Universidad Pública de Navarra
Curso 2024/2025

Informática - Práctica de laboratorio 02

Programación: Estructuras alternativas y Funciones

1. Normas de Entrega

Todas las prácticas deberán entregarse siguiendo una convención de nombres y un formato específico. En el caso de las prácticas de Python, se solicitará un script de Python para cada ejercicio, generando así varios archivos de texto por práctica.

Todas las prácticas se desarrollarán utilizando el IDE `spyder`. `Spyder` es un entorno científico de código abierto y gratuito diseñado para científicos, ingenieros y analistas de datos. Ofrece una combinación única de funcionalidades avanzadas para edición, análisis, depuración y perfilado, con capacidades excelentes para exploración de datos, ejecución interactiva, inspección profunda y visualización de paquetes científicos. Puedes acceder a él aquí: `https://www.spyder-ide.org/`.

1.1. Nombre del Archivo

Cada ejercicio de programación debe estar codificado en un archivo independiente, es decir, cada ejercicio será un archivo separado. A menos que se indique lo contrario, la convención de nombres será *Ejercicio_YY.py*, donde *XX* será el número de la práctica y *YY* el número del ejercicio en el documento de la práctica. Por ejemplo, el primer ejercicio de la práctica cero llevará el nombre *Ejercicio_01.py*.

Para entregar los ejercicios, debes acceder a la sección del curso en mi aula virtual (*20XX_0_501103_91_G*). Allí, en la columna izquierda, aparecerá la sección *Tareas*.

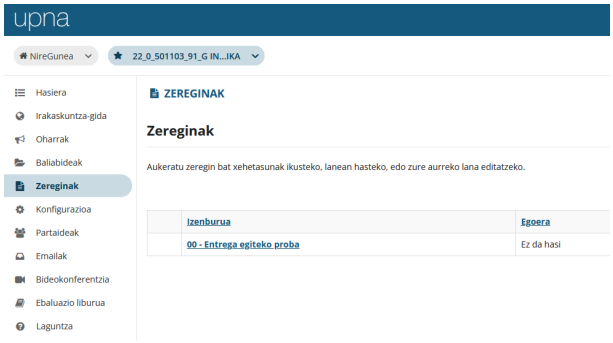

Figura 1: Primer paso para entregar las tareas.

En cada entrega aparecerá una tarea y podrás entregar los ejercicios. También tendrás la opción de escribir un mensaje junto con la entrega. Para cada práctica, se solicitarán diferentes ejercicios, y deberás subir un archivo para cada ejercicio (ver imagen 2).

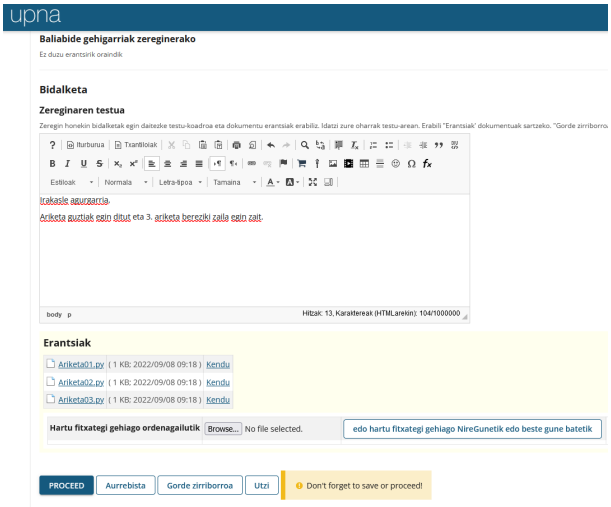

Figura 2: Primer paso para entregar las tareas.

2. Normas de Desarrollo

Con cada práctica tendrás una chuleta (LabXXPR_CheatSheet). En la chuleta encontrarás un resumen del material trabajado en clase. Por ejemplo, en Lab01PR_CheatSheet, se presentarán los elementos necesarios para realizar Lab01PR. Se recomienda imprimir la chuleta o, al menos, tenerla a la vista mientras desarrollas el programa. La CheatSheet contendrá tipos de datos, operadores, estructuras y funciones trabajadas en clase. Cualquier elemento de Python utilizado fuera de estos será evaluado con una nota de 0 en los evaluables.

