Introducción al Pensamiento Computacional

Clase 17: Funciones, parte 1 🧩

Actividad 1-a

Verificación de CUIL

Actividad 1-a | Verificación de CUIL

Una empresa de desarrollo de sistemas bancarios firmó un convenio con un cine para que sus empleados tengan un descuento presentando su CUIL. En esta parte, nos concentraremos en verificar que el CUIL sea válido.

Escribir **precondiciones, estrategia y programa** que le pida al usuario (el cajero del cine) un CUIL en formato `"XY-ABCDEFGH-Z"` e imprima por consola un mensaje indicando si el CUIL es válido. Para verificar la validez de un CUIL se debe seguir el siguiente procedimiento:

1. `cuenta = X*5 + Y*4 + A*3 + B*2 + C*7 + D*6 + E*5 + F*4 + G*3 + H*2`
2. `verificador = 11 - (cuenta % 11)`
3. Si `Z == verificador` entonces el CUIL es válido, sino no lo es.

```
CUIL: 20-12345678-6
El CUIL es válido
```

Con las herramientas que tenemos hasta este momento podemos resolver “fácilmente” el problema: tendríamos que extraer cada dígito del CUIL ingresado por el usuario por consola, realizar las cuentas indicadas en el procedimiento e imprimir un mensaje u otro si el CUIL es válido o no.

Actividad 1-a | Verificación de CUIL

```
'''
Precondiciones:
    • El formato del CUIL es correcto

Estrategia:
    1. Pedirle al usuario el CUIL
    2. Calcular el dígito verificador
    3. Si el dígito verificador es correcto, indicar que el CUIL es válido, sino no.
'''

cuil = input("CUIL: ")

cuenta = int(cuil[0])*5 + int(cuil[1])*4 + int(cuil[3])*3 + int(cuil[4])*2 + int(cuil[5])*7 + \
         int(cuil[6])*6 + int(cuil[7])*5 + int(cuil[8])*4 + int(cuil[9])*3 + int(cuil[10])*2
verificador = 11 - (cuenta % 11)

if (verificador == int(cuil[12])):
    print(f"El CUIL {cuil} es válido")
else:
    print(f"El CUIL {cuil} NO es válido")
```

Actividad 1-b

Verificación de CUIL

Actividad 1-b | Verificación de CUIL

Una empresa de desarrollo de sistemas bancarios firmó un convenio con un cine para que sus empleados tengan un descuento presentándose de a dos con sus CUILs. En esta parte, el cajero del cine debe verificar 2 CUILs.

Modificar **precondiciones, estrategia y programa anterior** para que le pida al usuario (el cajero del cine) dos CUILs en formato `"XY-ABCDEFGH-Z"` e imprima por consola un mensaje indicando si ambos CUILs son válidos o alguno de ellos no lo es.

```
CUIL 1: 20-12345678-6
CUIL 2: 20-23456789-7
Los CUILs son válidos
```

Actividad 1-b | Verificación de CUIL

```
'''
Precondiciones:
    • El formato de los dos CUILs es correcto

Estrategia:
    1. Pedirle al usuario los dos CUILs
    2. Calcular los dígitos verificadores de ambos CUILs
    3. Si los dígitos verificadores son correctos, indicar que los CUILs son válidos, sino no
'''

cuil1 = input("CUIL 1: ")
cuil2 = input("CUIL 2: ")

cuenta = int(cuil1[0])*5 + int(cuil1[1])*4 + int(cuil1[3])*3 + int(cuil1[4])*2 + int(cuil1[5])*7 + \
         int(cuil1[6])*6 + int(cuil1[7])*5 + int(cuil1[8])*4 + int(cuil1[9])*3 + int(cuil1[10])*2
verificador1 = 11 - (cuenta % 11)

cuenta = int(cuil2[0])*5 + int(cuil2[1])*4 + int(cuil2[3])*3 + int(cuil2[4])*2 + int(cuil2[5])*7 + \
         int(cuil2[6])*6 + int(cuil2[7])*5 + int(cuil2[8])*4 + int(cuil2[9])*3 + int(cuil2[10])*2
verificador2 = 11 - (cuenta % 11)

if (verificador1 == int(cuil1[12])) and (verificador2 == int(cuil2[12])):
    print("Los CUILs son válidos")
else:
    print("Alguno de los CUILs NO es válido")
```

¿Identifican algún problema en esta solución?

