Práctica 1

1. Entorno de desarrollo

Antes de hacer la práctica, será necesario instalar el entorno de desarrollo y las herramientas que estarán utilizando a lo largo del semestre. El proceso se desglosa a continuación.

1.1. Python con NLTK

Si están en Windows, pueden instalar Anaconda:

```
https://docs.anaconda.com/anaconda/install/windows/
```

Instalen Python 3. Python 3 maneja utf-8 (*e.g.* los acentos) por defecto.
Les recomiendo que creen un espacio virtual con el comando `conda`:

```
conda create -n practica01
conda activate practica01
```

Una vez que el espacio está activo, pueden instalar paquetes.

```
conda install nltk
```

Ejecuten Python e instalen los paquetes de nltk.

```
import nltk

nltk.download()
```

Anaconda viene con un editor de código por defecto: *Spyder*. Si no está instalado, pueden hacerlo de este modo.

```
conda install spyder
spyder
```

Finalmente, para salir del espacio virtual:

```
conda deactivate
```

Todas instalaciones posteriores de paquetes pueden hacerse con: `conda install <nombre>`. Anaconda también cuenta con un gestor visual donde pueden instalar/remover/lanzar paquetes sin usar la línea de comandos: Anaconda Navigator.

1.1.1. Otros paquetes

Otros paquetes útiles que podrían usar durante el semestre:

- **Numpy**: Programación vectorial en Python.
- **Scipy**: Herramientas para cómputo científico en Python.

- **scikit-learn**: Herramientas de ML en Python.
- **Pandas**: Análisis de datos en Python.
- **Matplotlib**: Visualización de datos en Python.
  - **Seaborn**: Interfaz de alto nivel para Matplotlib.
  - **Networkx**: Visualización de redes en Python.
- **spaCy**: Algoritmos PLN del estado del arte en Python.

Todos se pueden instalar con `conda` o `pip`. Procuren sólo instalar los que necesitan en su ambiente virtual.

1.2. Git

Instalen Git.

```
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
```

Configuren Git.

```
git config --global user.name "Juan Perez"
git config --global user.email juanperez@correo.com
```

Creen un repositorio. En linea de comandos, en una carpeta vacía (*e.g.* "Práctica_01/"):

```
git init
```

Creen un archivo README siguiendo la siguiente plantilla en Markdown (pueden simplemente copiar y pegar).

````
# Práctica 1

Este es el archivo README de la práctica 1.

### Prerequisitos

Aquí van los paquetes que se necesitan para que esta cosa funcione.

```
Dar ejemplos:

* NLTK
```

### Instalar

Aquí van las instrucciones de instalación del proyecto, de modo que
    alguien más lo pueda ejecutar sin ayuda del desarrollador.
````

```
```
import nltk

nltk.download()
```

Y seguir:

```
```
otra instrucción
```

Autor

* **Juan Perez** - *Trabajo inicial* - [Sitio web](https://google.com)
```

Les recomiendo visitar: `https://www.makeareadme.com/`

Añadan el README al repositorio git y hagan su primer `commit`.

```
git add README
git commit -a
```

Si desean pueden introducir un archivo ".gitignore" en el repositorio. Sirve para decirle a Git que ignore archivos innecesarios presentes en el repositorio (*e.g.* logs o ejecutables). Aquí hay plantillas para proyectos en varios lenguajes: `https://github.com/github/gitignore`.

Otros comandos útiles:

```
git status
```

Útil para ver el estado del repositorio (*e.g.* si hay cambios, si hay archivos nuevos, etc).

```
git log
```

Para ver la lista de `commits`. Una hoja de ayuda con los comandos de git se encuentra en la carpeta de la sesión.

Cada repo tendrá la lista de requerimientos obtenida con `conda` o `list`:

```
conda list
```

La salida del comando será guardada en un archivo "requerimientos.txt". En sistemas UNIX:

```
conda list > requerimientos.txt
```

El archivo tendrá que añadirse al repo.

```
git add .
git commit -a
```

Esto permitirá reproducir el ambiente en otras computadoras sin problemas al compartirlo.
```

2. GitHub

Cada práctica será subida a un repositorio en GitHub.

1. Creen una cuenta en el sitio.
2. Creen un repositorio nuevo llamado "Practica01".
3. Sigan las instrucciones en pantalla para subir el repositorio.

A partir de ahora, la ejecución del comando `git push` enviará el código a GitHub.

