo ejecutable de Python** y que Python puede correrlo.

**Nota** Deberías notar una de las cosas más geniales de los editores de código: ¡los colores! En la consola de Python, todo era del mismo color, ahora deberías ver que la función print es de un color diferente de las cadenas dentro de ella. Esto se denomina "sintaxis resaltada", y es una característica muy útil cuando se programa. El color de las cosas te dará pistas, como cadenas no cerradas o errores tipográficos en un nombre clave (como la función def, que veremos a continuación). Esta es una de las razones por las cuales se usa un editor de código :)

Con el archivo guardado, ¡es hora de ejecutarlo! Utilizando las habilidades que has aprendido en la sección de línea de comandos, utiliza la terminal para **cambiar los directorios** e ir al escritorio.

**Change directory: OS X**

En una Mac, el comando se verá algo como esto:

$ cd ~/Desktop

**Change directory: Linux**

En Linux, va a ser así (la palabra "Desktop" puede estar traducida a tu idioma):

$ cd ~/Desktop

**Change directory: Windows Command Prompt**

Y en Windows Command Prompt, será así:

> cd %HomePath%\Desktop

**Change directory: Windows Powershell**

Y en Windows Powershell, será así:

> cd $Home\Desktop

Si te quedas atascado, sólo pide ayuda.

Ahora usa Python para ejecutar el código en el archivo así:

$ python3 python\_intro.py  
Hello, Django girls!

Nota: en Windows 'python3' no es reconocido como un comand. En cambio, utiliza 'python' para ejecutar el archivo:

> python python\_intro.py

¡Muy bien! Ejecutaste tu primer programa de Python desde un archivo. ¿No te sientes increíble?

Ahora puedes moverte a una herramienta esencial en la programación:

## 

## 

## **If...elif...else**

Un montón de cosas en el código sólo son ejecutadas cuando se cumplen las condiciones dadas. Por eso Python tiene algo llamado **sentencias if**.

Reemplaza el código en tu archivo **python\_intro.py** con esto:

if 3 > 2:

Si lo guardáramos y lo ejecutáramos, veríamos un error como este:

$ python3 python\_intro.py  
File "python\_intro.py", line 2  
 ^  
SyntaxError: unexpected EOF while parsing

Python espera que le demos más instrucciones las cuales se ejecutan si la condición 3 > 2 es verdadera (o True). Intentemos hacer que Python imprima "It works!". Cambia tu código en el archivo **python\_intro.py** para que se vea como esto:

if 3 > 2:  
 print('It works!')

*¿Observas cómo hemos indentado la siguiente línea de código con 4 espacios?* Necesitamos hacer esto para que Python sepa qué código ejecutar si el resultado es verdadero. Puedes poner un espacio, pero casi todas las programadores y los programadores Python hacen 4 espacios para hacer que el código sea más legible. Un solo tab también cuenta como 4 espacios.

Guárdalo y ejecútalo de nuevo:

$ python3 python\_intro.py  
It works!

Note: Recuerda que en Windows, 'python3' no es reconocido como un comando. De ahora en adelante, reemplaza 'python3' por 'python' para ejecutar el archivo.

### **¿Qué pasa si una condición no es verdadera?**

En ejemplos anteriores, el código fue ejecutado sólo cuando las condiciones eran ciertas. Pero Python también tiene declaraciones elif y else:

if 5 > 2:  
 print('5 is indeed greater than 2')  
else:  
 print('5 is not greater than 2')

Cuando esto se ejecute imprimirá:

$ python3 python\_intro.py  
5 is indeed greater than 2

Si 2 fuera un número mayor que 5, entonces el segundo comando sería ejecutado. Fácil, ¿verdad? Vamos a ver cómo funciona elif:

name = 'Sonja'  
if name == 'Ola':  
 print('Hey Ola!')  
elif name == 'Sonja':  
 print('Hey Sonja!')  
else:  
 print('Hey anonymous!')

y al ejecutarlo:

$ python3 python\_intro.py  
Hey Sonja!

