Programación en Python

Practica 4. Funciones y testing

Universidad Politécnica de Valencia

2021-2022

Todas las funciones Python que tienes que escribir deben ser testeado usando el módulo pytest para comprobar su correcto funcionamiento.

Ejercicio 1. Escribir una función `es_digito` que recibe como parámetro un carácter y devuelve un booleano. La función devolverá `True` cuando el carácter es un dígito del 0 al 9, si no devolverá `False`).

Puedes usar los siguientes pytests para testear tu función:

```
@pytest.mark.parametrize("testcase, entrada, salida_esperada",[
    (1, '0 ', True),     # smallest digit
    (2, '9 ', True),     # largest digit
    (3, '5 ', True),     # other digit
    (4, '12', False),    # it is not a digit between 0 and 9
    (5, '-2', False),    # negative digit
    (6, 'hello ', False),# string
    ]
)

def test_es_digito(testcase, entrada, salida_esperada):
    assert es_digito(entrada) == salida_esperada, "caso {0}".format(testcase)
```

Ejercicio 2. Escribir un programa que lea un carácter desde teclado y determine con la función `es_digito` del ejercicio anterior si se trata de uno de los dígitos del 0 al 9. Escribir el programa de tal forma que sirva para leer varios caracteres diferentes desde teclado hasta el usuario escribe la palabra fin.

ejemlos de test ejecuciones

```
>>> %Run
  Escribe un caracter o 'fin' para terminar: 4
  4 es un digito de 0 a 9
  Escribe un caracter o 'fin' para terminar: -1
  -1 no es un digito de 0 a 9
  Escribe un caracter o 'fin' para terminar: 56
  56 no es un digito de 0 a 9
  Escribe un caracter o 'fin' para terminar: dfg
  dfg no es un digito de 0 a 9
  Escribe un caracter o 'fin' para terminar: 0
  0 es un digito de 0 a 9
  Escribe un caracter o 'fin' para terminar: fin
>>>
```

Ejercicio 3. Escribir una función `es_primo` que recibe un valor y devuelve un booleano. La función devolverá `True` cuando el valor es un número natural que es un primo, si no devolverá `False`).

Puedes usar los siguientes pytest para testear tu función.

```python
@pytest.mark.parametrize("testcase, entrada, salida_esperada",[
    (1, 0, False),
    (2, 1, False),
    (3, 2, True),
    (4, 25, False),
    (5, 23, True),
    (6, 97, True),
    (7, "97", False),
    (8, 97.0, False),
    (9, 3, True),
    (10,-3, False),
    ]
)

def test_es_primo(testcase, entrada, salida_esperada):
    assert es_primo(entrada) == salida_esperada, "caso {0}".format(testcase)
```

Ejercicio 4. Escribir un programa que lea un numero natural desde teclado y determine con la función `es_primo` del ejercicio anterior si se trata de numero primo. Escribir el programa de tal forma que sirva para leer varios números diferentes desde teclado hasta el usuario escribe la palabra fin. Cuando el usuario teclea algo que no es numero natural, hay que indicárselo y volver a darle la oportunidad de escribir un numero.

Puedes testear tu programa con los siguientes tests:

ejemlos de test ejecuciones

```
>>> %Run
  Escribe un numero entero, o 'fin' para terminar: fin
>>> %Run
  Escribe un numero entero, o 'fin' para terminar: 4
  4 no es primo
  Escribe un numero entero, o 'fin' para terminar: 97
  97 es primo
  Escribe un numero entero, o 'fin' para terminar: -4
  -4 no es primo
  Escribe un numero entero, o 'fin' para terminar: -97
  -97 no es primo
  Escribe un numero entero, o 'fin' para terminar: hello?
  solo numeros enteros o 'fin' para terminar!
  Escribe un numero entero, o 'fin' para terminar: x
  solo numeros enteros o 'fin' para terminar!
  Escribe un numero entero, o 'fin' para terminar: fin
>>>
```

Ejercicio 5. Escribir una función `minu_to_mayu` que recibe como parámetro una `letra` en minúscula y devuelva ese mismo carácter en mayúscula. Para ello debes emplear las funciones **`chr`** y **`ord`**. Si la `letra` no pertenece al abecedario español de minúsculas hay que devolver la misma `letra`.

Recuerda que:

- **`chr`**: devuelve el carácter correspondiente a un entero dentro de la tabla ASCII. La tabla de ASCII esta basado en el abecedario internacional con los 26 letras. La primera letra `'a'` esta en la posición 97, por eso **`chr`**`(97)` devuelve `'a'` y 25 caracteres después (i.e. 97+25=122) esta la **`chr`**`(122)` que devuelve la `'z'`. Recuerda que la `'ñ'` que esta en el abecedario del español este en la posición 241.
- **`ord`**:devuelve el valor entero de un carácter en la tabla ASCII. Las letras mayúsculas del abecedario internacional van del **`ord`**`("A")` que devuelve `65` al **`ord`**`("Z")` que devuelve `90`.