Al desarrollar los programas, debes seguir las siguientes normas

1. Los nombres de las variables deben ser descriptivos
   - Todos en minúsculas
   - Los nombres compuestos por más de una palabra se separarán con '_'
2. Todos los programas deben tener una cabecera obligatoria
   - La primera línea de comentario debe ser `# python script`
   - En la segunda línea debe aparecer `# Autor: tu nombre`, donde `tu nombre` es el nombre del estudiante

- En la tercera línea debe aparecer `# Descripción: descripción del programa`, donde `descripción del programa` es una frase o párrafo que describa el programa

3. No se deben usar funciones que no aparezcan en la CheatSheet.

3. Lista de Ejercicios

En esta lista de ejercicios se presentan problemas de dos temas diferentes: estructuras de control `if` y diseño de funciones. En el primer bloque de ejercicios se trabaja con la estructura de control `if`, sin embargo, todos los ejercicios también pueden ser programados utilizando funciones. Consulta el ejercicio 1, donde se solicita un programa que determine si un número es positivo o negativo.

Ejercicio 1. Escribe un programa que determine si un número almacenado en una variable es positivo o negativo.

Para abordar este ejercicio, primero debemos tener en mente la estructura que utilizaremos. Recuerda que todos los programas creados en esta lección deben estar estructurados en tres bloques: 1) asignación de variables, 2) lógica de la aplicación y 3) presentación del resultado en pantalla.

Así, utilizaremos los tres bloques para resolver este ejercicio. Para determinar si un número es positivo o no, primero debemos formular la pregunta matemática que debe cumplirse. Un número será positivo si es mayor que 0, es decir, debemos formular la pregunta $nmero > 0$. Esta expresión lógica generará un valor booleano `True` o `False`. Esta sentencia se puede convertir directamente en programación.

Código 1: El número es positivo.

```
# Asignacion de la variable
numero = 5

# Logica del programa
if numero > 0:
  print('El numero es positivo.')
else:
  print('El numero no es positivo.')
```

Consulta el script en 1, que muestra la solución del Ejercicio 1. En este caso, y dado que el ejercicio es muy simple, la lógica del programa y la presentación del resultado aparecen en un solo bloque. Aunque este ejercicio es muy simple, también se puede realizar utilizando funciones.

Código 2: El número es positivo con funciones.

```
# Definicion de funciones
def es_positivo(numero):
  if numero > 0:
    print('El numero es positivo.')
  else:
    print('El numero no es positivo.')

# Asignacion de la variable
variable = 5

# Logica del programa
```

```
12 es_positivo(variable)
```

Consulta el script en 2. Aquí, la lógica del programa se define dentro de una función, proporcionando una estructura más organizada. Al definir una nueva función, hemos añadido un nuevo bloque, el bloque de funciones. El bloque de funciones siempre debe estar por encima de los bloques de variables.

Código 3: El número es positivo con funciones y print al final.

```
1  # Definicion de funciones
2  def es_positivo(numero):
3    return numero > 0
4
5  # Asignacion de la variable
6  variable = 5
7
8  # Presentacion del resultado
9  if es_positivo(variable):
10   print('El numero es positivo.')
11 else:
12   print('El numero no es positivo.')
```

Aunque el Ejercicio 1 es muy simple, es recomendable presentar los cuatro bloques y añadir el bloque de funciones. Normalmente, la lógica del programa se desarrollará en funciones y la presentación se hará al final del programa. Consulta el script en 3. En él, hemos guardado la lógica del programa dentro de la función `es_positivo` y la presentación del resultado se realiza al final, garantizando así la legibilidad.
□

3.1. Estructuras de Control: `if`

Ejercicio 2. Escribe un programa que determine si el número almacenado en una variable es par o impar.

