1

TEMA 2:

Programación Modular y Estructurada

ÍNDICE

2

Final de estos apartados:

EJEMPLOS ILUSTRATIVOS
TRABAJO PERSONAL

3

2.1. Introducción

- Hasta ahora nosotros hemos aprendido a invocar funciones predefinidas (abs, round…..)
- Otras funciones las hemos utilizado importándolas de algunos módulos (módulo math: sin, log…)
- ***En este capítulo se explicarán todos los elementos necesarios para crearlas***

Programación modular : Diseño de programas mediante la descomposición del problema en módulos sencillos e independientes

Programación estructurada : Programación de cada uno de estos módulos de forma estructurada . Unión de módulos para la resolución del problema global.

4

2.1. Introducción

Recordemos el principal objetivo de la asignatura:

"***Aprender a programar según este paradigma =>Escribir programas que funcionen correctamente y sean claros, legibles y fácilmente actualizables y depurables.***"

5

2.1. Introducción

- Evitar determinados '**malos hábitos**' de programación que llevan a problemas como:
  - Código demasiado largo, enrevesado y poco claro, ilegible en la mayoría de los casos no solo para otros programadores, también para el autor del mismo.
  - Dificultades en la corrección de errores, siendo generalmente difícil la localización de los mismos. Las modificaciones o actualizaciones son costosas por lo que generalmente se recurre a 'parches' que contribuyen a su vez a un empeoramiento en la lógica inicial del programa.
  - A estos inconvenientes se suele añadir una documentación escasa e incompleta y no siempre actualizada.

2.2. Principios de la Programación Estructurada

6

Programación Estructurada.

Principios:

- **Abstracción.**

  No se diseña la solución de un problema pensando en una máquina y lenguaje concreto.

- **Teorema de la Estructura (Bohn y Jacopini)**

  Todo diagrama o programa propio, cualquiera que sea el trabajo que tenga que realizar, se puede implementar utilizando las tres estructuras de control básicas que son la secuencial, alternativa o selectiva y repetitiva o iterativa.

  La importancia de este teorema radica en que estas tres estructuras tienen un único punto de entrada y un único punto de salida => Todo lo que se construya con estas tres estructuras tendrá también un único inicio y un único final.

- **Diseño descendente (Top-down).**

  Resolver el problema original apoyándonos en una serie de subtareas o módulos.

7

2.2.1. Instrucción goto

Observación: la instrucción "ir a" (goto).

- Esta instrucción fue la base de lenguajes como BASIC, o FORTRAN y muchos otros, pero desde la aparición de la programación estructurada, esta instrucción esta proscrita puesto que va contra los principios de la misma.
- Esta instrucción permite saltar desde cualquier punto del programa a cualquier otro, y por tanto **va contra el principio enunciado en el Teorema de la estructura**.
- Esta instrucción nos impide hacer una descomposición del problema, puesto que si desde cualquier lugar podemos ir a cualquier otro es imposible ir separando el programa en áreas claras, todo estará mezclado con todo. Nuestro objetivo es trabajar de forma estructurada.
- ***Norma importantísima**: se llega al fin del programa por un sólo camino, de la misma forma que se sale de un bucle por una única vía.*
- *En programación estructurada esta **totalmente prohibido el uso de la instrucción goto***

Tema 2: Fundamentos de Programación

8

2.3. Pasos para la construcción de un programa modular

Programación modular

- División del problema a resolver en varios subproblemas más sencillos con entidad y sentido propios .
- Búsqueda de la abstracción => no importa como se hace sino que hace cada uno de los módulos.
- Para cada uno de estos subproblemas o módulos se construye un algoritmo que los resuelve.
- Terminada esta fase, se componen todos los módulos para obtener un algoritmo global que resuelva el problema. Esto se conoce como la técnica **Divide y vencerás.**
- Existen dos formas de realizar el diseño modular:
  - **Diseño descendente (Top Down):** Resolver el problema original apoyándonos en una serie de subtareas o módulos que se suponen ya resueltos y obtener la solución del problema, posponiendo la solución de dichas subtareas.
  - **Diseño ascendente (Bottom Up):** Se comienza por resolver las subtareas más pequeñas que contiene el problema original para, posteriormente, unirlas obteniendo la solución del problema.

