Universidad Nacional de Jujuy
Facultad de Ingeniería

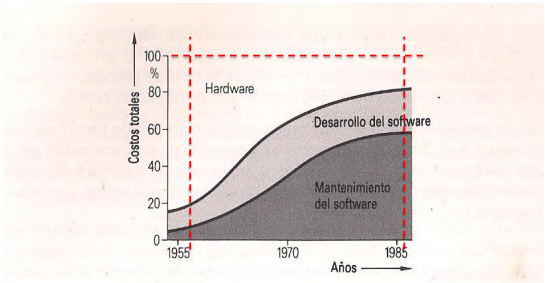
Introducción a la Programación
Programación I

Programación modular

Contenido

- Crisis del Software
- Modularización: concepto y justificación
- Diseño Top-Down
- Comunicación entre módulos
- Ámbito de las variables
- Efectos laterales
- Funciones

Desarrollo del *software*



Distribución de los costos totales en hardware y software

Crisis del *software*

Término utilizado en 1968, en la primera conferencia organizada por la OTAN sobre desarrollo de software. Refiere a la dificultad de escribir programas libres de defectos (bugs), fácilmente comprensibles y que sean verificables (fines de la década del 60)

Factores de influencia

- Aumento del poder computacional.
- Reducción del costo del hardware.
- Rápida obsolescencia de hardware y software.

Causas y sucesos

Causas:

- Complejidad en la forma de programar
- Cambios que afectan los programas
- Sin herramientas para estimar el esfuerzo

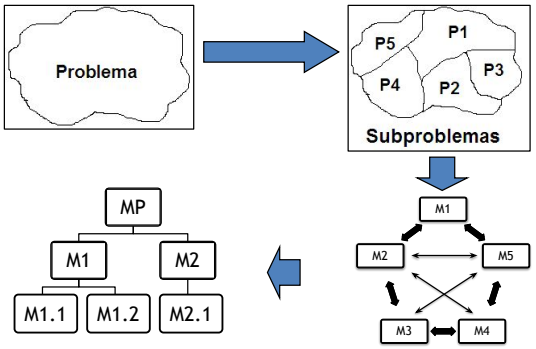
Sucesos que se observan

- Los proyectos no terminan en plazo
- Los presupuestos no se ajustan al inicial
- Baja calidad del software generado
- No cumplía con las especificaciones
- Código difícil de mantener

Divide y conquista

- Es fácil resolver los pequeños problemas pero es complejo resolver los grandes.
- Se resuelve un problema separando un algoritmo en partes que son más manejables.
- Las partes son resueltas individualmente, luego las soluciones pequeñas son integradas y se obtiene la gran solución.
- Este proceso se denomina descomposición, "divide y conquista" o comúnmente diseño *top-down*.

Complejidad - Diseño *top-down*



Modularización: justificación

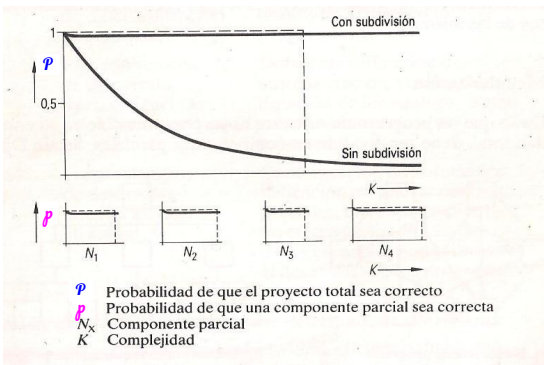
"Si la probabilidad de que un componente de un programa sea correcto es p entonces, la probabilidad P de que todo el programa de N componentes sea correcto es:

$$P = p^N$$

Si N es muy grande entonces p debe diferir muy poco de uno, si se quiere que P difiera de cero"

Edsger Dijkstra

Modularización: justificación - probabilidad



Diseño de los módulos

- Si un problema es difícil de modularizar, hay que intentar redefinir las tareas involucradas en el problema.
- Demasiados casos especiales o demasiadas variables que requieren un tratamiento especial, son signos de que la definición del problema es inadecuado

En consecuencia los módulos

- Deberán estar jerarquizados
- Deberán ser pequeños, sencillos e independientes
- Deben resolver una sola tarea