¿Ves lo que pasó allí? elif te permite agregar condiciones adicionales que se ejecutan si la condición previa falla.

Puedes agregar tantas sentencias elif como quieras después de la sentencia if inicial. Por ejemplo:

volume = 57  
if volume < 20:  
 print("It's kinda quiet.")  
elif 20 <= volume < 40:  
 print("It's nice for background music")  
elif 40 <= volume < 60:  
 print("Perfect, I can hear all the details")  
elif 60 <= volume < 80:  
 print("Nice for parties")  
elif 80 <= volume < 100:  
 print("A bit loud!")  
else:  
 print("My ears are hurting! :(")

Python corre a través de cada prueba en secuencia e imprime:

$ python3 python\_intro.py  
Perfect, I can hear all the details

## **Comentarios**

Las líneas que comienzan con # son comentarios. Puedes escribir lo que quieras luego de # y Python lo ignorará. Los comentarios pueden hacer que tu código sea más fácil de entender para otras personas.

# Change the volume if it's too loud or too quiet  
if volume < 20 or volume > 80:  
 volume = 50  
 print("That's better!")

No necesitas escribir un comentario por cada línea de código, pero son útiles para explicar por qué tu código está haciendo algo, o proveer un resumen cuando se está haciendo algo complejo.

### **Resumen**

En los últimos tres ejercicios aprendiste acerca de:

* **Comparar cosas** - en Python puedes comparar cosas haciendo uso de >, >=, ==, <=, < y de los operadores and y or
* **Boolean** - un tipo de objeto que sólo puede tener uno de dos valores: True o False
* **Guardar archivos** - cómo almacenar código en archivos así puedes ejecutar programas más grandes
* **if... elif... else** - sentencias que te permiten ejecutar código sólo cuando se cumplen ciertas condiciones

¡Es hora de leer la última parte de este capítulo!

## **¡Tus propias funciones!**

Para lectores en casa: esta parte está cubierta en el video [Python Basics: Functions](https://www.youtube.com/watch?v=5owr-6suOl0) (en Inglés).

¿Recuerdas funciones como len() que puedes ejecutar en Python? Bueno, te tenemos buenas noticias, ¡ahora aprenderás a escribir tus propias funciones!

Una función es una secuencia de instrucciones que Python debe ejecutar. Cada función en Python comienza con la palabra clave def, se le asigna un nombre y puede tener algunos parámetros. Vamos a empezar con algo fácil. Reemplaza el código en **python\_intro.py** con lo siguiente:

def hi():  
 print('Hi there!')  
 print('How are you?')  
  
hi()

Bien, ¡nuestra primera función está lista!

Te preguntarás por qué hemos escrito el nombre de la función en la parte inferior del archivo. Esto es porque Python lee el archivo y lo ejecuta desde arriba hacia abajo. Así que para poder utilizar nuestra función, tenemos que reescribir su nombre en la parte inferior.

Ejecutemos esto y veamos qué sucede:

$ python3 python\_intro.py  
Hi there!  
How are you?

Nota: si no funciona, ¡no te desesperes!. El resultado te ayudará a entender porqué:

* Si recibiste un NameError, eso probablemente significa que has tipeado algo de forma incorrecta. Deberías comprobar que utilizaste el mismo nombre cuando creaste la función con def hi(): y cuando la llamaste con hi().
* Si recibiste un IndentationError, comprueba que ambas líneas print tengan la misma cantidad de espacios blancos desde el principio de la línea: Python quiere todo el código dentro de la función alineado correctamente.
* Si no hay ninguna respuesta, comprueba que el último hi() no esté indentado - si lo está, esa línea será parte de la función también y nunca será ejecutada.

