Preguntas teóricas

1. ¿Qué es un condicional?

   Un condicional, tal como su nombre lo sugiere, es un tipo de proposición que se utiliza en programación para el manejo de condiciones. En términos formales, para decirlo de alguna manera, dicho condicional se basa en la *implicación* (p $\rightarrow$ q) de la lógica condicional: "Sí p, entonces q" (donde p es la condición y q la consecuencia). Ejemplo: "Sí mañana llueve, no iré a la playa".

   Ahora bien, en *python* así como en la mayoría de los programas de computación, las condiciones utilizdas no son como las del ejemplo anterior, sino más bien condiciones que involucran una relación entre dos términos o variables cualesquiera (a, b)[1]. Algunos ejemplos de condiciones basadas en relaciones de términos o variables (que representan ideas) son los siguientes:

   - "a es padre de b" (relación filial)
   - "$a > b$" (relación lógica)
   - "$a \in b$" (relación teoría de conjuntos)

   calificarla como *lógica* la distinguimos de otro tipo relaciones como podrían ser, entre otras, la relación de filiación ("a es hijo de b"), por ejemplo, o la de pertenencia ("a pertenece a b"), en la teoría de conjuntos.

   A continuación presentamos los 6 tipos de condicionales que pueden ser utilizados en python[2]:

   - igual: a == b
   -

2. ¿Cuáles son los diferentes tipos de bucles en Python? ¿Por qué son útiles?

   En *python* existen 2 grandes tipos de bucles o *loops*: *for* y *while*.

   - Bucle *while*

     Dicho tipo de bucle se caracteriza por ejecutar un código una y otra vez, hasta que la(s) condicion(es) especificada(s) sea(n) falsa(s). A continuación analizaremos algunos ejemplos para aclarar un poco más dicha caracterización.

[1] La condición del ejemplo anterior, me parece que es más bien una condición *fáctica*, en el sentido que depende de un hecho (y en particular de la experiencia de aquel hecho) para determinar su valor de verdad (si es verdadera o falsa).

[2] En relación a esto, véase, por ejemplo, https://www.w3schools.com/python/python_conditions.asp.

```
a = 1
while a < 10:
print("¡Hola mundo!")
```

Si ejecutamos este código, veremos que el programa imprimirá *infinítamente* el mensaje que aparece ahí. ¿Por qué infinitamente? Básicamente, porque dicho código posee dos *proposiciones* que de alguna manera no se "cruzan" nunca. Por un lado, una primera proposición que define el valor inicial de `a` como una constante e igual a `1`. Y por el otro lado, una proposición *condicional* que plantea que, mientras `a` sea menor que `10`, se imprima el mensaje que aparece entre comillas. Dado que `a` es definida como una constante o una variable estática, es decir, que no varía su valor luego de cada bucle, `a` nunca será mayor que `10`, haciendo que el valor de verdad de dicha condición sea siempre verdadera.

Para que no suceda lo anterior, a saber, una iteración que tienda al infinito[3], es necesario que el bucle cuente con lo que se llama un *contador*. Aquello permite que la variable se comporte de manera dinámica, aumentando o disminuyendo después de cada bucle.

```
a = 1
while a < 10:
    print("¡Hola mundo!")
    a = a + 1
```

- Bucle *for*

3. ¿Qué es una lista por comprensión en Python?
4. ¿Qué es un argumento en Python?
5. ¿Qué es una función Lambda en Python?
6. ¿Qué es un paquete pip?

   Se llama *pip* en python a un "sistema de gestion de paquetes". Su función principal es instalar y administrar los paquetes existentes de *python*. Dichos paquetes son obtenidos del Python Package Index, aunque es posible descargarlos de otros índices tambien[4].

   Una de las grandes ventajas de *pip* es la facilidad con la cual se pueden instalar y desinstalar los paquetes de software desde la "línea de comandos": basta con un simple comando como el siguiente para hacerlo:

[3] Claramente un *script* o código que "dé vueltas en redondo" presenta un error.
[4] El índice de paquetes de Python (PyPI) es un gran repositorio de *software* para el lenguaje de programación python.

```
C:> py -m pip install nombre-paquete
```

Tal como está señalado en la documentación de *pip*, para poder empezar a utilizarlo, es necesario tenerlo *ya* instalado en nuestro sistema junto con *python*. Podemos revisar si aquello es así a través de los siguientes comandos[5]:

```
C:> py --version
Python 3.N.N
C:> py -m pip --version
pip X.Y.Z from ... (python 3.N.N)
```

Ahora bien, en el caso que *pip* no se encuentre ya instalado en su sistema, existen dos maneras de hacerlo:

- `ensurepip`
- `get-pip.py`

En el caso de utilizar `get-pip.py`, es necesario ejecutar los dos siguientes pasos:

- Descargar el *script* desde https://bootstrap.pypa.io/get-pip.py.
- Abrir el terminal, luego `cd` a la carpeta que contiene el archivo recién descargado `get-pip.py`, y finalmente ejecutar:

  ```
  C:> py get-pip.py
  ```

Una vez *ya* instalado el programa *pip* en nuestro sitema, podemos entonces comenzar a descargar los distintos paquetes que necesitemos mediante el comando inicial.

[5] Dependiendo del sistema operativo que tengamos en nuestro ordenador, sea Linux, MacOS o Windows, el comando empleado podrá variar un poco. En el siguiente cuadro solo he colocado los comandos utilizados en Windows.