9

2.3.1. Ejemplo 1

- Algoritmo para el calculo del área de un cilindro:

leer radio y altura

Calcular área circunferencia (*)

calcular longitud circunferencia

Calcular área rectángulo (**)

Calcular área cilindro

(*) área circunferencia = p * radio * radio

Longitud = 2 * p * radio

(**) área rectángulo = longitud * altura

área cilindro= 2 * área circunferencia + área rectángulo

10

2.3.2. Organigramas

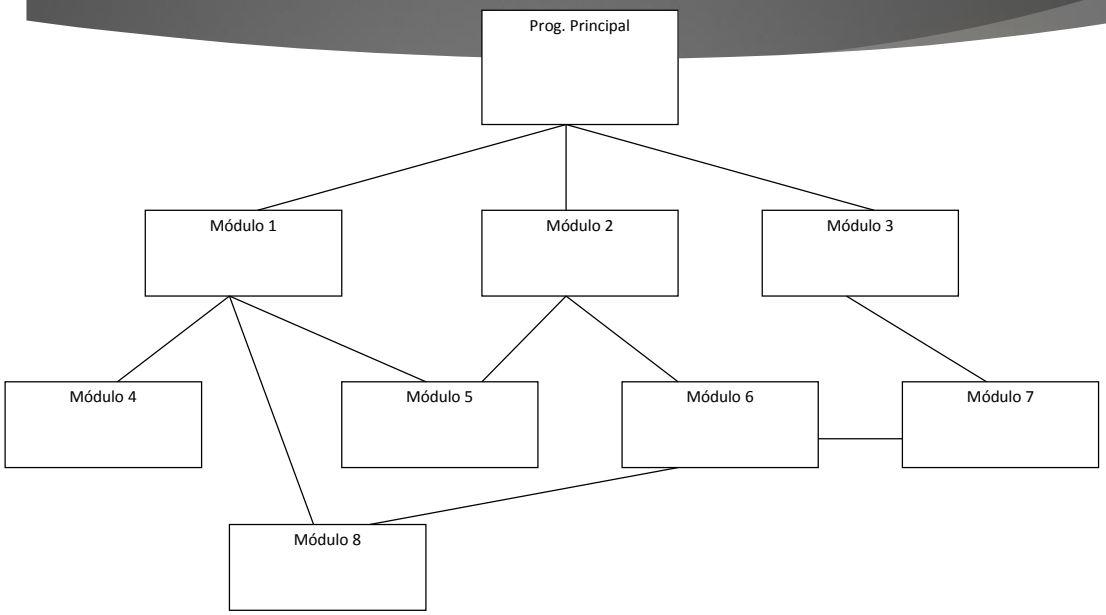
Diseño del organigrama modular, donde se muestran los diferentes módulos en los que se divide el problema inicial y la comunicación entre ellos.

Diseño del modulo principal que realiza la distribución del trabajo entre los diferentes submódulos.

- Incluye básicamente las llamadas a los diferentes submódulos y algunas operaciones de carácter general.
- Cuando se llama a un módulo, el módulo que llama cede el control al submódulo llamado, y este le devuelve el control en el punto inmediatamente siguiente al de la llamada, una vez completado el proceso o tarea que realiza es decir cuando finalicen TODAS SUS INSTRUCCIONES (incluyendo las llamadas a otros módulos si las hay).
- Los módulos se pueden llamar tantas veces como sea necesario, es decir es posible realizar el mismo proceso sobre diferentes datos tantas veces como se precise sin repetir código.

11

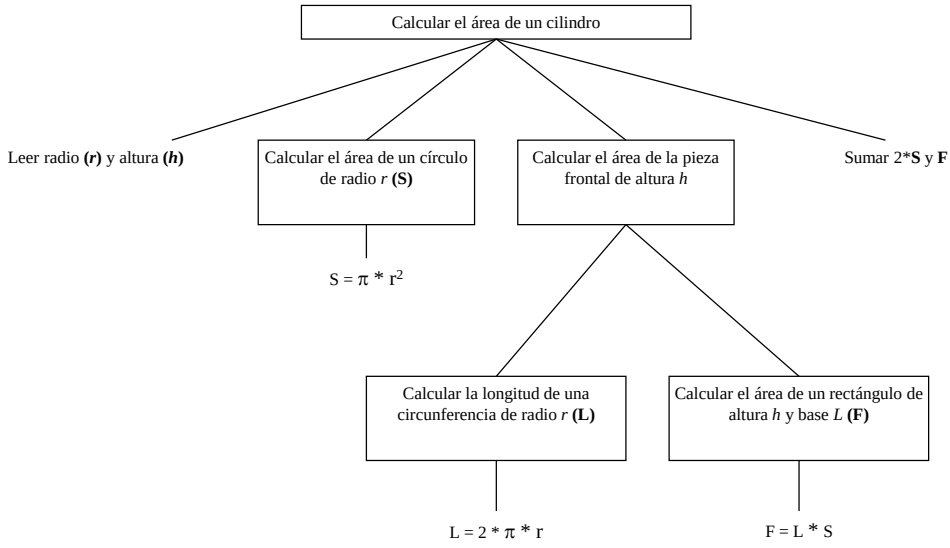
2.3.2. Organigramas



Tema 2: Fundamentos de Programación

12

2.3.3. Organigrama Ejemplo1



Tema 2: Fundamentos de Programación

13

2.3.4. Comunicación entre módulos

Establecer la comunicación entre los módulos: Qué información necesitan, qué información devuelven y quién y a quién se la proporcionan. Los módulos deben cumplir las siguientes características:

