Practica #3 - Introducción a Python

Vamos a empezar con Python. Pero primero, déjame decirte qué es Python. Python es un lenguaje de programación muy popular que puede utilizarse para la creación de sitios web, juegos, software académico, gráficos y mucho, mucho más.

Python se originó en la década de 1980 y su objetivo principal es ser legible por los seres humanos, por eso parece mucho más simple que otros lenguajes de programación. Esto hace que sea más fácil de aprender, pero no solo eso, ¡Python es también muy poderoso!.

Python prompt

Para empezar a jugar con Python, tenemos que abrir una línea de comandos en nuestra computadora. Ya sabes cómo hacerlo, lo aprendiste en la práctica anterior.

Una vez abierta, sigue las siguientes instrucciones. Queremos abrir una consola de Python, así que escribe `python3` y pulsa **Enter**.

```
$ python3
Python 3.5.3 (...)
Type "copyright", "credits" or "license" for more information.
>>>
```

Primer comando en Python

Después de ejecutar el comando `python3` en *BASH*, 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 + D`. Luego no verás más `>>>`.

Pero ahora no queremos salir de la consola de Python. Queremos aprender más sobre ella. Por ejemplo, trata de escribir algo de matemáticas, como `2 + 3` y pulsa **Enter**.

```
>>> 2 + 3
5
```

Podrías intentar otros comandos como: `- 4 * 5 - 5 - 1 - 40 / 2`. Prueba a hacer todo tipo de operaciones, si crees que alguna no resulta como esperas consulta con tu profesor.

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

Strings

¿Y tu nombre? Escribe tu nombre de pila en frases como ésta:

```
>>> "Daw"
'Daw'
```

Has creado tu primer `string`. Es una secuencia de caracteres que puede ser procesada por una computadora. El `string` (o en español, cadena) debe comenzar y terminar con el mismo carácter. Esto puede ser comillas simples (') o dobles (") - ellas le dicen a Python que lo que esta dentro es una cadena.

Las cadenas pueden ser concatenadas. Prueba esto:

```
>>> "Hola " + "Daw"
'Hola Daw'
```

También puedes multiplicar las cadenas con un número:

```
>>> "Daw" * 3
'DawDawDaw'
```

Si necesitas poner un apóstrofe dentro de tu cadena, tienes dos maneras de hacerlo.

Usando comillas dobles:

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

o escapando el apóstrofe con una barra invertida ():

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

Bien, para ver tu nombre en letras mayúsculas, simplemente escribe:

```
>>> "Daw".upper()
'DAW'
```

Usaste la función `upper()` en tu cadena. Una función (como `upper()`) es un conjunto de instrucciones que Python tiene que realizar sobre un objeto determinado ("Daw") una vez que se llama.

Si quisieras saber el número de letras que contiene tu nombre, también existe una función para esto.

```
>>> len("Daw")
3
```

Te preguntarás por qué a veces se llama a las funciones con un `.` al final de una cadena (como `"Ola".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 `"Daw"` como un parámetro a la función `len()`.

Resumen

De acuerdo, 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. Ahora hagámoslo un poco más difícil.

Errores

Intentemos con algo nuevo. ¿Podríamos obtener la longitud de un número de la misma manera que obtuvimos la longitud de nuestro nombre? Teclea `len(304023)` y presiona **Enter**:

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

Obtuvimos nuestro primer error (al menos a propósito), dice que los objetos de tipo "`int`" (números enteros) no tienen ninguna longitud. ¿Qué podemos hacer ahora? Quizás podemos escribir el numero como un string. Los strings tienen longitud,

```
>>> len(str(304023))
6
```

Funciona. Utilizamos la función `str()` dentro de la función `len()`. `str()` convierte todo a strings.

- La función `str()` convierte cosas en `strings`
- La función `int()` convierte cosas en `integers`

> Importante: podemos convertir números en texto, pero no podemos necesariamente convertir texto en números.

Variables

Un concepto importante en programación son las variables. Una variable no es más que un nombre para alguna cosa para que puedas usarla más tarde. Los programadores usan estas variables para almacenar datos, hacer su código más legible y así no tener que seguir recordando qué hace cada cosa.

Supongamos que queremos crear una nueva variable llamada `name`:

```
>>> name = "Daw"
```

Cómo te has dado cuenta, el programa no regresa algo como lo hacia antes. Entonces, ¿Cómo sabemos que la variable existe realmente? Simplemente introduce `name` y pulsa **Enter**:

```
>>> name
'Daw'
```

Tu primer variable. Siempre podrás cambiar a lo que se refiere:

```
>>> name = "Goku"
>>> name
'Goku'
```

Puedes usarla dentro de funciones también:

```
>>> len(name)
4
```

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?

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

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.

La función `print`

Intenta esto:

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

Cuando sólo escribes `name`, el intérprete de Python responde con la representación del `string` 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 mejor.