Actividad 1-b | Verificación de CUIL

```
'''
Precondiciones:
    • El formato de los dos CUILs es correcto

Estrategia:
    1. Pedirle al usuario los dos CUILs
    2. Calcular los dígitos verificadores de ambos CUILs
    3. Si los dígitos verificadores son correctos, indicar que los CUILs son válidos, sino no
'''

cuil1 = input("CUIL 1: ")
cuil2 = input("CUIL 2: ")

cuenta = int(cuil1[0])*5 + int(cuil1[1])*4 + int(cuil1[3])*3 + int(cuil1[4])*2 + int(cuil1[5])*7 + \
         int(cuil1[6])*6 + int(cuil1[7])*5 + int(cuil1[8])*4 + int(cuil1[9])*3 + int(cuil1[10])*2
verificador1 = 11 - (cuenta % 11)

cuenta = int(cuil2[0])*5 + int(cuil2[1])*4 + int(cuil2[3])*3 + int(cuil2[4])*2 + int(cuil2[5])*7 + \
         int(cuil2[6])*6 + int(cuil2[7])*5 + int(cuil2[8])*4 + int(cuil2[9])*3 + int(cuil2[10])*2
verificador2 = 11 - (cuenta % 11)

if (verificador1 == int(cuil1[12])) and (verificador2 == int(cuil2[12])):
    print("Los CUILs son válidos")
else:
    print("Alguno de los CUILs NO es válido")
```

Instrucciones repetidas

Instrucciones repetidas

Funciones para reutilizar soluciones

Hay veces en las que hay subproblemas que aparecen varias veces en diferentes partes del problema principal. Si detectamos esto, podemos definir una **función**, que nos permite resolver el problema una sola vez y luego reutilizar la solución (sin necesidad de copiarla) las veces que sea necesario.

```
def <función>(<parámetro1>, ..., <parámetroN>):
    <instrucciones>
    return <resultado>
```

Ejemplo:

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen
```

Cuestiones de **sintaxis**:

- Después de `def` y de `return` hay un espacio en blanco
- El nombre de la función y de los parámetros debe seguir las mismas convenciones de las variables
- Al final de la línea del `def` debe haber dos puntos
- Después del nombre de la función y antes de los dos puntos deben ir paréntesis
- Los parámetros se separan con comas
- Todas las instrucciones del cuerpo deben tener la misma indentación

¿Cómo se utilizan las funciones?

1. Debemos definir la función, es decir, abstraer la acción que representa y escribir la **firma** (nombre y **parámetros**) y el **cuerpo** (instrucciones que ejecuta) de la función. Para esto:
   a. Hay que elegir un nombre representativo para la función
   b. Hay que elegir nombres representativos para los parámetros
2. Debemos llamar o invocar la función pasándole los **argumentos** requeridos, siguiendo (y respetando) la firma de la función.

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```

¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```

programa principal

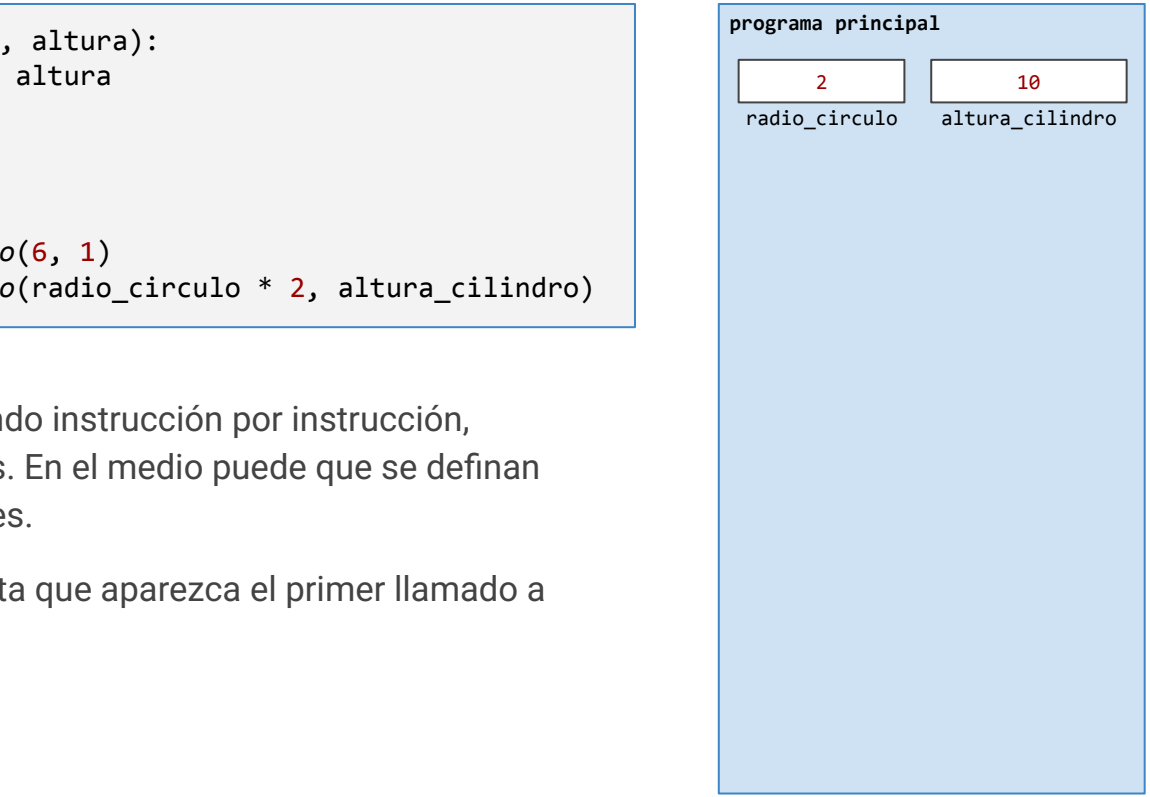
1. El programa comienza ejecutándose desde la primera instrucción que no sea una definición de una función. En lo que resta del curso, todas las funciones serán definidas antes que el **programa principal**.

   Es en este momento que el programa reserva espacio en la memoria de la computadora para trabajar.

¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```



2. El programa principal se irá ejecutando instrucción por instrucción, secuencialmente, como ya sabemos. En el medio puede que se definan variables y realicen otras operaciones.

   Esto se repetirá de esta manera hasta que aparezca el primer llamado a alguna función.

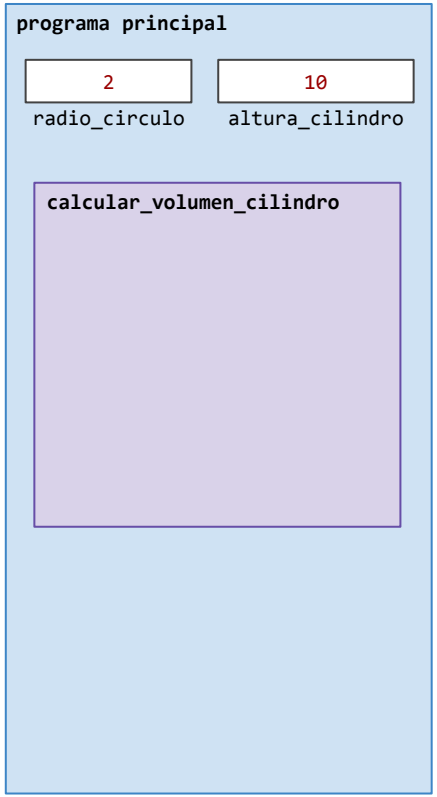
¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```

3. En un llamado a función, el programa se quedará esperando a que la función ejecute sus instrucciones y devuelva un resultado, si corresponde.

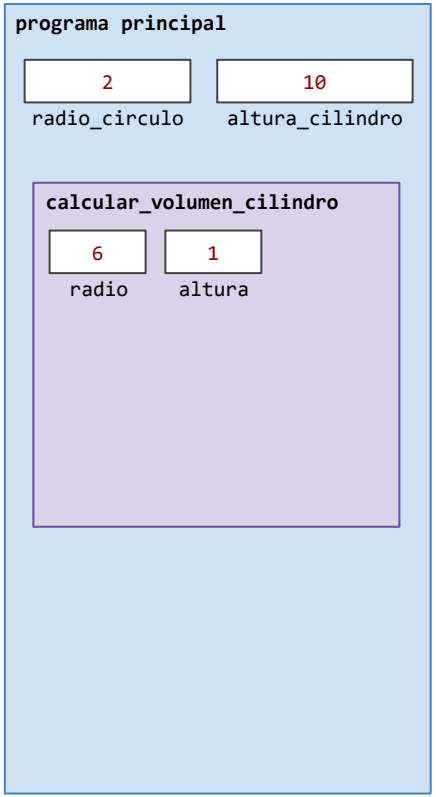
   Es en este momento que el programa reserva un espacio de memoria para la función, dentro del espacio de memoria del programa principal, y está listo para comenzar a ejecutar la función.



¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```



4. En un llamado a función, antes que cualquier cosa se le asignan los valores a los **parámetros** de la función, es decir, los datos que la función necesita que los valores le sean provistos desde donde se invocó la función.

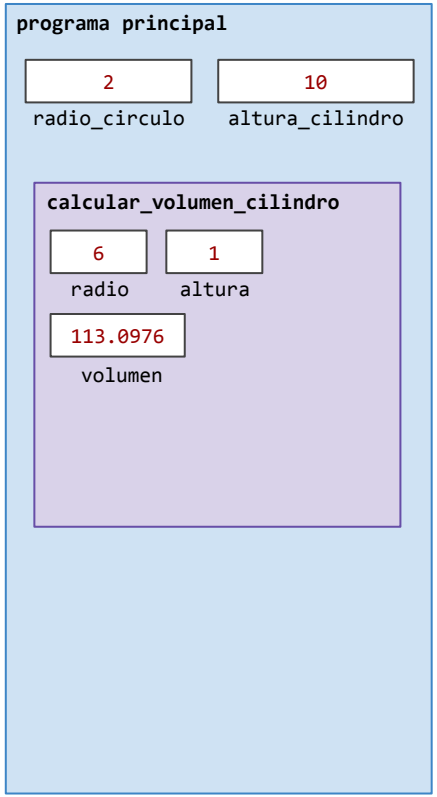
   En este sentido, los parámetros no son más que variables donde su valor inicial (diremos sus **argumentos**) viene de fuera de la función.

   Se debe considerar que el pasaje de parámetros es **posicional**, es decir, el valor del 1° argumento irá al 1° parámetro, el valor del 2° argumento al 2° parámetro, etc.

¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```



5. La función se irá ejecutando instrucción por instrucción, secuencialmente, como si fuera un programa común y corriente. En el medio puede que se definan variables y realicen otras operaciones.

   Esto se repetirá de esta manera hasta que se alcance un `return`.

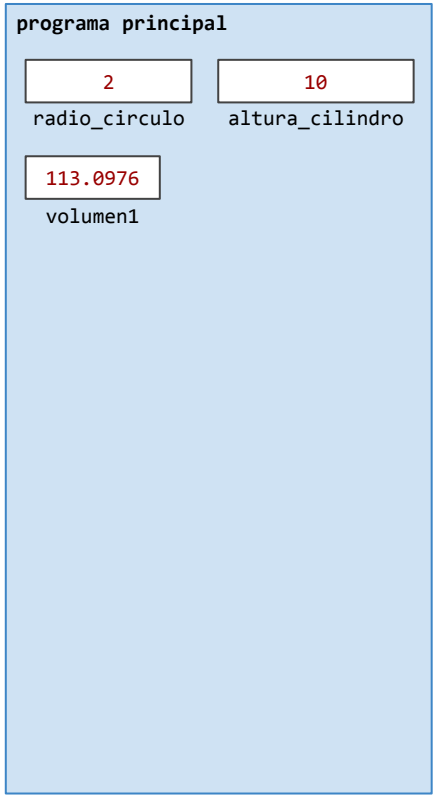
¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```

6. Cuando la ejecución de una función llega a un `return`, se devuelve el control al programa que llamó a la función junto con el resultado (si lo hubiera).

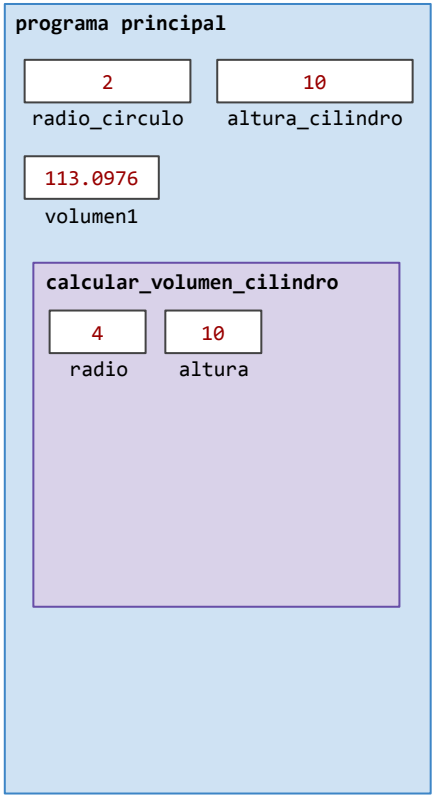
   El espacio de memoria reservado para la función desaparece por completo en este momento.



¿Cómo funcionan las funciones?