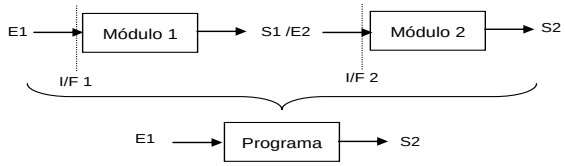
Máxima cohesión, es decir, las salidas de un módulo deben ser compatibles con las entradas de otros módulos.

Mínimo acoplamiento, es decir, la cantidad de información que recibe un módulo de otro módulo debe ser la mínima.

El resultado de un módulo debe ser función directa de sus entradas y no depender de ningún estado interno.

Los módulos serán como **'cajas negras'** que recibe un / unos valores de entrada y devuelven un / unos valores de

salida (los programas son combinaciones de esas cajas negras enlazadas por sus salidas y entradas).



14

2.3.4. Comunicación entre módulos

- Describir en detalle cada módulo básico, como paso previo a su implementación.

  En el caso de que el proceso que realiza un módulo sea a su vez una tarea compleja puede ser necesario volver a dividirlo en otros submódulos con idéntica filosofía (diseño descendente).

- Unión de los subprogramas construidos para obtener la solución del problema (diseño ascendente).

  Esta técnica aporta dos características importantes :

  - Reutilización de código
  - Abstracción

- La programación modular se implementa utilizando módulos que toman diferentes nombres según los lenguajes, funciones en C y C++, subrutinas en BASIC y FORTRAN, funciones y procedimientos en PASCAL, secciones en COBOL...

15

2.3.4. Comunicación entre módulos

- En programación Estructurada normalmente se trabaja con **dos tipos de módulos**:
  - **Funciones y Procedimientos**
- En este curso se aprenderá a implementar **Funciones y Procedimientos en Python** (En python existen dos tipos de funciones dependiendo de su salida, una de ellas simulará el concepto que veremos de función y la otra el concepto de procedimiento)

16

2.4. Ámbito de las variables

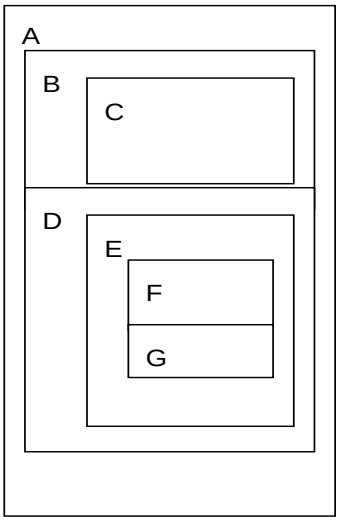
Ámbito de las variables

- La zona del programa en la que una variable puede ser utilizada es lo que se conoce como su **ámbito.** (Es decir, el ámbito es la parte del algoritmo en el que se define una variable => parte en la que la variable es accesible. El mismo concepto para las constantes...).
- Las variables en programación estructurada pueden ser **globales** o **locales** .
  - Una **variable global** generalmente se declara para el programa o algoritmo completo, es decir en el modulo principal. Su valor está disponible tanto en el cuerpo del programa principal como en el de cualquiera de los subprogramas declarados.
  - Una **variable local** es la que está declarada y definida dentro de un subprograma, y es distinta de las posibles variables con el mismo nombre definidas en otros módulos que no estén anidados en su interior, puesto que se refieren a posiciones diferentes de memoria . Su valor sólo está disponible mientras se ejecuta el subprograma.
  - El programa principal no tiene conocimiento alguno de las variables locales de sus procedimientos y funciones. Estas variables hacen a los módulos totalmente independientes.