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 `string` e `integers`, Python tiene toda clase de diferentes tipos de objetos. Ahora vamos a introducir uno llamado `list`. Las listas son exactamente lo que piensas que son: son objetos que son listas de otros objetos.

Crea una lista:

```
>>> []
[]
```

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

```
>>> 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?

```
>>> len(lottery)
6
```

Sí, `len()` puede darte el número de objetos en una lista. Útil, tal vez la ordenemos ahora:

```
>>> lottery.sort()
```

Esto no devuelve nada, sólo cambió el orden en que los números aparecen en la lista. Vamos a imprimir la lista otra vez y ver qué pasó:

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

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

¿Invirtamos ese orden?

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

Fácil, si quieres añadir 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. La computadora inicia la cuenta en `0`, así que el primer objeto en tu lista está en el índice `0`, el siguiente es `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 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.

Diccionarios

Un `dictionary` (en español, diccionario) es similar a una lista, pero accedes a valores usando una `key` en vez de un `index`. Una `key` (en español, clave) puede ser cualquier `string` o `integer`. La sintaxis para definir un diccionario vacío es:

```
>>> {}
{}
```

Esto demuestra que acabas de crear un diccionario vacío. Ahora, trata escribiendo el siguiente comando (intenta reemplazando con información propia):

```
>>> participant = {'name': 'Daw', 'country': 'Mexico', 'favorite_numbers': [7, 10, 89]}
```

Con este comando, acabas de crear una variable `participant` con tres pares `key-value`:

La clave `name` apunta al valor `Daw` (un objeto string), `country` apunta a `Mexico` (otro string), y `favorite_numbers` apunta a `[7, 10, 89]` (una `list` con tres números en ella).

Puedes verificar el contenido de claves individuales con esta sintaxis:

```
>>> print(participant['name'])
Daw
```

Lo ves, es similar a una lista. Pero no necesitas recordar el `index` - sólo el nombre.

¿Qué pasa si le pedimos a Python el valor de una `key` que no existe? ¿Puedes adivinar?

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

Este es un `KeyError`. Python te ayuda y te dice que la clave `age` no existe en este diccionario.

¿Cuándo utilizar un diccionario o una lista? Bueno, eso es un buen punto para reflexionar. Sólo ten una solución en mente antes de mirar la respuesta en la siguiente línea.

¿Sólo necesitas una secuencia ordenada de elementos? Usa una lista.

¿Necesitas asociar valores con `keys`, así puedes buscarlos eficientemente (usando las `keys`) más adelante? Utiliza un diccionario.

Los diccionarios, como las listas, son mutables, lo que significa que pueden ser cambiados después de ser creados. Puedes agregar nuevos pares `key/value` en el diccionario después de que ha sido creado, por ejemplo:

```
>>> participant['favorite_language'] = 'Python'
```

Como en las listas, el método `len()` en los diccionarios, devuelve el número de pares `key-value` en el diccionario. Adelante, escribe el comando:

```
>>> len(participant)
4
```

Espero tenga sentido hasta ahora.

Puedes utilizar el comando `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')
>>> participant
{'country': 'Mexico', 'favorite_language': 'Python', 'name': 'Daw'}
```

Como puedes ver en la salida, el par de `key-value` 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': 'Daw'}
```

Como puedes ver, el valor de la clave `country` ha sido modificado de `Mexico` a `Germany`.

Resumen

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

- errores - 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
- listas - listas de objetos almacenados en un orden determinado
- diccionarios - objetos almacenados como pares clave-valor

Compara cosas

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.

¿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
```

Los operadores `>` 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

Puedes darle a Python todos los números para comparar que quieras, y siempre te dará una respuesta - muy inteligente:

- `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` – y es probablemente el tipo más simple que existe.

Hay sólo dos objetos `Boolean` : - `True` - `False`

Pero para que Python entienda esto, es necesario que siempre lo escribas como `True` (primera letra mayúscula, con el resto de la letras minúsculas). `true` , `TRUE` , `tRUE` no funcionarán – sólo `True` es correcto. (Lo mismo aplica a `False` también, 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
```

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

¡Guárdalo!

Hasta ahora hemos estado escribiendo nuestro código Python en el intérprete, lo cual nos limita a 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 (desde la terminal `$ subl`)
- 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, instalamos un editor de código *Sublime text*. Tendremos que abrir el editor ahora y escribir algo de código en un archivo nuevo:

