Práctica 6:
Bucles I

Índice

Objetivos

Objetivos

- Depuración de programas.
- Excepciones: generación y captura de excepciones.
- Uso de estructuras de repetición while y for.
- Reforzar la estructuración del código en módulos.
- Selección de la estructura de repetición correcta para cada caso concreto.

Objetivos

En esta sesión se plantea desarrollar una aplicación con menú que permita ejecutar diferentes funciones matemáticas: desde la suma de enteros, pasando por mínimo común múltiplo, factoriales, desarrollos de seno y coseno por series de Taylor... Un conjunto importante de funciones.

Prueba el ejecutable de la aplicación disponible en Campus Virtual.

Planificación

Idea general

Primero se introducen los bucles while y for (Parte 1).

Seguidamente, resolveremos el diseño de interfaz de usuario con un menú de aplicación (Partes 2 y 3).

A continuación, se demostrará la importancia de la depuración (Parte 4).

Finalmente, se añadirán funciones a la calculadora (Parte 5).

Organización

Parte 1: Introducción a bucles while/for Warm up!

Parte 2: Manejo de excepciones.

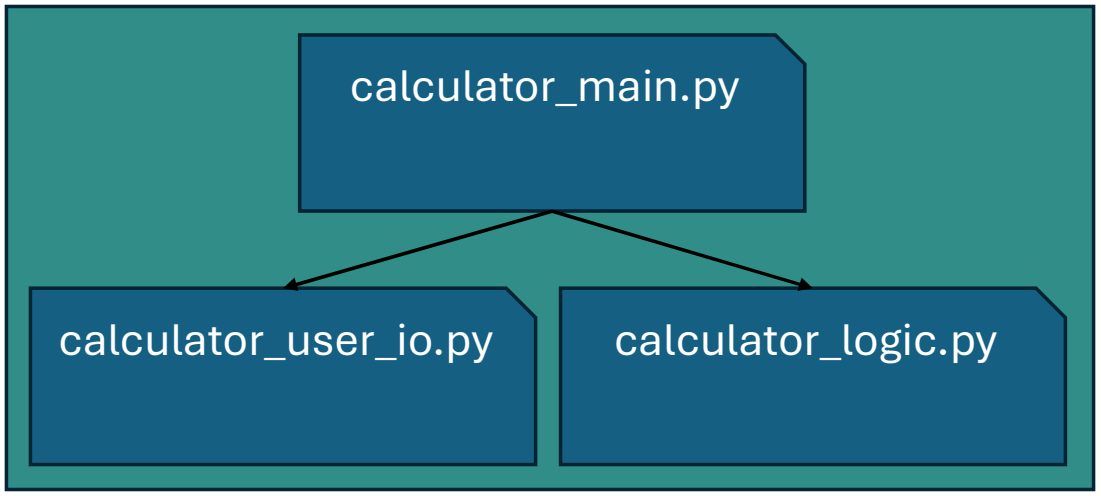
Parte 3: Las interfaces con el usuario

Parte 4: Validación de datos de usuario

Parte 5: Depuración de código

Parte 6: Ejercicios extra

Organización del código



Parte 1: Introducción a bucles while/for

Parte 1: Introducción a los bucles while/for

Descarga el archivo ***calculator_logic.py*** desde Campus Virtual. Este archivo contiene la declaración y la documentación –con doctest- de 3 funciones:

- **calculate_summatory_while**: suma los enteros entre dos límites dados UTILIZANDO UN BUCLE WHILE!
- **calculate_summatory**: suma los enteros entre dos límites dados UTILIZANDO UN BUCLE FOR!
  - ÉSTA SERÍA LA VERSIÓN QUE SE DEBERÍA REALMENTE IMPLEMENTAR Y USAR, ¿POR QUÉ?
- **concatenate_numbers_ascending**: retorna una cadena de texto concatenando los números enteros entre dos límites dados.

Implementa las funciones pedidas y ejecuta el módulo para comprobar que los casos de uso se cumplen.