17

2.4. Ámbito de las variables

Diagrama



Vbles definidas en:	Accesibles desde:
A	A, B, C, D, E, F, G
B	B, C
C	C
D	D, E, F, G
E	E, F, G
F	F
G	G

18

2.4.1 Ejemplos

Ejemplo:

Módulo Principal

VARIABLE x, y, z: entero {*Declaración de variables globales*}

Modulo 1

VARIABLE x, y: entero {*Declaración de variables locales*}

INICIO {Modulo 1}

leer (x) ***{Suponer lectura: 3}***

y← x*100

escribir (x, y) ***{Salida : 3, 300***

Fin { Módulo 1}

Inicio {Módulo Principal}

Leer (x, y) ***{Suponer lectura: 10, 20}***

Llamada a Módulo 1

z ← x + y

Escribir (x, y, z) ***{Salida: 10,20,30 }***

Fin {Módulo Principal}

19

2.4.1 Ejemplos

Módulo Principal

VARIABLE x, y, z: entero {*Declaración de variables globales*}

Modulo 1

VARIABLE x, y: entero {*Declaración de variables locales*}
INICIO {Modulo 1}

leer (x, y) ***{Suponer lectura: 3,2}***

z ← x + y

escribir (z) ***{ Salida: 5}***

FIN { Módulo 1}

Modulo 2

VARIABLE x, z: entero {Declaración de variables locales}

INICIO {Modulo 2}

leer (x, z) {Suponer lectura: 10,20}

z← x + z

escribir (x, y, z) {Salida: 10, 2, 30}

FIN { Módulo 2}

Inicio *{Módulo Principal}*

Leer (x, y, z) *{Suponer lectura: 1,2,3}*

Llamada a Módulo 1

Llamada a Módulo 2

Escribir (x, y, z) *{Salida 1,2,5 se ha modificado z en Modulo 1}*

Fin {Módulo Principal}

La única variable global modificada ha sido z (en Módulo 1)

20

2.4.1 Ejemplos

Módulo Principal

VARIABLE x,y,z: entero {variables globales}

Modulo 1

VARIABLE x,y: entero dato:real; **{variables locales}**

INICIO

leer (x,y) {Suponer lectura: 3,2}

z ← x+y

escribir (x,y, z) {salida : 3,2, 5}

dato:= 10.95

escribir (dato) {salida : 10.95}

Fin {Módulo 1}

Modulo 2

VARIABLE x,z: entero **{variables locales}**

INICIO

leer (x,z) {Suponer lectura: 10,20 }

z← x+z

escribir (x, y, z) {salida: 10, 2, 30}

Fin {Módulo 2}

Inicio {Módulo Principal}

Leer (x,y,z) {Suponer lectura: 1,2,3}

Llamada a Módulo 1

Escribir (x,y,z) {salida : 1,2, 5}

Escribir (dato) {salida : ERROR, variable desconocida}

Llamada a Módulo 2

Escribir (x,y,z) {Salida 1,2,5 modificado z en Modulo 1}

Fin {Módulo Principal}

21

2.4.2. Variables globales y locales en Python

Los conceptos explicados son exactamente igual en Python:

Recordemos en Python, **las variables no se declaran**. Se crean justo en el momento en el que reciben por **asignación un valor por primera vez** y, a partir de ese momento, ya pueden formar parte de una expresión.

- **Variables globales**: residen fuera de toda función, son visibles en todo el programa.
- **Variables locales**: Son creadas dentro de las funciones y solo son visibles dentro de las mismas.
- En python existen además **Variables no locales**. Si a una variable **no se le asigna valor** en una función, Python la considera **libre** y busca su valor en los niveles superiores de esa función, empezando por el inmediatamente superior y continuando hasta el programa principal. Si a la variable se le asigna valor en algún nivel intermedio la variable se considera **no local** y si se le asigna en el programa principal la variable se considera **global**

Las variables *locales*, al contrario que las *globales*, tienen una vida efímera. Solo **existen durante el momento en el que es llamada la función**. En el momento en el que esta concluye, desaparecen.

2.5. Creación de subprogramas

- Los tipos de subprogramas que estudiaremos:
  - **Las funciones** devuelven un sólo valor.
  - **Los procedimientos** devuelven cero, uno o varios valores, en caso de que devuelva cero valores devuelve un dato a la unidad de programa.
- **Parámetros**: Un argumento o parámetro es el medio a partir del cual podemos expandir el ámbito de variables locales de subprogramas, hacia otros subprogramas y además quienes nos permiten establecer comunicaciones entre las diferentes partes de un programa.