```
print('Hello world, in Python!')
```

> 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, pero ahora puedes ver que la función `print` es de un color diferente del `string` que está dentro de ella. Eso se denomina "syntax highlighting" (en español, resaltado de sintaxis), y es una gran ayuda cuando estás programando. Presta atención a los colores, y obtendrás una pista cuando te olvides de cerrar un `string` o cometes un error al escribir una palabra clave (como el `def` en una función, que veremos abajo). Esta es una de las razones por las cuales usar un editor de código.

Obviamente, ahora eres un desarrollador Python muy experimentado, así que eres 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 de cualquier manera que queramos, lo importante aquí es asegurarse que el archivo finalice con `.py`, esto le indica a nuestra computadora que este es un archivo ejecutable de Python y que Python puede correrlo.

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.

El comando se verá algo como esto:

```
cd ~/Desktop
```

Si te quedas atascado, sólo pide ayuda (pero recuerda que lo vimos en la práctica 2). Y luego usa Python para ejecutar el código en el archivo como sigue:

```
$ python3 python_intro.py
Hello world, in Python!
```

Ejecutaste tu primer programa de Python desde un archivo. 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 por 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 que se supone serán ejecutadas si la condición `3 > 2` resulta ser verdadera (o `True` en este caso). 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? Tenemos que hacer esto para que Python sepa qué código ejecutar si la comparación resulta verdadera. Puedes poner un espacio, pero casi todos los programadores Python hacen 4 espacios para hacer que el código sea más legible. Un solo **tab** también contará como 4 espacios.

Guárdalo y ejecútalo de nuevo:

```
$ python3 python_intro.py
It works!
```

¿Qué pasa si la 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')
```

Al ejecutar esto se 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, vamos a ver cómo funciona `elif`:

```
name = 'Goku'

if name == 'Daw':
    print('Hey Daw!')
elif name == 'Goku':
    print('Hey Goku!')
else:
    print('Hey anonymous!')
```

y al ejecutarlo:

```
$ python3 python_intro.py
Hey Goku!
```

¿Ves lo que pasó ahí?

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 esta práctica!

¡Tus propias funciones!

¿Recuerdas las funciones como `len()` que puedes ejecutar en Python? Bien, buenas noticias, ahora aprenderás cómo 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?
```

Eso fue fácil, vamos a construir 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 == 'Daw':
        print('Hi Daw!')
    elif name == 'Goku':
        print('Hi Goku!')
    else:
        print('Hi anonymous!')

hi()
```

Como puedes notar, tuvimos que poner dos indentaciones antes de la función `print` porque `if` necesita saber lo que debería ocurrir cuando se cumple la condición. 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("Daw")`. Y lo ejecutamos otra vez:

```
$ python3 python_intro.py
Hi Daw!
```

¿Y si cambiamos el nombre?, `hi("Goku")`. Y lo corremos:

```
$ python3 python_intro.py
Hi Goku!
```

Ahora, ¿Qué crees que pasará si escribes otro nombre allí? (No Daw o Goku). Pruébalo y verás si tienes razón. Esto debería imprimir:

```
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 el por qué necesitamos funciones - ¡para no repetir tu código!.

Podrías hacer tu aplicación un poco mejor y evitar el error cuando no asignas un nombre al llamar a tu función. Para esto puedes agregar un valor por defecto al parámetro de la función. Prueba así:

```
def hi(name=''):
    if name == 'Daw':
        print('Hi Daw!')
    elif name == 'Goku':
        print('Hi Goku!')
    else:
        print('Hi anonymous!')

hi()
```

Debería de ejecutarse así:

```
$ python3 python_intro.py
Hi anonymous!
```

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

```
def hi(name = ''):
    if name == ''
        name = anonymous

    print('Hi ' + name + '!')

hi("Dany")
```

Ahora vamos a llamar al código:

```
$ python3 python_intro.py
Hi Rachel!
```

¡Felicidades! Acabas de aprender cómo escribir funciones.

Bucles

Esta ya es la última parte. Como hemos mencionado, los programadores son perezosos, 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 los integrantes y ex-integrantes de *Metallica*:

```
metallica = ['James', 'Lars', 'Kirk', 'Robert', 'Cliff', 'Jason', 'Dave', 'Ron']
```

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

```
for name in metallica:
```

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 = ''):
    if name == ''
        name = anonymous

    print('Hi ' + name + '!')

metallica = ['James', 'Lars', 'Kirk', 'Robert', 'Cliff', 'Jason', 'Dave', 'Ron']

for name in metallica:
    hi(name)
    print('Next member')
```

y cuando lo ejecutamos:

```
$ python3 python_intro.py
Hi James!
Next member
Hi Lars!
Next member
Hi Robert!
Next member
Hi Cliff!
Next member
Hi Jason!
Next member
Hi Dave!
Next member
Hi Ron!
Next member
```

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

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
```

La función `range` 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).

Resumen

Eso es todo. Esto no fue tan fácil realmente, así que deberías sentirte orgulloso de ti mismo. ¡Deberías estar orgulloso de haber llegado y comprendido todo hasta el momento!

Tal vez quieras hacer algo distinto por un momento - Repetir todo, ir al curso en línea o leer un poco más de Python y Django.