Programación Modular
Fundamentos de la programación

Elena G. Barriocanal, Salvador Sánchez

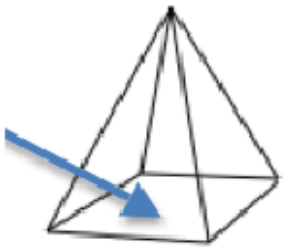
Universidad de Alcalá

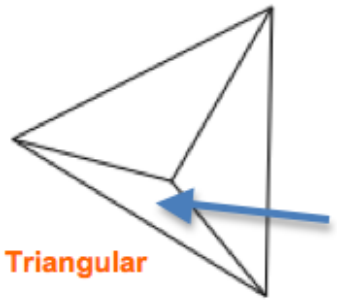
Septiembre de 2017

Licencia

Problema: cálculo del área de una pirámide

- $area = area_{base} + area_{lateral}$
- $area_{base} = ladobase^2$
- $area_{lateral} = \frac{perimetro \times apotema}{2}$
- $apotema = \sqrt{altura^2 + (\frac{ladobase}{2})^2}$
- En el caso de una pirámide de base triangular, $area_{base} = \frac{ladobase \times altura_{base}}{2}$
- Y para obtener la altura de la base: $altura_{base} = \sqrt{ladobase^2 - (\frac{ladobase}{2})^2}$





Calculando el área de pirámides

```
# Calculo del areas de varios tipos de piramide

# Piramide regular cuadrangular
lado_base = float(input("Introduzca la base de la piramide: "))
h_piramide = float(input("Introduzca ahora la altura: "))
apotema = pow((pow(h_piramide, 2) + pow((lado_base/2),2)),0.5)
perimetro = lado_base * 4
area_lateral = (perimetro * apotema) / 2
area_base = pow(lado_base, 2)
area_total = area_lateral + area_base
print("Area de la piramide cuadrangular: ", area_total);

# Piramide regular triangular
lado_base = float(input("Introduzca la base de la piramide: "))
h_piramide = float(input("Introduzca ahora la altura: "))
apotema = pow((pow(h_piramide, 2) + pow((lado_base/2),2)),0.5)
perimetro = lado_base * 3
h_base = pow(pow(lado_base,2) - pow((lado_base / 2), 2), 0.5)
area_lateral = (perimetro * apotema) / 2
area_base = (lado_base * h_base) / 2
area_total = area_lateral + area_base
print("Area de la piramide triangular: ", area_total);
```

Algunos problemas del programa anterior

- Ilegibilidad.

Algunos problemas del programa anterior

- Ilegibilidad.
- Repetición del código: Cálculo del perímetro, área lateral, área total, muestra de resultados, etc.

Algunos problemas del programa anterior

- Ilegibilidad.
- Repetición del código: Cálculo del perímetro, área lateral, área total, muestra de resultados, etc.
- Dificultad de depuración.
  - ¿Qué ocurre –por ejemplo– si hemos implementado mal la fórmula para calcular el área lateral?

Algunos problemas del programa anterior

- Ilegibilidad.
- Repetición del código: Cálculo del perímetro, área lateral, área total, muestra de resultados, etc.
- Dificultad de depuración.
  - ¿Qué ocurre –por ejemplo– si hemos implementado mal la fórmula para calcular el área lateral?
  - Se nos puede olvidar modificar la fórmula en alguna línea.