2.5. Creación de subprogramas

- La comunicación de un módulo con el resto de partes del programa se debe realizar a través de los valores de su **interfaz**. Cualquier otra comunicación entre el módulo y el resto del programa se denomina *efectos laterales,* considerados como una mala técnica de programación que dificulta el entendimiento de los programas.
- Los módulos se escriben, generalmente, antes que el programa principal (en el bloque declarativo) y constan de las mismas partes que un programa, es decir, sección de declaraciones y cuerpo del módulo.
- La interfaz de un módulo está formada por una lista de **parámetros**, denominados ***parámetros formales.***
- La llamada a un módulo se realiza mediante los **parámetros actuales**.
- Cuando el módulo es llamado se realiza una correspondencia entre los parámetros formales y los parámetros actuales ,que son utilizados en lugar de los parámetros formales, permitiendo así el intercambio de información. Es lo que se conoce como **paso de parámetros**.

24

2.5.1 Paso de parámetros por valor y por referencia

- **Paso por valor:**
  - Los parámetros se tratan como variables locales y los valores iniciales se proporcionan copiando los valores de los correspondientes parámetros actuales.
  - Los cambios que se produzcan por efecto del subprograma no producen cambios en los valores originales es decir no se devuelven valores de retorno al punto de llamada.
  - Por tanto este paso **puede ser** una ***constante, variable o expresión***.
  - El paso por valor se corresponde con parámetros de entrada. Y se especificara anteponiendo la palabra reservada **E/** al parámetro correspondiente en la lista de parámetros formales.

25

2.5.1 Paso de parámetros por valor y por referencia

- **Paso por referencia**:
  - El módulo que llama pasa al módulo llamado la dirección del parámetro actual, de forma que la variable que actúa como parámetro formal es compartida por ambos módulos, siendo visibles las modificaciones que se realicen en el módulo llamado, se devuelven por tanto valores de retorno al punto de llamada.
  - Por tanto **no podemos usar constantes ni expresiones, *solo variables***.
  - El paso por referencia se corresponde con parámetros de entrada / salida o simplemente de salida. La notación que utilizaremos para los parámetros de salida es **S/** y para los de entrada salida: **E/S.**

26

2.5.2 Definición de Función

Funciones

Una función es un módulo que recibe uno o varios datos de entrada y devuelve un **único valor**:

- Las funciones siempre devuelven un valor al programa que las invocó.
- La sintaxis de una función consta por un lado de la **definición o declaración** de la **función** y por otro de la **llamada** a la **función** .

Sintaxis en pseudocódigo de la Declaración o Definición de una función:

```
Funcion identFuncion (lista de parámetros formales): tipo_dato
    declaraciones
Inicio
cuerpo de la función
    identFuncion ← valor {imprescindible para que la función devuelva un valor}
Fin
```

Tema 2: Fundamentos de Programación

27

2.5.2 Definición de Función

Funciones

Sintaxis de la *llamada* a una función

identVble ← *identFunción* (*lista de parámetros actuales*)

ó bien la llamada se utiliza dentro de una expresión

La *variable* debe ser del mismo tipo que el valor de la salida de la función

28

2.5.2.1 Características de las Funciones

- La lista de parámetros formales es la información que se le tiene que pasar a la función debe indicar el tipo de datos que se trasfiere. Estos parámetros, dentro de la función, se utilizan como si fueran variables locales definidas en la función.
- Para cada parámetro hay que poner su nombre (identificador) y tipo de dato. Sintaxis:
  (ident_F1: tipo1, ident_F2 : tipo2,.....)
- La lista de parámetros actuales está formada por un conjunto de identificadores correspondientes a variables, constantes, expresiones o literales separados por comas, definidas en el modulo que llama, del mismo tipo especificado en la lista de parámetros formales y coincidiendo en número y orden con los parámetros formales. Sintaxis:

  (ident_A1, ident_ A2,…, ident_AN)
- El nombre de la función lo da el usuario y tiene que ser significativo.
- En las variables locales se declaran las variables que solo se necesita usar dentro de la función.
- Entre las acciones existirá necesariamente una del tipo ***retorno <valor>***. Esta sentencia pondrá fin a la ejecución de la función y devolverá el valor de la función, asociado al nombre o identificador de la función que será del mismo tipo que el indicado al declarar la función en la parte final de la cabecera.
- No se permiten funciones que no devuelvan nada.

29

2.5.2.2 Ejemplos Funciones pseudocódigo

Ejemplo.

Diseñar una función booleana o lógica que comprueba si un carácter dado es o no un dígito.

```
Funcion es_digito(car:carácter):booleano
    Inicio
        es_digito←(car>='0') y (car<='9')
    Fin
.............
    inicio {modulo principal}
.................
    si es_digito(car)
        escribir (' el carácter es un dígito')
    sino
        escribir(' el caracter no es un digito').
fin.
```

30

2.5.2.3 Definición de Función en Python

En Python las subrutinas reciben el nombre de funciones:

Las funciones se pueden definir en cualquier punto de un programa, aunque siempre antes de ser utilizadas, **NOSOTROS LAS IMPLEMENTAREMOS EN EL BLOQUE DECLARATIVO**.