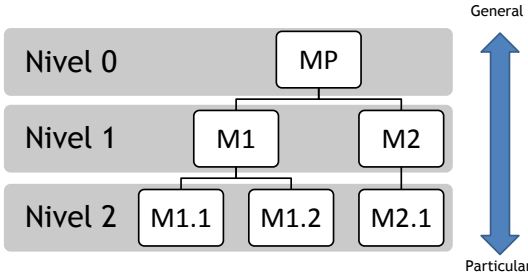
Modularización: ventajas

- Simplifica el análisis y el diseño.
- Disminuye la complejidad de los algoritmos.
- Disminuye el tamaño total del programa.
- Promueve la reusabilidad.
- Promueve la abstracción
  - Características de una caja negra.
  - Aislamiento de los detalles.
- Favorece el trabajo en equipo.
- Facilita la depuración y prueba.
- Facilita el mantenimiento.
- Permite la creación de librerías específicas.

Modularización: desventajas

- La integración de los módulos puede ser compleja en particular si fueron escritos por diferentes personas.
- Los módulos requieren de documentación cuidadosa dado que pueden afectar a otras partes del programa. Se debe prestar mucha atención cuando se utilizan módulos que modifican estructuras de datos compartidas con otros módulos.
- La prueba de un módulo puede ser difícil en situaciones en las que se necesitan datos generados por otros módulos.

Diseño top-down

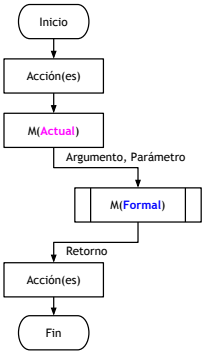


Módulo función en Python: definición

Son módulos, subrutinas o subalgoritmos que son llamados o invocados desde algún punto del programa o desde otro módulo, a través de un identificador como una acción simple o dentro en una expresión algebraica, relacional o lógica, con o sin una lista de argumentos y podrá o no devolver valores.

def nombre_funcion (<par1,par2,…>):
<"""*Documentación de la función*""">
acción(es) #cuerpo de la función
<**return** val1, val2, ...>

Parámetros: invocación y recepción



La *lista de argumentos y parámetros* son el medio de comunicación formal entre los módulos.

Argumento Actual: son valores constantes, variables o expresiones que son pasadas al módulo en el momento de la invocación.

Parámetro Formal: lista de parámetros que aparecen en la definición del módulo.

Ámbito de las variables

Para poder utilizar una función en un programa se tiene que haber definido antes. Por ello, normalmente las definiciones de las funciones se suelen escribir al principio de los programas.

Python distingue tres tipos de variables: las variables locales y dos tipos de variables libres (globales y no locales):

- variables locales: las que pertenecen al ámbito de la subrutina (y que pueden ser accesibles a niveles inferiores)
- variables globales: las que pertenecen al ámbito del programa principal.
- variables no locales: las que pertenecen a un ámbito superior al de la subrutina, pero que no son globales.

Para identificar explícitamente las variables globales y no locales se utilizan las palabras reservadas *global* y *nonlocal*. Las variables locales no necesitan identificación.

Variables globales y no locales

```
1   def modulo_uno():
2       global a  #2
3       a = 7     #3
4       print(a)  #4
5       return
6
7   a = 5
8   modulo_uno()
9   print(a)
```

```
#Casos
#2, -, -, -, 2
#3, 3, -, 4, 4
#4, 4, 4, 3, 3

#7, 7, 5, e, 5
#7, 5, 5,  , 7
```

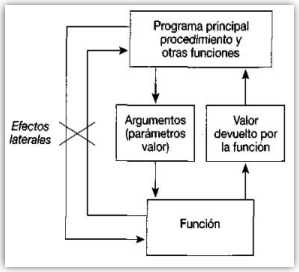
```
1   def modulo_uno():
2       def modulo_dos():
3           #nonlocal a  #3
4           #global a    #4
5           print(a)
6           a = 1
7           return
8
9       a = 3            #9
10      modulo_dos()
11      print(a)
12      return
13
14  a = 4
15  modulo_uno()
16  print(a)
```

```
#Casos
#3, -, 3, 3, -
#4, -, -, -, 4
#9, 9, 9, -, 9

#e, e, 3, e, 4
#  ,  , 1,  , 3
#  ,  , 4,  , 1
```


Efectos laterales

La comunicación entre el programa principal (o módulos) con los demás módulos, debe realizarse a través de parámetros. Cualquier otra comunicación se conoce como efectos laterales.