Parte 2: Tratamiento de Excepciones

Parte 2: Tratamiento de excepciones

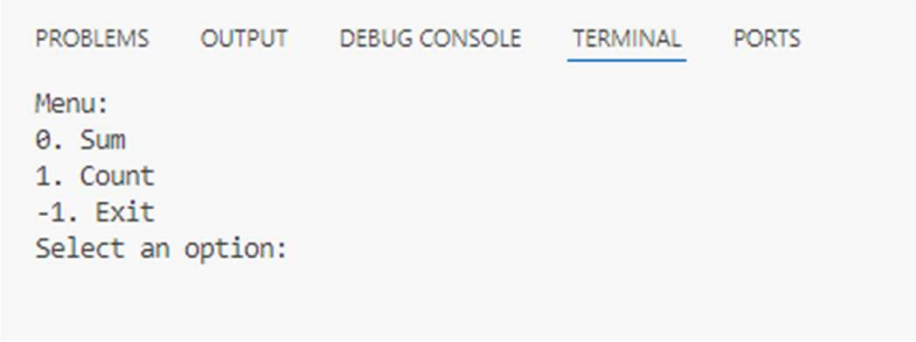
- Descarga el fichero **exceptions.py** desde Campus Virtual.
- En él se pueden observar dos funciones:
  - **ask_for_positive_integer** que retorna un entero mayor que 0. También comprueba que no haya errores en la entrada de datos y que el valor introducido sea positivo.
  - **test_is_bigger_and_positive** comprueba si el primer número es mayor que el segundo. Si alguno de ellos no es entero, entonces genera una excepción de tipo **TypeError**.
- El programa principal comprueba el caso de excepción en *test_is_bigger_and_positive*, el caso de dos enteros y, finalmente, pide evaluar *ask_for_positive_integer* con los diferentes posibles casos (el usuario debe introducir 'a', -5 y 10).

Parte 3: Las interfaces de usuario

Parte 3: Las interfaces con el usuario

El objetivo es crear un menú para la aplicación, de manera que el usuario seleccione la opción que desea ejecutar.

Ejemplo de lo que vamos a crear:



Tenemos 3 opciones: calcular la suma de enteros, concatenar enteros o salir del programa.

Para este menú necesitamos: i) imprimir a pantalla las cadenas necesarias, ii) pedir al usuario que seleccione la opción deseada, y iii) ejecutar la opción seleccionada.

Mientras no se seleccione salir, se seguirán repitiendo los tres pasos anteriores.

Parte 3: Las interfaces con el usuario

- El menú lo implementaremos en el módulo ***calculator_main.py***, desde donde llamaremos a las funciones en ***calculator_logic.py***.
- Para facilitar la codificación, se definirán una serie de globales:

```
SUMMATORY_OPTION = 0
COUNT_NUMBERS_OPTION = 1
EXIT_OPTION = -1
NO_OPTION = -2

MIN_OPTION_VALUE = EXIT_OPTION
MAX_OPTION_VALUE = COUNT_NUMBERS_OPTION
```

Opciones de menú

Parte 3: Las interfaces con el usuario

- Se escribirán funciones para i) mostrar a pantalla y pedir opción, ii) ejecutar cada una de las opciones al completo, y iii) manejador de opción. Además de la función main!
- La función **show_menu** se encargará de mostrar las diferentes cadenas que componen el menú, conjuntamente con el valor numérico asociado: Llamar al sumatorio (0), llamar al concatenado (1), o salir(-1).
- La función **ask_menu_option** solicita un valor entero al usuario, si no está dentro de los límites de opción entonces genera una excepción *ValueError* indicando el valor seleccionado, así como los límites. Los límites son variables globales definidas.

Parte 3: Las interfaces con el usuario

- La función **run_summatory** será la encargada de pedir dos valores enteros inicial y final. Si inicial es menor que final llamará a *calculate_summatory_while* mostrando el resultado obtenido; si no, imprimirá un mensaje de error.
- La función **run_concatenate_numbers_ascending** hace lo propio con *concatenate_numbers_ascending*.

- La función **handle_option** recibe la opción numérica seleccionada, llamando a la correspondiente función de entre las dos anteriores. En caso de haber seleccionado salir, muestra un mensaje con información de tal hecho.
- Finalmente, la función **main**, de manera iterativa, muestra las opciones posibles, pide la opción seleccionada y ejecuta la correspondiente tarea. Estos pasos se repetirán siempre que dicha opción no sea la de finalizar la aplicación.

Parte 4: Validación de datos de usuario

Parte 4: Validación de datos de usuario

En la Parte 3 ¿te fijaste que las funciones…?