- La primera línea de la definición de una función contiene: la palabra reservada def.
- El nombre de la función (la guía de estilo de Python recomienda escribir todos los caracteres en minúsculas separando las palabras por guiones bajos).
- Paréntesis (que pueden incluir los argumentos de la función, como se explica más adelante).
- Las instrucciones que forman la función se escriben con sangría con respecto a la primera línea.
- Por comodidad, se puede indicar el final de la función con la palabra reservada return (más adelante se explica el uso de esta palabra reservada), aunque no es obligatorio.
- Ver **ejemplo en apartado 2.5.2.4.**

31

2.5.2.4 Ejemplos Funciones en Python

Diseñar una función booleana o lógica que comprueba si un carácter dado es o no un dígito.

```
# algoritmo para ver si un carácter es un dígito
    def es_digito(car):
    ``` (str)->bool
        Esta función dado un carácter me dice si es un dígito'''
     return (car>='0') and (car<='9')
# Cuerpo del programa
    n = input(' Introduce el carácter a comprobar: ')
    if es_digito(n):   # Llamada a la función en el cuerpo principal
            print(' el carácter es un dígito')
    else:
          print(' el caracter no es un digito')
```
```

32

2.5.2.4 Ejemplos Funciones en Python

En los ejemplos anteriores hemos documentado la función mediante docstring:

```
``` (str)->bool
    Esta función dado un carácter me dice si es un dígito'''
```

- Este comentario sirve para documentar cualquier función, podremos acceder a él en la Shell mediante la función help(nombre de la función), ejemplo:

```
>>> help(abs)
Help on built-in function abs in module builtins:
abs(...)
    abs(number) -> number
    Return the absolute value of the argument.
```

- Es importante crear esta información para todas las funciones que implementemos posteriormente visualizar lo que hace la función en la shell ejemplo

  >>> help(es_digito).

¡¡¡Esta es una costumbre muy recomendable!!!!
```

33

2.5.2.4 Ejemplos Funciones en Python

Ejemplo Diseñar una función que calcule x^3.

```
# Programa eleva un número al cubo
def Eleva_cubo(x):
    resultado= 1
    for i in range(1,4):
        resultado = resultado * x
    return resultado
#cuerpo del programa
base = int(input(' Dime la base '))
potencia = Eleva_cubo(base)
print(' El número ', base, 'elevado al cubo es: ', potencia)
```

34

2.5.2.4 Ejemplos Funciones en Python

Ejemplo.

Diseñar una función que compruebe si un número es par.

```
# algoritmo para comprobar si un número es par
        def es_par(n):
          return  n%2 == 0


# MISMA FUNCIÓN PROGRAMADA POR NOVATOS
        def es_par():
          n = int(input(' Dame un número '))
          return  n%2 == 0
```

*#este **código esta mal** demuestra falta de soltura, esta función solicita un numero entero. Si piden una función que recibe uno o más datos se sobreentiende que los datos debes suministrarlos como argumentos en las llamadas.*

35

2.5.2.4 Ejemplos Funciones en Python

Resumen Partes de una Función con Python

```
def es_perfecto (n): # 1. cabecera incluye nombre y parámetros
    #2. docstring es un comentario que siempre añadiremos indicando los parámetros de entrada y salida y un resumen de lo que hace el #módulo y a veces también por aclarar, podemos añadir casos de uso
    ''' (n)->bool  Esta función dado un número evalúa si es un número perfecto: es un entero que es  igual a la suma de los divisores propios menores que él mismo'''
    # 3. Cuerpo de la función
        sumatorio = 0
        for i in range (1,n):
            if n % i == 0:
                sumatorio += i
        return sumatorio == n
```