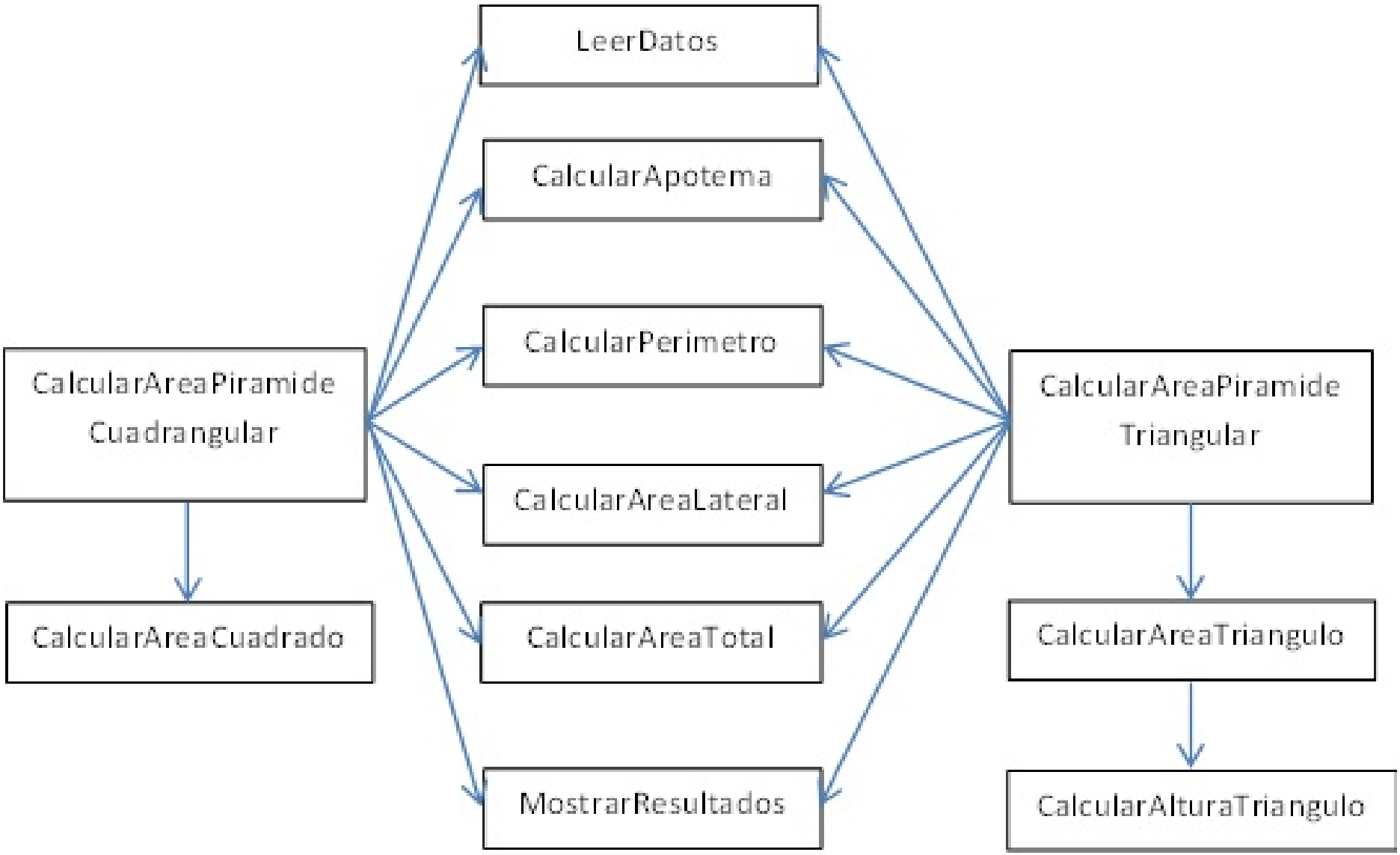
Algunos problemas del programa anterior

- Ilegibilidad.
- Repetición del código: Cálculo del perímetro, área lateral, área total, muestra de resultados, etc.
- Dificultad de depuración.
  - ¿Qué ocurre –por ejemplo– si hemos implementado mal la fórmula para calcular el área lateral?
  - Se nos puede olvidar modificar la fórmula en alguna línea.
- Dificultad de mantenimiento.

Algunos problemas del programa anterior

- Ilegibilidad.
- Repetición del código: Cálculo del perímetro, área lateral, área total, muestra de resultados, etc.
- Dificultad de depuración.
  - ¿Qué ocurre –por ejemplo– si hemos implementado mal la fórmula para calcular el área lateral?
  - Se nos puede olvidar modificar la fórmula en alguna línea.
- Dificultad de mantenimiento.
- Mayor coste de desarrollo.