□

Ejercicio 3. Escribe un programa que asigne tres números y muestre cuál de ellos es el mayor.

□

Ejercicio 4. Asigna un número que represente un año a una variable y escribe un programa que determine si ese año es bisiesto o no. Consulta cómo calcular si un año es bisiesto aquí https://es.wikipedia.org/wiki/A%C3%B1o_bisiesto.

□

Ejercicio 5. Asigna una variable con la edad y escribe un programa que verifique si cumple con los requisitos para obtener una licencia de conducir, es decir, si tiene 18 años o más.

⊡

Ejercicio 6. Escribe una calculadora básica que pueda realizar operaciones de suma, resta, multiplicación y división. Para que el programa funcione, deberá asignar tres variables al inicio: dos números y un operador matemático en forma de cadena de caracteres.

⊡

Ejercicio 7. Escribe un programa que asigne dos números reales y luego muestre diferentes mensajes según el signo de los dos números: `ambos números son positivos`, `ambos números son negativos` o `uno de los números es negativo y el otro es positivo`. El programador decidirá si el cero es positivo o negativo.

Ejemplo: Si los números asignados son `-2` y `7`, el mensaje será `uno de los números es negativo y el otro es positivo`. Si los números son `-2` y `-7`, el mensaje será `ambos números son negativos`.

⊡

Ejercicio 8. Escribe un programa que asigne un número que represente un día de la semana (1 para lunes, 2 para martes, etc.) y luego muestre el nombre del día correspondiente. Si el valor asignado no está en el rango de 1 a 7, el mensaje debe ser `Valor desconocido`.

⊡

Ejercicio 9. Escribe un programa que asigne un número que represente un mes (1 para enero, 2 para febrero, etc.) y un año. Luego, muestra la cantidad de días en ese mes. Ten en cuenta los años bisiestos para febrero (29 días). Puedes utilizar el resultado del Ejercicio 4 para esto. Si el valor asignado no está en el rango de 1 a 12, el mensaje debe ser `Valor desconocido`.

⊡

Ejercicio 10. Escribe un programa que asigne las longitudes de los tres lados de un triángulo y luego determine si es equilátero (todos los lados iguales), isósceles (dos lados iguales) o escaleno (ningún lado igual).

⊡

Ejercicio 11. Escribe un programa que asigne dos números enteros y determine si la división entre ellos es exacta o no.

Ejemplo: Si los números proporcionados por el usuario son `4` y `2`, el programa mostrará `la división entre los números es exacta`.

⊡

Ejercicio 12. Asigna una edad y crea un programa que muestre la categoría de edad correspondiente. Las categorías de edad son: `niño` (menos de 12 años), `adolescente` (entre 12 y 18 años), `joven` (entre 18 y 30 años) o `adulto` (más de 30 años). Si el valor asignado es menor que 0, el mensaje debe ser `Valor desconocido`.

⊡

Ejercicio 13. Escribe un programa que asigne una letra del alfabeto y determine si es una vocal o una consonante. La letra debe ser asignada a una variable al inicio del programa. Puedes utilizar el operador `in`.

⊡

Ejercicio 14. Escribe un programa que asigne el año de nacimiento y determine la generación correspondiente. Las categorías de generación son: `Baby Boomer` (1946-1964), `Generación X` (1965-1980), `Millennials` (1981-1996), `Generación Z` (1997-2012) o `Generación Alfa` (2013-presente). Si el valor asignado es menor que 1946, el mensaje debe ser `Valor desconocido`.

⊡

Ejercicio 15. Escribe un programa que asigne tres números y muestre cuál de ellos es el mayor.

⊡

Ejercicio 16. Escribe un programa que ordene tres números enteros de menor a mayor.

Ejemplo: Si los valores son -2, 7 y 1, el mensaje debe mostrar `-2`, `1` y `7`.

⊡

3.2. Funciones

Ejercicio 17. Crea una función llamada `sumar` que tome dos números como argumentos y devuelva la suma de esos números.