Paso de argumentos y parámetros

```python
#Valores iniciales
def formula(x=3,y=2,z=1):
    print(x + y - z)

#principal
formula(1,2,3) #0
formula(1) #2
```

```python
#Orden de los parámetros
def formula(x=3,y=2,z=1):
    print(x + y - z)

#principal
formula(z=8,x=5,y=9) #6
```

```python
#funcion como parámetro
def incremento(x):
    return x + 1

def mostrar(y):
    print(y)

#Principal
mostrar(incremento(5)) #6
```

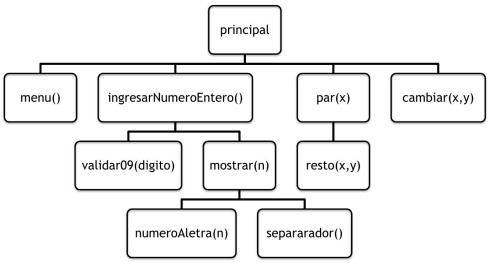
```python
def calcular(numero1, numero2):
    suma = numero1 + numero2
    resta = numero1 - numero2
    producto = numero1 * numero2
    return suma, resta, producto

#principal
print(calcular(15, 8)) #(23, 7, 120)
s,r,p = calcular(7, 25)
print(s,r,p) #32 -18 175
```


Ejemplo: Diseño top-down

Mediante un menú de opciones resolver:

1. Ingresar dos valores A y B validando que estén en el intervalo [0,9] y mostrarlos en letras separados por una serie de asteriscos
2. Si A y B son pares intercambiar los valores y mostrarlos en letras
3. Salir



Módulo: principal

```python
#Módulo principal
opcion = 0
while (opcion !=3):
    os.system ("clear") #windows "cls"
    opcion = menu()
    if (opcion == 1):
        a = ingresarNumeroEntero()
        b = ingresarNumeroEntero()
    elif (opcion == 2):
        if (par(a) and par(b)):
            a,b = cambiar(a,b)
            mostrar(a)
            mostrar(b)
    elif (opcion == 3):
        print('Hasta la vista baby ...')
    else:
        print('Error')
    input('Para continuar presione enter ...')
```


Módulos: menú, validar09, ingresoDatos

```python
def menu():
    print('1) Ingresar A y B. Mostrar letras')
    print('2) Si A y B son pares intercambiar')
    print('3) Salir')
    eleccion = int(input('Elija una opción: '))
    while not((eleccion >= 1) and (eleccion <= 3)):
        eleccion = int(input('Elija una opción: '))
    return eleccion
```

```python
def validar09(digito):
    aux = False
    if ((digito>=0) and (digito<=9)):
        aux = True
    return aux
```

```python
def ingresarNumeroEntero():
    x = int(input('ingrese un valor entre [0,9]: '))
    while (not validar09(x)):
        x = int(input('ingrese un valor entre [0,9]: '))
    mostrar(x)
    return x
```


Otros módulos

```python
def mostrar(n):
    separador()
    print(numeroAletra(n))
    separador()
```

```python
def separador():
    print('*'*5)
```

```python
def par(x):
    aux = False
    if (resto(x,2)==0):
        aux = True
    return aux
```

```python
def resto(x,y):
    r = x-x//y*y
    return r
```

```python
def cambiar(x,y):
    aux = x
    x = y
    y = aux
    return x,y
```

```python
def numeroAletra(n):
    if (n==0):
        cadena = 'cero'
    elif (n==1):
        cadena = 'uno'
    elif (n==2):
        cadena = 'dos'
    elif (n==3):
        cadena = 'tres'
    elif (n==4):
        cadena = 'cuatro'
    elif (n==5):
        cadena = 'cinco'
    elif (n==6):
        cadena = 'seis'
    elif (n==7):
        cadena = 'siete'
    elif (n==8):
        cadena = 'ocho'
    elif (n==9):
        cadena = 'nueve'
    else:
        cadena = 'error'
    return cadena
```


Bibliografía

- Libro: Introducción a la Programación con Python.
  - Capítulo 6 - Funciones
- https://docs.python.org/es/3/
- https://docs.python.org/es/dev/faq/programming.html
- https://www.mclibre.org/consultar/python/lecciones/python-funciones-1.html