Pasos para calcular el área de una pirámide regular



Subprograma

Definición

Un subprograma es una porción de código relativamente independiente que puede ser **llamado**, **enviándole (o no) datos** para que realice una determinada **tarea** y/o proporcione una serie de **resultados**.

- Cuando el subprograma retorna valor(es) al código que lo invocó hablamos de una **función**
- Si por el contrario no retorna valores, es un **procedimiento**

Ventajas de usar subprogramas

- **Reutilización** de código.
  - El mismo cálculo del perímetro, área lateral, etc. se reutiliza para calcular el área de la pirámide triangular y cuadrangular.

Ventajas de usar subprogramas

- **Reutilización** de código.
  - El mismo cálculo del perímetro, área lateral, etc. se reutiliza para calcular el área de la pirámide triangular y cuadrangular.
  - Un mismo subprograma puede ser utilizado por diferentes programas o en diferentes partes del programa.

Ventajas de usar subprogramas

- **Reutilización** de código.
  - El mismo cálculo del perímetro, área lateral, etc. se reutiliza para calcular el área de la pirámide triangular y cuadrangular.
  - Un mismo subprograma puede ser utilizado por diferentes programas o en diferentes partes del programa.
- **Legibilidad**.

Ventajas de usar subprogramas

- **Reutilización** de código.
  - El mismo cálculo del perímetro, área lateral, etc. se reutiliza para calcular el área de la pirámide triangular y cuadrangular.
  - Un mismo subprograma puede ser utilizado por diferentes programas o en diferentes partes del programa.
- **Legibilidad**.
- Mejora de los procesos de **depuración y pruebas**.
  - Ambos se llevan a cabo por subprogramas, no globalmente.

Ventajas de usar subprogramas

- **Reutilización** de código.
  - El mismo cálculo del perímetro, área lateral, etc. se reutiliza para calcular el área de la pirámide triangular y cuadrangular.
  - Un mismo subprograma puede ser utilizado por diferentes programas o en diferentes partes del programa.
- **Legibilidad**.
- Mejora de los procesos de **depuración y pruebas**.
  - Ambos se llevan a cabo por subprogramas, no globalmente.
- Facilidad de **mantenimiento**.
  - Las modificaciones se realizan una sola vez: en el subprograma.

Subprogramas: Características

- Fuerte **cohesión**:
  - Debe existir una clara relación funcional entre las sentencias o grupos de sentencias de un mismo subprograma.
  - Un subprograma fuertemente cohesionado ejecutará una única tarea sencilla.

Subprogramas: Características

- Fuerte **cohesión**:
  - Debe existir una clara relación funcional entre las sentencias o grupos de sentencias de un mismo subprograma.
  - Un subprograma fuertemente cohesionado ejecutará una única tarea sencilla.
- **Idea**: para ver si es cohesivo un subprograma, analizar si puede describirse con una oración simple, con un solo verbo activo. Si hay más de un verbo activo en su descripción, sopesar su descomposición en más de un subprograma.

Subprogramas: Características

- Débil **acoplamiento**:

Subprogramas: Características

- Débil **acoplamiento**:
  - Debe existir poca interdependencia entre los subprogramas de un programa.

Subprogramas: Características

- Débil **acoplamiento**:
  - Debe existir poca interdependencia entre los subprogramas de un programa.
  - Un subprograma se comunicará con otros únicamente a través de los parámetros de entrada y de los valores de salida.

Subprogramas: Características

- Débil **acoplamiento**:
  - Debe existir poca interdependencia entre los subprogramas de un programa.
  - Un subprograma se comunicará con otros únicamente a través de los parámetros de entrada y de los valores de salida.
  - Evitar el uso de variables globales

Implementación de subprogramas

Definición

La definición de un **subprograma** consiste en la escritura del código necesario para que éste realice las tareas para la que ha sido previsto.