⊡

Ejercicio 18. Define una función llamada `restar` que acepte dos números como parámetros y devuelva la resta del primero menos el segundo.

⊡

Ejercicio 19. Escribe una función llamada `multiplicar` que tome dos números como entrada y regrese su producto.

⊡

Ejercicio 20. Crea una función llamada `dividir` que reciba dos números como argumentos y devuelva el resultado de la división del primero por el segundo. ¿Qué ocurre si el divisor es 0?

⊡

Ejercicio 21. Escribe una función llamada `potencia` que tome dos números como argumentos y calcule el resultado de elevar el primero a la potencia del segundo.

⊡

Ejercicio 22. Utiliza las funciones creadas en 17, 18, 19, 20, 21 y la estructura de cálculo de la calculadora de 6 para crear una calculadora completa. ¿Qué sucede si la calculadora necesita realizar una división y el divisor es 0?

⊡

Ejercicio 23. Define una función para calcular el precio final de un producto después de aplicar un descuento. La función debe tomar como argumentos el precio original y el porcentaje de descuento, y devolver el precio final. Luego, el programa debe mostrar el precio a pagar.

⊡

Ejercicio 24. Define una función llamada `area_triangulo` que calcule el área de un triángulo. La función debe tomar como argumentos la base y la altura del triángulo y devolver su área.

⊡

Ejercicio 25. Escribe una función llamada `area_circulo` que calcule el área de un círculo. La función debe tomar como argumento el radio del círculo y devolver su área.

⊡

Ejercicio 26. Crea una función llamada `es_par` que determine si un número es par. La función debe devolver `True` si el número es par y `False` si es impar.

⊡

Ejercicio 27. Define una función que determine si un número es positivo o negativo. A diferencia del ejercicio 1, en este caso, el 0 no se considerará ni positivo ni negativo.

⊡

Ejercicio 28. Utiliza las funciones definidas en 26 y 27 para crear un programa que determine si un número es par o impar y si es positivo o negativo. El programa debe mostrar un único mensaje que combine ambos resultados.

⊡

Ejercicio 29. Crea una función llamada `area_cuadrado` que calcule el área de un cuadrado. La función debe tomar como parámetro la longitud de un lado del cuadrado y devolver su área.

⊡

Ejercicio 30. Escribe una función llamada `area_rectangulo` que calcule el área de un rectángulo. La función debe tomar como parámetros la longitud y el ancho del rectángulo y devolver su área.

⊡

Ejercicio 31. Define una función llamada `area_triangulo_equilatero` que calcule el área de un triángulo equilátero. La función debe tomar como argumento la longitud de un lado del triángulo equilátero y devolver su área.

⊡

Ejercicio 32. Define una función llamada `es_bisiesto` que determine si un año es bisiesto. La función debe tomar un año como argumento y devolver `True` si es bisiesto y `False` si no lo es.

⊡

Ejercicio 33. Define una función llamada `concatenar` que tome dos cadenas como argumentos y las concatene en una sola cadena.

⊡

Ejercicio 34. Define una función que calcule el promedio ponderado de cuatro calificaciones, donde cada calificación tiene un peso diferente: examen (48 %), pruebas en clase (7 %), evaluaciones de trabajos (30 %) y trabajo en grupo (15 %). La función debe tomar como argumentos las cuatro calificaciones. Para que el promedio ponderado sea calculado, la calificación del examen debe ser al menos 5.0. La función debe devolver `True` si el promedio ponderado es mayor o igual a 60 (indicando que el estudiante ha aprobado), y `False` en el caso contrario.

⊡

Ejercicio 35. Escribe una función que resuelva una ecuación cuadrática de la forma $ax^2 + bx + c = 0$. La función debe tomar como argumentos los coeficientes `a`, `b` y `c`. Asegúrate de que $b^2 \geq 4ac$, y calcula las dos soluciones usando las fórmulas:

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

La función debe devolver las dos soluciones si existen. Si no hay soluciones reales, la función debe devolver el mensaje `Con esos valores de entrada, la formula no tiene solución`.

⊡