- Las funciones calculate_*summatory* y *concatenate_numbers_ascending* generan excepciones en caso de argumentos no válidos.
- Las funciones *run_summatory* y *run_concatenate_numbers_ascending*, controlan que los valores de los enteros pedidos tengan valores coherentes.
- En *handle_option* también se estudia el valor del argumento.

¿Por qué?

- Las funciones de lógica asumen un procesado previo de los valores pasados como argumentos: si no son válidos, se generará una excepción.
- Las funciones que actúan solicitando datos al usuario son las responsables de garantizar estos valores adecuados para los argumentos.
- Estas funciones repetirán tantas veces como sea necesario la adquisición de datos hasta que se obtengan datos válidos!

Parte 4: Validación de datos de usuario

- En esta parte nos centraremos en:
  - 4.1 Modificar el código escrito en la parte 3 para disponer de datos de usuario válidos.
  - 4.2 Introduciremos una nueva capacidad (cálculo del máximo común divisor de dos enteros). Introducir casos de prueba con doctest en la documentación.
  - 4.3 Escribiremos la función de petición de datos válidos.
  - 4.4 Extenderemos el módulo calculator_main con la nueva funcionalidad.

Parte 4.1 Mejorar el código de la parte 3

- Crearemos el módulo **calculator_user_io.py**.
- Escribir la función **ask_inicial_and_final_integers** que retorna una tupla con dos valores enteros inicial y final tal que inicial < final. Mientras no se cumpla esta condición se pedirán ambos valores por teclado. En caso que el usuario no teclee enteros, se gestionará la excepción ValueError correspondiente sin salirse del bucle.
  - Modificar las funciones **run_concatenate_numbers_ascending** y **run_summatory** para que pidan y reciban los dos valores, simplificando el código de ambas.

Parte 4.2 Cálculo de Máximo Común Divisor (MCD)

- El MCD de dos números enteros se calcula mediante el algoritmo de Euclides, que en 'lenguaje moderno' se describe como (Wikipedia):

```
Sean dos enteros a y b (con a >= b). Para números negativos basta con coger el valor
absoluto. Llamemos residuo al valor de a.
Mientras el residuo sea distinto de 0
    Actualizar residuo con el resto de la división entre a y b
    a pasa a valer b, mientras que b pasa a valer el residuo.
El valor de a será el MCD.
```

```
Ejemplo:
55 20 --> 15
20 15 --> 5
15 5 --> 0

5 es el MCD
```

Escribir la función **calculate_MCD** en el módulo **calculator_logic.py**. Casos de uso para doctest: MCD(4,6) es 2, MCD(15,45) es 15, MCD(12,8) es 4, MCD(150,340) es 10.

Parte 4.3 Obtener entrada de usuario validada

- Escribir la función **ask_integer** (en el módulo **calculator_user_io.py**) que pida al usuario un entero, retornándolo. La función debe ser segura en cuanto a errores del usuario.

Parte 4.4 Extender el menú

- Escribir la función **run_MCD** (en el módulo calculator_main.py) que pide dos enteros, calcula y muestra por pantalla el MCD.
- Extender, en éste mismo módulo, el menú –y también las globales – con esta nueva opción de cálculo (funciones **show_menu**, **ask_menu_option**, **handle_option**).

Parte 5: Depuración de código

Parte 5: Depuración de código

- La depuración de código cobra mayor interés si cabe al utilizar bucles de iteración.
  - A veces resulta la única herramienta que nos permite encontrar los errores en el código.
  - Un consejo: paciencia, que es la madre de la ciencia.
- Para ejemplarizar esta virtud de la depuración se plantea un ejercicio partiendo de un código con errores.

Parte 5: Depuración de código

- Descargue el archivo **part4_new_code.py**. Contiene una serie de funciones para determinar si un año es mágico o no.
  - No está documentado de ninguna manera.
  - Contiene errores.
  - La siguiente página describe el objetivo del código.
- Se deben corregir los errores mediante la depuración.