Implementación de subprogramas

Definición

La definición de un **subprograma** consiste en la escritura del código necesario para que éste realice las tareas para la que ha sido previsto.

```
# (1) Encabezamiento o cabecera
def nombre_subprograma (lista_argumentos):
# (2) Documentacion  (docstrings)
# (3) Instrucciones del subprograma
 return (expresion) # (4) Retorno [opcional]
```

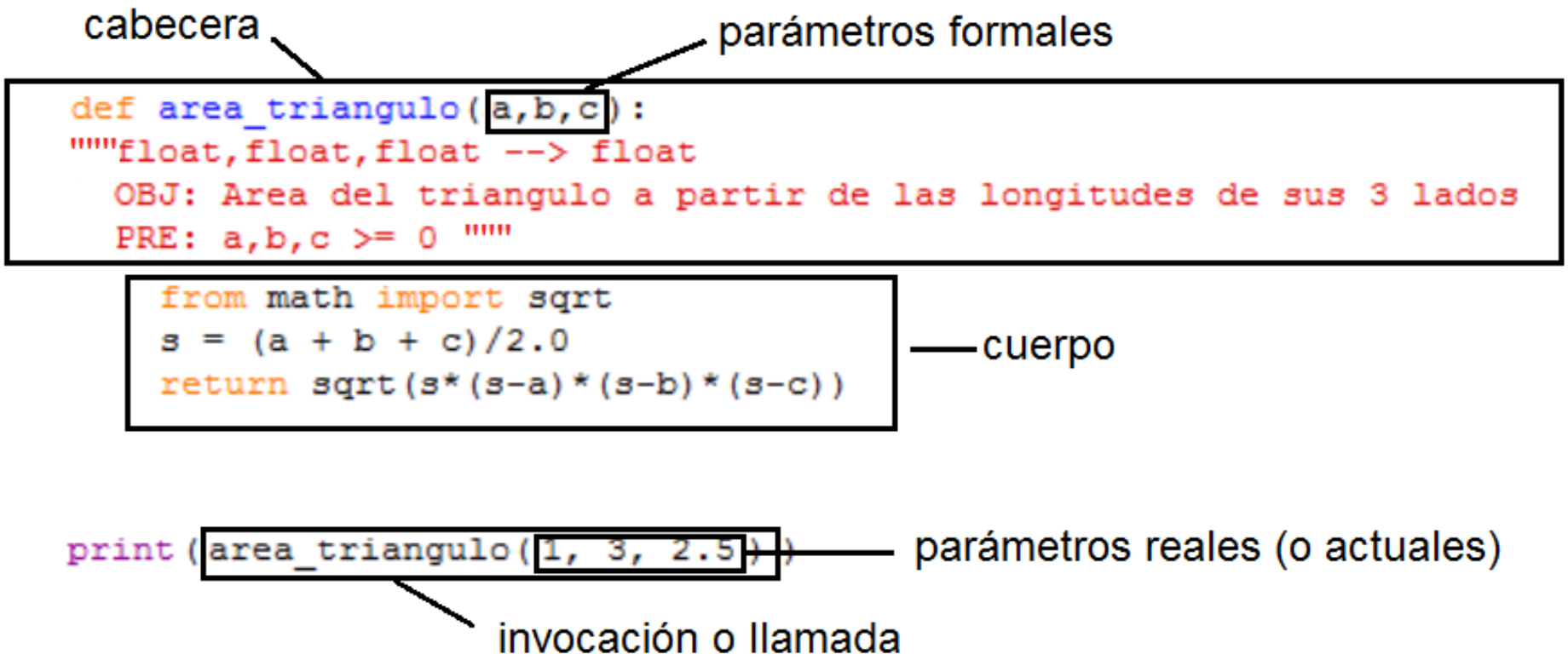
Documentación de los subprogramas

- La documentación es transparente para la máquina: sin ella el subprograma funciona igual, pero es muy importante para su reutilización
- Debe ser clara y concisa.
- Orientada a los programadores: puede incluir términos técnicos.
- En Python se escribe con docstrings: entre tres símbolos de dobles comillas (“ “ “ documentación aquí ” ” ”) detrás de la instrucción `def` y antes el cuerpo del subprograma.