```
# 3. Cuerpo del programa

m = int(input('Dame dato'))
a = es_perfecto(m)
if a:
    print ('es perfecto')
else:
    print('no es perfecto')
```

Tema 2: Fundamentos de Programación

36

2.5.2.5 Llamada a Funciones predefinidas en Python

>>> import builtins # estas dos instrucciones muestran todas las funciones de python

>>> dir(builtins)

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

37

2.5.2.5 Llamada a Funciones en Python

>>> help(nombre de una función)#Ayuda para funciones

>>> help(abs)

Help on built-in function abs in module builtins:

abs(...)

abs(number) -> number

Return the absolute value of the argument.

>>> n=abs(-2) #Ejemplos

>>> n

2

38

TRABAJO PERSONAL

Tema1: Fundamentos de Programación

39

Trabajo personal

Lectura recomendada

Cap 6 Libro Introducción Programación Python (Andrés Marzal).

Laboratorio Tema 2

Resolver ejemplos con Python empleando la sentencias más adecuada:

1. Suma de 2 números.
2. Calculo de la longitud de una circunferencia.
3. Calculo del cuadrado de un número.
4. *Dada la edad de una persona ver si alguien es mayor de edad.*

40

2.5.3 Definición de Procedimientos

Procedimientos

Un procedimiento es un subprograma que recibe cero o más valores de entrada y puede devolver cero o más valores de salida.

En los procedimientos, tanto la entrada como la salida de información se realizan a través de Los parámetros.

De nuevo es necesario distinguir entre la *sintaxis* de la ***declaración*** o ***definición*** de un procedimiento y la ***sintaxis*** de la ***llamada*** al mismo.

La *sintaxis* de la *declaración o definición* de un procedimiento es:

Procedimiento *identProcedimiento* (*lista de parámetros formales*)
    Declaraciones
Inicio
    Cuerpo
Fin
.....

Sintaxis de la *llamada* a un procedimiento:

identProcedimiento (*lista de parámetros actuales*)

Tema 2: Fundamentos de Programación

41

2.5.3 Definición de Procedimientos

Procedimientos

En los procedimientos la lista de parámetros formales debe indicar el tipo de datos que se trasfiere y el tipo de paso de parámetro. Sintaxis:

([E/ o E/S o S/] ident_P1: tipo1, [E/ o E/S o S/] ident_P2 : tipo2,.....)

La lista de parámetros actuales serán variables, definidas en el modulo que llama, del mismo tipo especificado en la lista de parámetros formales y en el mismo orden.

(ident_A1, ident_ A2,…, ident_AN)

Los parámetros actuales deben coincidir en tipo, número y orden con los parámetros Formales.

42

2.5.3 Definición de Procedimientos

Diferencias entre funciones y procedimientos:

- Una función devuelve un único valor y un procedimiento puede devolver 0,1 o N.
- Ninguno de los resultados devueltos por el procedimiento se asocian a su nombre como ocurría con la función.
- Mientras que la llamada a una función forma siempre parte de una expresión, la llamada a un procedimiento es una instrucción que por sí sola no necesita instrucciones.
- Esta llamada consiste en el nombre del procedimiento y va entre paréntesis van los parámetros que se le pasan.

43

2.5.3.1 Definición de Procedimientos en Python

- No todas las funciones devuelven un valor. Una función que no devuelve un valor se denomina *procedimiento*.
- ¿Para qué sirve una función que no devuelve nada?:
- Mostrar mensajes o resultados por pantalla. No te equivoques: mostrar algo por pantalla no es devolver nada. Mostrar un mensaje por pantalla es un efecto secundario.
- Veámoslo con un ejemplo. Vamos a implementar ahora un programa que solicita al usuario un número y muestra por pantalla todos los números perfectos entre 1 y dicho número.
- En Python no existe el paso de parámetro por valor y por referencia, en realidad los valores mutables se comportan como paso por referencia, y los inmutables como paso por valor.

- https://librosweb.es/libro/python/capitulo_4/definiendo_funciones.html

2.5.3.1 Definición de Procedimientos en Python

Sintaxis:

```
def mi_funcion(param1, param2):
    print (param1)
    print (param2)
```

Es decir, la palabra clave def seguida del nombre de la función y entre paréntesis los argumentos separados por comas. A continuación, en otra línea, indentado y después de los dos puntos tendríamos las líneas de código que conforman el código a ejecutar por la función.

También podemos encontrarnos con una cadena de texto como primera línea del cuerpo de la función. Estas cadenas se conocen con el nombre de docstring (cadena de documentación sirven para documentar la función).

```
def mi_función(param1, param2):
'''Esta función imprime los dos valores pasados como parametros'''
    print (param1)
    print (param2)
```

45

2.5.3.1 Definición de Procedimientos en Python

```
##Función tabla perfecto utiliza función anterior.  Observar la llamada en el cuerpo del programa
def es_perfecto (n):
    sumatorio = 0
    for i in range (1, n):
        if n % i == 0:
            sumatorio += i
    return sumatorio == n
def tabla_perfectos(m): #Muestra todos los números perfectos entre 1 y m
    for i in range(1,m+1):
        if es_perfecto(i):
            print(i, 'es un número perfecto')
# cuerpo de programa
n = int(input(' Tabla de números perfecto de 1 a : '))
tabla_perfectos(n)
```