La ultima vez que te regalamos los pytest que puedes usar para testear tu función. A partir de ahora lo tendrás que hacer tu.

```
@pytest.mark.parametrize("testcase, entrada, salidas_esperadas",[
    (1, 'a', 'A'),
    (2, 'z', 'Z'),
    (3, 'ñ', '  '),
    (4, '*', '*'),
    (5, 'Q', 'Q'),
    (6, '%', '%'),
    (7, '`', '`'),  #carácter justo antes de la 'a' in tabla ASCII
    (8, '{', '{'),  #carácter justo después de la 'z' in tabla ASCII
    (9, '|', '|'),
    (10, 10, 10),
    (11, [], []),
    ]
)
def test_minu_to_mayu(testcase, entrada, salida_esperada):
    assert minu_to_mayu(entrada) == salida_esperada, "caso {0}".format(testcase)
```

Ejercicio 6. Escribe una función `factorial` que dado un numero entero positivo `n` calcula el factorial. Recuerda que el factorial de `n` se define como el producto de todos los números enteros positivos desde 1 (es decir, los números naturales) hasta `n`. Por ejemplo:

$$5! = 5\ x\ 4\ x\ 3\ x\ 2\ x\ 1$$

Escribe pytests para testear tu implementación. Recuerda que $0! = 1$ y $1! = 1$.

Ejercicio 7. La función exponencial e^x puede ser definida como una serie de potencias.

Desarrollo en Serie de Taylor:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

Por ejemplo, cuando $x = 1$: $e = \sum_{n=0}^{\infty} \frac{1}{n!}$

entonces:

$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \ldots$

Escribe una función `my_exp` que recibe el valor de x como un parámetro y utiliza una iteración para realizar el cálculo del término n-ésimo de la serie y sumando estos términos, obtener una aproximación al valor de e^x. Puedes usar tu función `factorial` del ejercicio anterior.

Puedes usar `math.exp` como resultado esperado en tus casos parametrizados de pytest (es decir en tu lista de `@pytest.mark.parametrize`). Recuerda que hay que tener en cuenta que comparar flotantes para la igualdad tiene problemas de redondeo y precisión. Podemos comparar que la diferencia entre lo que sale de nuestra función y el math.exp es menor a por ejemplo 10^{-7}

```
def test_my_exp(tc, entrada, salida_esperada):
    assert abs(my_exp(entrada) - salida_esperada)<10**-7, "caso {0}".format(tc)
```

Ejercicio 8. Escribir una función para calcular el máximo común divisor (`mcd`) de sus dos parametros x e y que son números enteros y mayores que 0. Utiliza el algoritmo de Euclides. Sean x e y los valores originales de las variables a y b, el algoritmo dice:

Mientras a y b no sean iguales, cambiar el mayor de los dos por la diferencia entre el mayor y el menor. Cuando tengan el mismo valor, ese es el `mcd` de x e y.

Propiedades en las que se basa el algoritmo de Euclides:

- Al finalizar cada iteración: `mcd`(x, y)= `mcd`(a, b)
- Esta propiedad es una consecuencia de la propiedad matemática:
  - `mcd`$(a, b) =$ `mcd`$(a - b, b)$ cuando $a > b$
  - `mcd`$(a, b) =$ `mcd`$(a, b - a)$ cuando $b > a$
- Cuando finalmente $a = b$, `mcd`$(x, y) =$ `mcd`(a, b)

Escribe pytests para testear tu implementación. Si leemos bien la descripción del ejercicio vemos que la función no tiene porque funcionar para números que no son mayor a 0.

Ejercicio 9. Escribir una funcion `mcd_de3` que calcule el máximo común divisor de más de 2 números. Para ello utiliza la función `mcd` (máximo común divisor de 2 números) .

Escribe pytests para testear tu implementación.

Ejercicio 10. Escribir una función que dado un número entero $N1$, devuelva otro número entero $N2$ que sea el resultado de eliminar la primera y última cifra de $N1$. Nota: Si $N1$ tiene 2 cifras o una sola, entonces $N2$ deberá ser 0. Ejemplos de casos de test que puedes automatizar con pytest son:

testcase número	input ($N1$)	output esperado ($N2$)
1	42635	263
2	23	0
3	5	0
4	0	0
5	-3456	-45

Ejercicio 11. Escribir una función que recibe como parámetro un número N y genere un string con los números:

$$1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, ..., 1, 2, 3, ..., N$$

Ejemplos de casos de test que puedes automatizar con pytest son:

testcase número	input (N)	output esperado
1	4	`"1, 1, 2, 1, 2, 3, 1, 2, 3, 4"`
2	1	`"1"`
3	0	`""`
4	-3	`"-1, -1, -2, -1, -2, -3"`

Ejercicio 12. Escribe una función que recibe como parámetro una contraseña y determina la complejidad de una contraseña determinada según estas reglas:

- Una contraseña muy débil contiene solo números y tiene menos de ocho caracteres.
- Una contraseña débil contiene solo letras y tiene menos de ocho caracteres.
- Una contraseña segura contiene letras y al menos un número y tiene al menos ocho caracteres.
- Una contraseña muy segura contiene letras, números y caracteres especiales y tiene al menos ocho caracteres.
- Las contraseñas que no son débiles ni seguras son normales.

Recuerda que en clase de teoría hemos visto las siguientes funciones predefinidas en Python:

- `isdigit`, para chequear si un string tiene digitos.

- `isalpha` para chequear si un string solo contiene caracteres del alfabeto.

Para probar bien tu función, ¿cuantos casos de test has ejecutado?, ¿has pensado tanto en minúsculas como mayúsculas?