Documentación de los subprogramas

- Línea 1: especificar el tipo de los argumentos y de la salida, si es una función
- Línea 2: tras "OBJ:" (objetivo) una frase que sintetice el objetivo del subprograma, el papel que desempeña cada argumento y lo que devuelve el subprograma (si es función).
- Si existen condiciones previas para que el subprograma haga su trabajo, deben indicarse en una línea aparte (3) tras OBJ, precedida por "PRE:" (precondición).
- La instrucción `def` y la documentación conforman la cabecera.
  - La cabecera es el contrato del subprograma con el mundo: "si Vd. me da argumentos que satisfagan lo especificado en PRE, le garantizo que haré lo que dice OBJ".

Estructura de una función



Ejemplos de subprogramas

```
def conversion_a_pesetas(importe_en_euros):
    """ float  --> float
    OBJ: convierte euros a pesetas
    """
    importe_en_pesetas = importe_en_euros * 166.386
    return (importe_en_pesetas)

def conversion_a_dolares(importe_en_euros, cambio_del_dia):
    """ float, float  --> float
    OBJ: convierte euros a dolares segun el cambio del dia
    """
    return (importe_en_euros * cambio_del_dia)

def saludo_bilingue(nombre):
    """ str  --> nada
    OBJ: saluda en espanol e ingles a una persona
    """
    print("Hola,", nombre)
    print("Hello,", nombre)
```

Implementación de los subprogramas

- Una función se puede llamar desde otra parte del programa posterior a su definición.

```
# Llamada o invocacion:
nombre_funcion (lista_arg_reales)
```

Implementación de los subprogramas

- Una función se puede llamar desde otra parte del programa posterior a su definición.

```
# Llamada o invocacion:
nombre_funcion (lista_arg_reales)
```

- Dependiendo del tipo de retorno, las llamadas se pueden realizar:
  - Dentro de una expresión: El tipo de retorno debe coincidir con los tipos válidos de la expresión.
  - En una sentencia que únicamente contenga la llamada, si no retorna ningún valor (procedimientos).

Ejemplo: Función que calcula el cuadrado de un número

```
# Declaracion de la funcion:
def cuadrado (x):
  """ float  --> float
  OBJ: Calcula el cuadrado de un numero x
  """
  return x * x

# Usos de la funcion...
y = cuadrado (5)
z = 45 + cuadrado (a+b) / 2
print("El cuadrado de 13 es: ", cuadrado(13))
```

Subprogramas sin implementar (aún)

- Python permite postergar la implementación de un subprograma permitiendo utilizarlo en llamadas, etc.

Subprogramas sin implementar (aún)

- Python permite postergar la implementación de un subprograma permitiendo utilizarlo en llamadas, etc.
- Tal vez su implementación aún no se conoce o no se desea especificar

Subprogramas sin implementar (aún)

- Python permite postergar la implementación de un subprograma permitiendo utilizarlo en llamadas, etc.
- Tal vez su implementación aún no se conoce o no se desea especificar
- Sentencia pass:

```
def cuadrado(x): pass
  """ OBJ: Aun no esta claro
  """

# Codigo de prueba
print(cuadrado (5))

>>> None
```

Comunicación

- Los subprogramas se comunican con otras partes del código mediante el paso de argumentos y el retorno de valores

Comunicación

- Los subprogramas se comunican con otras partes del código mediante el paso de argumentos y el retorno de valores
- Un parámetro es un valor que el subprograma espera recibir cuando sea invocado, a fin de ejecutar acciones según dicho valor.

Comunicación

- Los subprogramas se comunican con otras partes del código mediante el paso de argumentos y el retorno de valores
- Un parámetro es un valor que el subprograma espera recibir cuando sea invocado, a fin de ejecutar acciones según dicho valor.
- Un subprograma puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.