```
>>>
 Tabla de números perfecto de 1 a : 100
6 es un número perfecto
28 es un número perfecto
>>>
```

Tema 2: Fundamentos de Programación

2.5.3.2 Ejemplos de Procedimientos Python (Parametros por valor)

Algoritmo EjercicioA

Variable

a,c: entero

Procedimiento pa (E/ b: entero)

Variable

c: entero

inicio

c← b+5

b← 2

fin

Inicio

c ← 3

a← 2

pa (a)

Escribir (c, a) ***{Salida → 3, 2}***

fin

 # Implementación código Python

def pa(b):

c = b+5

b = 2

c = 3

a = 2

pa (a)

print (c, a) ***#Salida → 3, 2***

#a se esta utilizando como variable global

#paso de parámetro a por valor, la variable c

Aparece en cuerpo principal y se vuelve a llamar en función (var global)como no se dice nada se volveria a definir y sería también una variable local del procedimiento

47

2.5.3.2 Ejemplos de Procedimientos Python (Uso variables globales)

Algoritmo ejercicio8

```
Variable
    a, c: entero
Procedimiento pb (E/ b: entero)
    Inicio
        a← 4
        b← 5
    fin
 Inicio
        c←3
        a ←2
        pb(c)
        Escribir (c,a) {salida→3, 4}
Fin
```

a se esta utilizando como variable global

 # Implementación código Python

```
def pb (b):
    global a # En programación Estruc Prohibido toma valor del cuerpo principal
    a = 4
    b = 5

#Cuerpo Principal
c = 3
a = 2
pb(c)
print(c,a)
#salida 3, 4, La variable a se ha modificado en el procedimiento pb Prohibido en Programación Estructurada
```

48

2.5.3.3 Ejemplos de Procedimientos Python (Parametros por valor y por referencia)

Algoritmo ejercicioC

```
Variable    a, b: entero
Procedimiento pc (E/ i: entero, E/S j: entero)
  Inicio
      i ← i+10
      j← j+10
      Escribir(i, j)      {salida → 12, 13}
    Fin
Inicio
      a← 2
      b← 3
      pc(a,b)
      Escribir (a, b)  {salida → 2, 13}
Fin
```

 # Implementación código Python

```
def pc (i,b):

   i = i+10
   b=  b+10
   print(i, b)
   return b
#salida funcion 12, 13

a = 2
b = 3
b = pc(a,b)
print (a, b)
# salida 2, 13. Parametro b por referencia se modifica valor variable global
```

2.5.3.3 Ejemplos de Procedimientos Python (Parametros por valor y por referencia)

Programa que calcula el área y la longitud de una Circunferencia

```
Pi= 3.1416
def Calculos ( r):
    a = 2 * Pi * r
    l = Pi * r * r
    return a,l

def respuesta():
    while True:
        resp = input('¿Quieres salir? (S/N)')
        if resp.upper()== 'S' or resp.upper()=='N':
            return resp
            break

#Cuerpo principal
print('¿Desea calcular el área  y longitud de una circunferencia?')
resp = respuesta()
while resp.upper() == 'N':
    radio = int(input(' Dime el radio'))
    area, long = Calculos (radio)
    print(area, long)
    resp = respuesta()
#area y long: simularian el paso de parámetro por referencia cuando las
#variables  son solo de salida
#radio: paso de parámetro por valor
```

2.5.3.3 Ejemplos de Procedimientos Python (Parametros por valor y por referencia)

50

```
def es_par (x):
    return es_par == ((x % 2) == 0)
def modulo( t , b):
    a = 0
    while t == 1:
        a = a + 1
        t  = 2
        b = b + a
    return b
```

```
#Cuerpo Principal
x = 10
k  = 1
while True:
    if es_par(x):
        x = modulo(k,x)
    else:
        k = k+1
    if k > 2:
        break
print (x , k)
```

51

2.5.3.4 Resumen para simular parámetros por valor y referencias con Python

- Parámetros por valor:

Aparecen en cabecera de procedimiento y no salen en return,

- Parámetros por referencia:

1. Salida-> Han de salir mediante instrucción return del procedimiento y ser asignados a una variable
2. E/S ->Aparecen en cabecera de procedimiento y han de salir mediante instrucción return del procedimiento y ser asignados a una variable

52

TRABAJO PERSONAL

Tema1: Fundamentos de Programación

53

2.6.2 Trabajo personal

Lectura recomendada

Cap 6 Libro Introducción Programación Python (Andrés Marzal).

Laboratorio Tema 2

Resolver ejemplos

Implementar los ejercicios del tema 1 parte 3 bucles modularizándolos