Construyamos nuestra primera función con parámetros. Utilizaremos el ejemplo anterior - una función que dice 'Hi' a la persona que ejecuta el programa - con un nombre:

def hi(name):

Como puedes ver, ahora dimos a nuestra función un parámetro que llamamos name:

def hi(name):  
 if name == 'Ola':  
 print('Hi Ola!')  
 elif name == 'Sonja':  
 print('Hi Sonja!')  
 else:  
 print('Hi anonymous!')  
  
hi()

Recuerda: la función print está indentada cuatro espacios dentro de la condición if. Esto es porque la función se ejecutan cuando la condición se cumple. Vamos a ver cómo funciona:

$ python3 python\_intro.py  
Traceback (most recent call last):  
File "python\_intro.py", line 10, in <module>  
 hi()  
TypeError: hi() missing 1 required positional argument: 'name'

Oops, un error. Por suerte, Python nos da un mensaje de error bastante útil. Nos dice que la función hi() (la que definimos) tiene un argumento requerido (llamado name) y que se nos olvidó pasarlo al llamar a la función. Vamos a arreglarlo en la parte inferior del archivo:

hi("Ola")

Y lo ejecutamos de nuevo:

$ python3 python\_intro.py  
Hi Ola!

¿Y si cambiamos el nombre?

hi("Sonja")

Y lo ejecutamos:

$ python3 python\_intro.py  
Hi Sonja!

Ahora, ¿qué crees que suceda si escribes otro nombre ahí? (No Ola ni Sonja). Inténtalo y ve si tienes razón. Debería imprimir esto:

Hi anonymous!

Esto es increíble, ¿verdad? De esta forma no tienes que repetir todo cada vez que deseas cambiar el nombre de la persona a la que la función debería saludar. Y eso es exactamente por qué necesitamos funciones - ¡para no repetir tu código!

Vamos a hacer algo más inteligente - hay más de dos nombres, y escribir una condición para cada uno sería difícil, ¿no?

def hi(name):  
 print('Hi ' + name + '!')  
  
hi("Rachel")

Ahora vamos a llamar al código:

$ python3 python\_intro.py  
Hi Rachel!

¡Felicidades! Acabas de aprender cómo escribir funciones :)

## **Bucles**

Esta es la última parte. Eso fue rápido, ¿cierto? :)

A los programadores no les gusta repetir cosas. La programación intenta automatizar las cosas, así que no queremos saludar a cada persona por su nombre manualmente, ¿verdad? Es ahí donde los bucles se vuelven muy útiles.

¿Todavía recuerdas las listas? Hagamos una lista de las chicas:

girls = ['Rachel', 'Monica', 'Phoebe', 'Ola', 'You']

Queremos saludar a todas ellas por su nombre. Tenemos la función hi que hace eso, así que vamos a usarla en un bucle:

for name in girls:

La sentencia for se comporta de manera similar a la sentencia if, el código que sigue a continuación debe estar indentado usando cuatro espacios.

Aquí está el código completo que estará en el archivo:

def hi(name):  
 print('Hi ' + name + '!')  
  
girls = ['Rachel', 'Monica', 'Phoebe', 'Ola', 'You']  
for name in girls:  
 hi(name)  
 print('Next girl')

Y cuando lo ejecutamos:

$ python3 python\_intro.py  
Hi Rachel!  
Next girl  
Hi Monica!  
Next girl  
Hi Phoebe!  
Next girl  
Hi Ola!  
Next girl  
Hi You!  
Next girl

Como puedes ver, todo lo que pones con una indentación dentro de una sentencia for será repetido para cada elemento de la lista girls.

También puedes usar el for en números usando la función range:

for i in range(1, 6):  
 print(i)

Lo que imprimirá:

1  
2  
3  
4  
5

range es una función que crea una lista de números en serie (estos números son proporcionados por ti como parámetros).

Ten en cuenta que el segundo de estos dos números no será incluido en la lista que retornará Python (es decir, range(1, 6) cuenta desde 1 a 5, pero no incluye el número 6). Eso es porque "range" está medio-abierto, y con eso queremos decir que incluye el primer valor, pero no el último.

## **Resumen**

Eso es todo. **¡Eres genial!** Este fue un capítulo difícil, por lo que debes sentirte orgulloso de ti mismo. ¡Nosotros estamos orgullosos de ti porque has llegado lejos!

Tal vez quieras hacer algo distinto por un momento - estirarte, caminar un poco, descansar tus ojos - antes de pasar al siguiente capítulo. :)