Comunicación

- Los subprogramas se comunican con otras partes del código mediante el paso de argumentos y el retorno de valores
- Un parámetro es un valor que el subprograma espera recibir cuando sea invocado, a fin de ejecutar acciones según dicho valor.
- Un subprograma puede esperar uno o más parámetros (que irán separados por una coma) o ninguno.
- Al concluir su cometido el subprograma puede (o no) devolver uno o más valores al módulo que los invocó.

Tipo de argumentos

- Los argumentos o parámetros pueden ser:
  - Formales. Los que se definen en la cabecera del subprograma.
  - Reales o actuales. Los que se especifican en la llamada al subprograma.

Tipo de argumentos

- Los argumentos o parámetros pueden ser:
  - Formales. Los que se definen en la cabecera del subprograma.
  - Reales o actuales. Los que se especifican en la llamada al subprograma.
- Hay correspondencia biunívoca entre los parámetros reales y formales: Deben coincidir en número, semántica y tipo esperado.

Tipo de argumentos

- Los argumentos o parámetros pueden ser:
  - Formales. Los que se definen en la cabecera del subprograma.
  - Reales o actuales. Los que se especifican en la llamada al subprograma.
- Hay correspondencia biunívoca entre los parámetros reales y formales: Deben coincidir en número, semántica y tipo esperado.
- Pueden no existir (en Python se conservan los paréntesis)

Ejemplo de subprogramas sin argumentos

```
# Declaracion del subprograma
def muestra_fecha_hora ():
    """ OBJ : Muestra en pantalla la fecha y hora actuales
    """
    import time
    print ('Fecha : ', time.strftime ("%x"))
    print ('Hora  : ', time.strftime ("%H:%M:%S"))

# Uso del subprograma:
muestra_fecha_hora()
```

Parámetros no posicionales

En Python es posible invocar subprogramas especificando el nombre y valor de los parámetros aunque no estén posicionalmente colocados (parámetros de tipo keyword):

```
def ejemplo (a,b):
 """ OBJ : Funcion tonta para ejemplos
 """
 print('a: ', a)
 print('b: ', b)

#probador
ejemplo(8,9)
>> a: 8 b: 9
ejemplo(b=90, a=9)
>> a: 9 b: 90
```

Comunicación entre módulos: retorno

- Un subprograma que realiza un cómputo retorna un valor (**función**):

```
def hipotenusa (a,b):
    """ float,float-->float
    OBJ: hipotenusa del triangulo rectangulo de lados a,b
    PRE: a,b reales positivos
    """
    from math import sqrt
    return sqrt(a**2+b**2)
```

Comunicación entre módulos: retorno

- Un subprograma que realiza un cómputo retorna un valor (**función**):

```
def hipotenusa (a,b):
    """ float,float-->float
    OBJ: hipotenusa del triangulo rectangulo de lados a,b
    PRE: a,b reales positivos
    """
    from math import sqrt
    return sqrt(a**2+b**2)
```

- Los **procedimientos** imprimen, muestran, ordenan, etc. pero no se espera de ellos un valor como retorno:

```
# Declaracion:
def saludar (nombre):
    """ OBJ : Saludo personalizado a una persona
    """
    print('Hola, ', nombre)

# Uso del procedimiento:
saludar('Maria')
```

Elegir bien el nombre de los subprogramas

- El nombre debe describir claramente lo que hace (si es un procedimiento) o lo que devuelve (si es una función)

Elegir bien el nombre de los subprogramas

- El nombre debe describir claramente lo que hace (si es un procedimiento) o lo que devuelve (si es una función)
- El nombre de un procedimiento debe ser un verbo de acción: `imprimir`, `mostrar`, `ordenar`, `centrar`...