```
def calcular_volumen_cilindro(radio, altura):
    volumen = 3.1416 * (radio**2) * altura
    return volumen

radio_circulo = 2
altura_cilindro = 10
volumen1 = calcular_volumen_cilindro(6, 1)
volumen2 = calcular_volumen_cilindro(radio_circulo * 2, altura_cilindro)
```



7. En el caso del llamado a función con argumentos que son variables y/o expresiones primero se debe conocer el valor de ambos tipos de datos y luego se pasan por parámetro a la función. Luego el procedimiento es el mismo.

   Notar que no es necesario que las variables y los parámetros tengan el mismo nombre porque, como vimos, lo que se hace es enviar por parámetro el **valor** de los argumentos.

Actividad 1-b | Verificación de CUIL

```
'''
Precondiciones:
    • El formato de los dos CUILs es correcto

Estrategia:
    1. Pedirle al usuario los dos CUILs
    2. Calcular los dígitos verificadores de ambos CUILs
    3. Si los dígitos verificadores son correctos, indicar que los CUILs son válidos, sino no
'''

cuil1 = input("CUIL 1: ")
cuil2 = input("CUIL 2: ")

cuenta = int(cuil1[0])*5 + int(cuil1[1])*4 + int(cuil1[3])*3 + int(cuil1[4])*2 + int(cuil1[5])*7 + \
         int(cuil1[6])*6 + int(cuil1[7])*5 + int(cuil1[8])*4 + int(cuil1[9])*3 + int(cuil1[10])*2
verificador1 = 11 - (cuenta % 11)

cuenta = int(cuil2[0])*5 + int(cuil2[1])*4 + int(cuil2[3])*3 + int(cuil2[4])*2 + int(cuil2[5])*7 + \
         int(cuil2[6])*6 + int(cuil2[7])*5 + int(cuil2[8])*4 + int(cuil2[9])*3 + int(cuil2[10])*2
verificador2 = 11 - (cuenta % 11)

if (verificador1 == int(cuil1[12])) and (verificador2 == int(cuil2[12])):
    print("Los CUILs son válidos")
else:
    print("Alguno de los CUILs NO es válido")
```

¿Cómo podemos mejorar este programa con **funciones**?

¿Qué función conviene definir para reutilizar código?

Actividad 1-b | Verificación de CUIL

```
'''
Precondiciones:
    • El formato de los dos CUILs es correcto

Estrategia:
    1. Pedirle al usuario los dos CUILs
    2. Calcular los dígitos verificadores de ambos CUILs
    3. Si los dígitos verificadores son correctos, indicar que los CUILs son válidos, sino no
'''

def es_cuil_valido(cuil):
    cuenta = int(cuil[0])*5 + int(cuil[1])*4 + int(cuil[3])*3 + int(cuil[4])*2 + int(cuil[5])*7 + \
             int(cuil[6])*6 + int(cuil[7])*5 + int(cuil[8])*4 + int(cuil[9])*3 + int(cuil[10])*2
    verificador = 11 - (cuenta % 11)
    return (verificador == int(cuil[12]))

cuil1 = input("CUIL 1: ")
cuil2 = input("CUIL 2: ")

if (es_cuil_valido(cuil1)) and (es_cuil_valido(cuil2)):
    print("Los CUILs son válidos")
else:
    print("Alguno de los CUILs NO es válido")
```

Actividad 1-b | Verificación de CUIL

```
'''
Precondiciones:
    • El formato de los dos CUILs es correcto

Estrategia:
    1. Pedirle al usuario los dos CUILs
    2. Calcular los dígitos verificadores de ambos CUILs
    3. Si los dígitos verificadores son correctos, indicar que los CUILs son válidos, sino no
'''

def es_cuil_valido(cuil):
    cuenta = int(cuil[0])*5 + int(cuil[1])*4 + int(cuil[3])*3 + int(cuil[4])*2 + int(cuil[5])*7 + \
             int(cuil[6])*6 + int(cuil[7])*5 + int(cuil[8])*4 + int(cuil[9])*3 + int(cuil[10])*2
    verificador = 11 - (cuenta % 11)
    return (verificador == int(cuil[12]))

cuil1 = input("CUIL 1: ")
cuil2 = input("CUIL 2: ")