Parte 5: Depuración de código

- Las funciones **is_prime**, **is_divided_by_3**, e **is_divided_by_7** retornan True o False en función de si su argumento cumple ser primo, ser divisible por 3 o divisible por 7.
- La función **is_leap_year** retorna True si el año es divisible por 4 y no por 100, o si es divisible por 400. En otro caso, retorna False.
- La función **accumulate_years** suma los años de una forma particular… el siguiente orden se debe mantener. Si es primo, añade el año. Si es divisible por 7, suma el año dividido por 7. Si es divisible por 3, suma el año dividido por 3. Si es año bisiesto, suma el resto de dividir el año dividido entre 10. En el resto de los casos, suma el año dividido entre 10.
- Finalmente, la función **is_magic_year** recibe un año, retornando True si el acumulado de los años no es divisible por el año en cuestión.
- El **main** comprueba los primeros 10 años de esta era: son mágicos los años 6, 9 y 10; el resto no lo son.

Parte 6: Ejercicios extra

Parte 6: Ejercicios extra

- Desarrollo del coseno en series de Taylor.
  - Descargue el archivo **part5_cosine.py**, donde se tiene dos funciones: calculate_factorial y calculate_cosine. Están documentadas con ***doctest***.
  - Implementan el desarrollo en series de Taylor del coseno:

$$\cos(x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n)!} x^{2n}$$

Se pide, utilizando la herramienta de depuración, encontrar y solucionar los errores en el código.

Parte 6: Ejercicios extra

- Extensiones a la calculadora.

Añada las siguientes funciones a la calculadora.

- Desarrollo de Taylor del seno **calculate_sin**: recibe el valor del ángulo en sexagesimal x y el número de términos de la serie N. Calcula y retorna el seno por medio de la siguiente fórmula:

$$\text{sen}(x) = \sum_{n=0}^{N} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

  - Requiere de la función **calculate_factorial** corregida del ejercicio de depuración anterior.
  - Es necesaria la función **ask_angle** en el módulo calculator_user_io.py para pedir un ángulo en grados sexagesimales (en el intervalo [-360.0, 360.0].
  - Cree la función **run_calculate_sin** en el módulo calculator_main.py para pedir el ángulo y el número de términos, llamar a la función para cálculo de seno, y mostrar el resultado.
  - Extienda las capacidades de las funciones de menús para disponer la llamada a calcular el seno de un ángulo.
  - Casos de prueba: sin(0, 10) es 0.0, sin(90, 20) es 1.0000000000000002, sin(45, 10) es 0.7071067811865475

Parte 6: Ejercicios extra

- Extensiones a la calculadora.

Añada las siguientes funciones a la calculadora.

- Mínimo común múltiplo (**calculate_MCM**): recibe dos enteros a y b, operando para garantizar que a > b. Se busca el menor múltiplo de a tal que sea también divisible por b. Por lo tanto, se prueba a, 2a, 3a, … hasta encontrar aquél que sea múltiplo de b.
  - Cree la función **run_calculate_MCM** en el módulo calculator_main.py para pedir los dos valores enteros, llamar a la función para cálculo de MCM, y mostrar el resultado.
  - Extienda las capacidades de las funciones de menús para disponer la llamada a calcular el MCM de dos enteros.
  - Casos de prueba: MCM(4, 6) es 12, MCM(15, 45) es 45, MCM(12, 8) es 24, MCM(150, 340) es 5100.

Parte 6: Ejercicios extra

- Extensiones a la calculadora.

Añada las siguientes funciones a la calculadora.

- Desarrollo de Taylor del logaritmo neperiano de 1+x **calculate_ln_x_plus_1**: recibe el valor de x (|x| < 1) y el número de términos de la serie N. Calcula y retorna el logaritmo por medio de la siguiente fórmula:

$$\ln(1+x) = \sum_{n=1}^{N} \frac{(-1)^{n+1}}{n!} x^n, \qquad \forall x/|x| < 1$$

  - Requiere de la función **calculate_factorial**.
  - Es necesaria la función **ask_float** en el módulo calculator_user_io.py para pedir un valor de x adecuado.
  - Cree la función **run_ln_x_plus_1** en el módulo calculator_main.py para pedir el ángulo y el número de términos, llamar a la función para cálculo de ln(1+x), y mostrar el resultado.
  - Extienda las capacidades de las funciones de menús para disponer la llamada a calcular el logaritmo del valor de 1 + x.
  - Casos de prueba: ln(1+0.5, 10) es 0.39346934027562475, ln(1-0.75, 5) es -1.1167236328125, ln(1+1.5, 10) genera una excepción.