Como en todo gestor de versiones, se pueden crear ramas del repositorio o empujarlo a otras ramas remotas. Para más información, revisen la documentación de Git.

3. Diagnóstico

Los siguientes ejercicios son para determinar su nivel en Python. Están acomodados del más fácil al más difícil. Realicen los que puedan.

3.1. Ejercicio: Suma de enteros (5 minutos)

Escriban un script que reciba un número n y sume todos los números enteros de 1 hasta n (incluido). El resultado se imprime en pantalla. Por ejemplo, si el script recibe 2, éste suma $1 + 2$ e imprime 3. Si recibe 6, éste suma $1 + 2 + 3 + 4 + 5 + 6$ e imprime 21.

3.2. Ejercicio: Lenguaje de la F (10 minutos)

Escriban un programa que reciba una entrada de texto desde el teclado y sustituya todas las vocales de la oración según las reglas del Lenguaje de la F:

a	→	afa	e	→	efe
i	→	ifi	o	→	ofo
u	→	ufu	y	→	yfi (cuando suena como vocal)

Si la vocal está acentuada, el acento irá antes de la "f": í → ífi. Ejemplos:

Buenos Dias → Bufuefenofos Difiafas
Hasta Luego → Hafastafa Lufuefegofo
Está frío, güey → Efestáfa frífiofo, güfuefeyfi

No hace falta respetar mayúsculas y minúsculas. La "y" es opcional.

3.3. Ejercicio: Ahorcado (45 minutos)

Desarrollen un programa que permita jugar Ahorcado.

3.3.1. Clase `Ahorcado`

Escriban una clase `Ahorcado` que se inicialize con:

- Una palabra a adivinar p.
- El número de errores máximos E permitidos al jugador antes de perder.
- Un método `jugar` que reciba un sólo caracter c.
- Un contador de errores cometidos e.

El método `jugar` debe revisar si c está en la palabra p. Si no está, se suma al contador de errores ($e += 1$). Si sí está, el objeto debe recordar que recibió c.

Además, la clase `Ahorcado` debe contar con un método `estado` que imprima en orden las letras adivinadas de la palabra p (con guiones donde faltan letras), así como la cantidad de errores que aún puede cometer el usuario antes de perder. Además, el método `estado` tiene que desplegar un mensaje "ganaste" o "perdiste", dependiendo si el usuario ganó (adivinó p) o perdió ($e > E$).

Por ejemplo, para crear un ahorcado con la palabra "galleta" y 4 errores posibles:

```
ahorcado = Ahorcado("galleta", 4)
```

Entonces, `ahorcado.estado()` regresaría:

```
_ _ _ _ _ _ _ (4 errores posibles)
```

Finalmente, el método `jugar` debe llamar al método `estado` antes de terminar. Así, llamar `ahorcado.jugar(``a'')` regresaría:

```
_ a _ _ _ _ a (4 errores posibles)
```

Desde aquí, llamar `ahorcado.jugar(``b'')` regresaría:

```
_ a _ _ _ _ a (3 errores posibles)
```

3.3.2. Juego del ahorcado

Escriba un programa que use la clase Ahorcado en una sesión de juego. Al ejecutar, el programa debe seleccionar una palabra al azar de una lista de palabras[1] y una cantidad aleatoria de errores permitidos (entre 3 y 7). Un ejemplo de una ejecución se muestra a continuación.

[1] `https://raw.githubusercontent.com/javierarce/palabras/master/listado-general.txt`

```
_ _ _ _ _ _ _ (4 errores posibles)

Introduzca una letra: a
_ a _ _ _ _ a (4 errores posibles)

Introduzca una letra: b
_ a _ _ _ _ a (3 errores posibles)

Introduzca una letra: c
_ a _ _ _ _ a (2 errores posibles)

Introduzca una letra: g
g a _ _ _ _ a (2 errores posibles)

Introduzca una letra: o
g a _ _ _ _ a (1 errores posibles)

Introduzca una letra: e
g a _ _ e _ a (1 errores posibles)

Introduzca una letra: z
g a l l e t a (perdiste)
```

4. Subir respuestas a Github

En el archivo README, añadan un título y una descripción a la práctica. **No olviden poner su nombre**. Finalmente, antes de terminar la clase, ejecuten lo siguiente en consola.

```
git add .
git commit -a
git push
```

Ésto debe subir el código que hicieron a Github. Envíenme el nombre de su repositorio a me@arturocuriel.com.