# Tecnología Digital 1: Introducción a la Programación - TP1

**Autores:** Ferrari, Sofia Oriana
García Vence, Gonzalo
Milde, Manuel

---

## Justificación de la elección de los casos de test

**Función es_divisor:**

En esta función decidimos tomar los siguientes casos de testeo:

- **(es_divisor(5, 10), True)** Testeo más general, con números comunes.
- **(es_divisor (1, 1), True)** Caso donde el divisor y dividendo sea el mismo, donde quizás al ingresar el mismo valor podría generar algun conflicto.
- **(es_divisor (1, 5), True)** En esta situación decidimos testear con el 1 (divisor de todos los numeros) para así probar que funcione.
- **(es_divisor (15, 5), False)** Probar con números con orden alterado, dejando al divisor cambiado de posición, verificar que no de verdadero.

**Función suma_divisores_positivos:**

En la función que suma los divisores tomamos los siguientes casos:

- **(suma_divisores_positivos(12), 28)** La elección es simplemente por ser el numero abundante más cercano a 0.
- **(suma_divisores_positivos(1), 1)** Decidimos elegir este ejemplo por el simple hecho de que 1 solo tiene como divisor a si mismo.
- **(suma_divisores_positivos(13), 14)** Este ejemplo lo escogimos por ser uno de los números primos más conocidos, y además por estar a un solo valor de 12 (primer numero abundante contando desde el 0)
- **(suma_divisores_positivos(179), 180)** En este peculiar caso, la elección se justifica con ser un número primo mayor a 100 y a simple vista no poder detectar si es primo o no.

**Función es_abundante:**

- **(es_abundante(12), True)** Prueba con el abundante más cercano a 0.
- **(es_abundante(1009), False)** Como en la función de arriba, decidí tomar un número primo de 4 cifras para realizar la prueba (por la dificultad de evaluar a simple vista si es o no abundante).
- **(es_abundante(13), False)** Prueba con un número primo (ya que esa condición deja imposible que sea abundante)
- **(es_abundante(32), False)** Prueba un caso False con un número que no sea primo y a simple vista podria parecer abundante (se queda a 1 de ser abundante)

**Función suma_abundantes:**

- **(suma_abundantes(1, 12) 12)** Prueba con abundante más cercano a 0.

- **(suma_abundantes(1, 11) 0)** Prueba con 2 valores que no poseen números abundantes entre ellos.
- **(suma_abundantes(12, 12) 12)** Con este testeo verificamos el caso de poner el mismo digito y que sea un abundante (lo toma en cuenta).
- **(suma_abundantes(12, 18) 30)** Testeo con un intervalo con 2 abundantes como limites, donde vemos que funciona correctamente ya que toma a ambos en la suma.

**Función abundante_mas_cercano:**

- **(abundante_mas_cercano(1), 12)** Verificamos que el abundante más cercano a 1 tiene que ser 12.
- **(abundante_mas_cercano(15), 18)** Verificamos que el abundante más cercano a 15 tiene que ser 18. Esto es debido al requerimiento de ante una misma distancia escoger el mayor.
- **(abundante_mas_cercano(12), 12)** Con esto, testeamos que si ingresamos un abundante tome ese número y no busque otro.
- EN ESTE CASO SE USO assertNotEqual**(abundante_mas_cercano(22), 20)** Nos aseguramos que la condicion de tomar al mayor funciona con otro ejemplo similar, pero utilizando assertNotEqual.

## Justificación de por qué los programas hacen lo esperado

**Función es_divisor:**

La función hace lo esperado ya que se ingresan dos valores [n,m] mayores que cero y los evalua con la operación “resto” (%). Con esta, nos aseguramos de que si el resultado es 0, el numero ’n’ es divisor de ‘m’. Para ello, utilizamos un *if* que compara el resultado con “0”, de ser iguales entra en el *if* y el ‘vr’ toma valores TRUE, caso contrario, tomas valores FALSE.

**Función suma_divisores_positivos:**

Terminación:

- La variable i empieza valiendo 1
- Por cada iteración del ciclo, i se incrementa en 1
- Por requerimiento de la función, sabemos que n > 0 y además el valor del mismo no varia
- Luego, es inevitable que i llegue a valer n en algún momento
- Entonces, la condición $i \leq n$ pasa a ser falsa, y el ciclo finaliza

Correctitud:

- i se mueve entre 1 y n inclusive, es decir, $1 \leq i \leq n$;
- La variable vr vale la suma de los numeros i divisores de n

**Función es_abundante:**

La siguiente función empieza definiendo una variable “suma_divisores” asignándole el valor de la función suma_divisores_positivos del valor ingresado [n]. Luego, encontramos un *if* donde se evalúa si la variable “suma_divisores” es mayor al valor ingresado multiplicado por 2. En caso de cumplir la condición, se entra al *if* y se le asigna un valor de retorno TRUE, de lo contrario, el valor de retorno será FALSE.

**Función suma_abundantes:**

Terminación:

- La variable i empieza valiendo n
- Por cada iteración del ciclo, i se incrementa en 1
- Por requerimiento de la función, sabemos que $0 < n \leq m$ y además el valor del mismo no varia
- Luego, es inevitable que i llegue a valer m en algún momento
- Entonces, la condición $i \leq m$ pasa a ser falsa, y el ciclo finaliza

Correctitud:

- i se mueve entre n y m inclusive, es decir, $n \leq i \leq m$;
- La variable vr vale la suma de los numeros i abundantes entre n y m

**Función abundante_mas_cercano:**

-Función auxiliar abundantemayor:

Terminación:

- La variable j empieza valiendo k
- Por cada iteración del ciclo, j se incrementa en 1
- Por requerimiento de la función, sabemos que $k > 0$ y además el valor del mismo no varia
- Luego, es inevitable que j en algún momento sea abundante
- Entonces, la condición *es_abundante(j) == False* pasa a ser falsa, y el ciclo finaliza

Correctitud:

- j se mueve entre k y el abundante mayor mas cercano de k (K), es decir, $k \leq j \leq K$;
- La variable vr vale el numero abundante más cercano mayor a n

-Función auxiliar abundantemenor:

Terminación:

- La variable j empieza valiendo k
- Por cada iteración del ciclo, j se decrementa en 1
- Por requerimiento de la función, sabemos que $k > 0$ y además el valor del mismo no varia
- Luego, es inevitable que j en algún momento llegue a valer cero o a ser abundante
- Entonces, la guarda (*j > 0* ***and*** *es_abundante(j) == False)* pasa a ser falsa, y el ciclo finaliza

Correctitud:

- j se mueve entre k y cero, es decir, $0 < j \leq k$ ;

- La variable vr vale el numero abundante más cercano menor a n

Esta última función empieza analizando si el valor ingresado es un abundante. En caso verdadero, ese va a ser el valor de retorno. En otro caso, se establecen las variables “abundante_mayor” y “abundante_menor” como funciones auxiliares, para así encontrar dos abundantes, uno mayor y otro menor al valor ingresado.

Ya teniendo ambos valores, mediante un *if* se evalúa cual de ambos esta más cerca del número ingresado, y en caso de estar a la misma distancia, se elige el mayor.

## Aclaraciones adicionales (si las hubiera)

- Variable **vr**: utilizamos ‘vr’ como nombre de variable en todas las funciones, haciendo referencia a la abreviación de “valor de retorno”.
- Funciones auxiliares: agregamos las funciones *abundantemenor y abundantemayor* para facilitar la comprensión del código.