Elegir bien el nombre de los subprogramas

- El nombre debe describir claramente lo que hace (si es un procedimiento) o lo que devuelve (si es una función)
- El nombre de un procedimiento debe ser un verbo de acción: `imprimir`, `mostrar`, `ordenar`, `centrar`...
- En el nombre de las funciones booleanas encajan bien verbos de estado como ser, estar o tener. Ej: `es_bisiesto`, `esta_llena`, `tiene_huecos`

Elegir bien el nombre de los subprogramas

- El nombre debe describir claramente lo que hace (si es un procedimiento) o lo que devuelve (si es una función)
- El nombre de un procedimiento debe ser un verbo de acción: `imprimir`, `mostrar`, `ordenar`, `centrar`...
- En el nombre de las funciones booleanas encajan bien verbos de estado como ser, estar o tener. Ej: `es_bisiesto`, `esta_llena`, `tiene_huecos`
- En aquellas funciones que devuelven la magnitud de una propiedad o una entidad, el nombre será el sustantivo correspondiente a lo que devuelve: `hipotenusa`, `cuadrado`, `raiz_cubica`...

Elegir bien el nombre de los subprogramas

- El nombre debe describir claramente lo que hace (si es un procedimiento) o lo que devuelve (si es una función)
- El nombre de un procedimiento debe ser un verbo de acción: `imprimir`, `mostrar`, `ordenar`, `centrar`...
- En el nombre de las funciones booleanas encajan bien verbos de estado como ser, estar o tener. Ej: `es_bisiesto`, `esta_llena`, `tiene_huecos`
- En aquellas funciones que devuelven la magnitud de una propiedad o una entidad, el nombre será el sustantivo correspondiente a lo que devuelve: `hipotenusa`, `cuadrado`, `raiz_cubica`...

El ámbito de las variables

- Variables **locales**: Se emplean dentro de un subprograma.
  - Sólo son accesibles en el subprograma en que se utilizan.

El ámbito de las variables

- Variables **locales**: Se emplean dentro de un subprograma.
  - Sólo son accesibles en el subprograma en que se utilizan.
- Variables **globales**: Se definen fuera de cualquier subprograma.
  - Accesibles desde cualquier lugar salvo que su nombre sea ocultado por otra variable dentro de un subprograma.

El ámbito de las variables

- Variables **locales**: Se emplean dentro de un subprograma.
  - Sólo son accesibles en el subprograma en que se utilizan.
- Variables **globales**: Se definen fuera de cualquier subprograma.
  - Accesibles desde cualquier lugar salvo que su nombre sea ocultado por otra variable dentro de un subprograma.
- El **acceso global a una variable** consiste en acceder a una variable perteneciente al subprograma llamante.
  - Especialmente desaconsejado si es en escritura.
  - ¡Pueden generar efectos laterales indeseados!

Ejemplo: uso no recomendado de variables globales

```
def f():
    """ nada -> nada
    OBJ: Ejemplo de uso no recomendado de variables globales
    """
    a=20
    media = (a + b)/2
    print('Media: ', media)

a=1
b=10
f()

# Salida:
>>>
Media:  15.0
```

El paso de argumentos en los lenguajes de programación

- En general hay 2 enfoques: por valor (entrada) y por referencia (entrada-salida).

El paso de argumentos en los lenguajes de programación

- En general hay 2 enfoques: por valor (entrada) y por referencia (entrada-salida).
- El paso **por valor** implica copia: el subprograma hace una copia local del argumento real y lo utiliza. Si modifica su copia local, el original no se altera.

El paso de argumentos en los lenguajes de programación

- En general hay 2 enfoques: por valor (entrada) y por referencia (entrada-salida).
- El paso **por valor** implica copia: el subprograma hace una copia local del argumento real y lo utiliza. Si modifica su copia local, el original no se altera.
- En el paso **por referencia** el subprograma usa el valor original mediante un nombre (referencia) local. Si se modifica el objeto a través de su referencia, el original se modifica.

Paso de argumentos en Python

- En Python, los subprogramas reciben parámetros que pueden ser tratados como de entrada o de entrada-salida.