if (es_cuil_valido(cuil1)) and (es_cuil_valido(cuil2)):
    print("Los CUILs son válidos")
else:
    print("Alguno de los CUILs NO es válido")
```

Si lo hicimos bien, el programa no sólo ocupará menos espacio sino que será mucho más fácil de comprender.

Funciones para crear soluciones legibles

Si no hubiéramos definido la función **`es_cuil_valido()`** el programa hubiera tenido muchas instrucciones en el medio que hubieran ofuscado la comprensión de la resolución, como ya vimos. Peor aún si las variables tuvieran nombres poco representativos… no se entendería nada del código.

Si el programa está sintácticamente y lógicamente bien, la computadora va a poder resolver el problema pero la idea es crear soluciones que las **personas** puedan comprender. Esto podemos lograrlo definiendo funciones y utilizando nombres descriptivos para variables y funciones.

```
def f(x):
 return x+x*0.21
m=float(input())
print(f(m))
```

vs.

```
def calcular_monto_con_IVA(monto):
    monto_con_IVA = monto + monto * (21/100)
    return monto_con_IVA

pago = float(input("Pago: "))
pago_con_IVA = calcular_monto_con_IVA(pago)
print(f"Pago con IVA: {pago_con_IVA}")
```

Actividad 1-c

Verificación de CUIL

Actividad 1-c | Verificación de CUIL

Una empresa de desarrollo de sistemas bancarios firmó un convenio con un cine para que sus empleados tengan un descuento presentándose de a dos con sus CUILs. En esta parte, el ingreso de los CUILs debe ser validado para que tenga la cantidad justa de caracteres y los guiones se encuentren donde corresponde.

Modificar **precondiciones, estrategia y programa anterior** para que le pida al usuario (el cajero del cine) dos CUILs, que deben ser validados, e imprima por consola un mensaje indicando si ambos CUILs son válidos o alguno de ellos no lo es.

```
CUIL: 20123456786
El CUIL no tiene el formato correcto. Ingréselo nuevamente.
CUIL: 20-12345678-6
CUIL: 20-23456789-7
Los CUILs son válidos
```

Actividad 1-c | Verificación de CUIL

```
...

def pedir_validar_cuil():
    cuil = input("CUIL: ")
    while (len(cuil) != 13) or (cuil[2] != "-") or (cuil[-2] != "-"):
        print("El CUIL no tiene el formato correcto. Ingréselo nuevamente.")
        cuil = input("CUIL: ")
    return cuil

def es_cuil_valido(cuil):
    cuenta = int(cuil[0])*5 + int(cuil[1])*4 + int(cuil[3])*3 + int(cuil[4])*2 + int(cuil[5])*7 + \
             int(cuil[6])*6 + int(cuil[7])*5 + int(cuil[8])*4 + int(cuil[9])*3 + int(cuil[10])*2
    verificador = 11 - (cuenta % 11)
    return (verificador == int(cuil[12]))

cuil1 = pedir_validar_cuil()
cuil2 = pedir_validar_cuil()

if (es_cuil_valido(cuil1)) and (es_cuil_valido(cuil2)):
    print("Los CUILs son válidos")
else:
    print("Alguno de los CUILs NO es válido")
```

Funciones y procedimientos, con y sin parámetros

- **Funciones**: realizan cálculos y/o devuelven resultados. Ejemplos:
  - Obtener y validar ingresos del usuario
  - Obtener resultados de operaciones lógicas, matemáticas y/o cadenas de texto
  - Procesar y/o generar datos estructurados

```
def <función>(<parámetro1>, ..., <parámetroN>):
    <instrucciones>
    return <resultado>
```

```
def <función>():
    <instrucciones>
    return <resultado>
```

- **Procedimientos**: realizan acciones y no devuelven ningún valor. Ejemplos:
  - Mostrar mensajes por consola
  - Modificar datos estructurados

```
def <función>(<parámetro1>, ..., <parámetroN>):
    <instrucciones>
```

```
def <función>():
    <instrucciones>
```

Para cerrar la clase...

Para cerrar la clase...

- Repasemos lo que vimos:
  - **Funciones** para reutilizar soluciones a subproblemas que aparecen varias veces dentro del programa, separadas entre sí.
  - **Funciones** para crear soluciones legibles para cualquier persona, priorizando la comprensión por sobre la ejecución “correcta”.
  - Las funciones pueden tener cualquier cantidad de **parámetros**, recordando tener precaución en el pasaje de parámetros ya que los mismos son posicionales.