Paso de argumentos en Python

- En Python, los subprogramas reciben parámetros que pueden ser tratados como de entrada o de entrada-salida.
- Si queremos que sean de **entrada**, únicamente aparecerán en la lista de argumentos.

Paso de argumentos en Python

- En Python, los subprogramas reciben parámetros que pueden ser tratados como de entrada o de entrada-salida.
- Si queremos que sean de **entrada**, únicamente aparecerán en la lista de argumentos.
- Si, en cambio, deben ser de **entrada-salida**, aparecerán también en el retorno del subprograma y recogeremos su resultado en el código llamante.

Paso de argumentos en Python

- En Python, los subprogramas reciben parámetros que pueden ser tratados como de entrada o de entrada-salida.
- Si queremos que sean de **entrada**, únicamente aparecerán en la lista de argumentos.
- Si, en cambio, deben ser de **entrada-salida**, aparecerán también en el retorno del subprograma y recogeremos su resultado en el código llamante.

```
def incrementar(x,n):
    """ int, int --> int
    OBJ: Incrementa el entero x en n unidades """
    return x+n;

#Probador
a = 10
print (a)
a = incrementar(a,2)
print (a)
```

Paso de argumentos en Python: otro ejemplo

```
def funcionPythonianaCorrecta(a,c):  #TODO LO QUE ENTRA
    """ int,int --> int,int """
    b=2*a+c                  #usa todo lo que entra
    c=2+c                    #modifico todo lo que sale
    return b,c               #TODO LO QUE SALE

#Probador
a=2
b=3
c=4
print('VALOR/REFERENCIA EN PYTHON')
print('\t E \tS\t E/S')
print('antes   a=',a,'\t b=',b,'\t c=',c)
b,c=funcionPythonianaCorrecta(a,c)
print('despues a=',a,'\t b=',b,'\t c=',c)
```

Bibliotecas

- Permiten agrupar funciones y otros elementos para que las puedan utilizar otros programas.

Bibliotecas

- Permiten agrupar funciones y otros elementos para que las puedan utilizar otros programas.
- Agrupación por funcionalidad relacionada: operaciones con cadenas, operaciones matemáticas, funciones gráficas, etc.

Bibliotecas

- Permiten agrupar funciones y otros elementos para que las puedan utilizar otros programas.
- Agrupación por funcionalidad relacionada: operaciones con cadenas, operaciones matemáticas, funciones gráficas, etc.
- En Python es tan sencillo como almacenarlas juntas en un archivo .py

Bibliotecas

- Permiten agrupar funciones y otros elementos para que las puedan utilizar otros programas.
- Agrupación por funcionalidad relacionada: operaciones con cadenas, operaciones matemáticas, funciones gráficas, etc.
- En Python es tan sencillo como almacenarlas juntas en un archivo .py
- Se interpretan por separado

Bibliotecas

- Permiten agrupar funciones y otros elementos para que las puedan utilizar otros programas.
- Agrupación por funcionalidad relacionada: operaciones con cadenas, operaciones matemáticas, funciones gráficas, etc.
- En Python es tan sencillo como almacenarlas juntas en un archivo .py
- Se interpretan por separado
- Las bibliotecas se importan en los programas que las van a usar:

```
import math #pone disponible los elementos de math
print(math.pi)
# otro enfoque...
from math import pi #hace disponible un elemento unicamente
print(pi)
```

Resumen

- Los diseños modulares tiene muchas e importantes ventajas frente a los no modulares.
- Es deseable que los subprogramas tengan poco acoplamiento y mucha cohesión.
- Las bibliotecas permiten agrupar subprogramas, constantes y otros elementos reutilizables.
- En función de dónde se definan las variables y de cómo se puede acceder a ellas son globales o locales.
- Los subprogramas se comunican con el resto del programa a través de los argumentos y los valores de retorno (concepto de contrato).
- En Python debe tenerse muy en cuenta el carácter mutable o inmutable del argumento de cara a la comunicación entre módulos.

Referencias y reconocimientos

- Algunos de los contenidos de esta presentación han sido adaptados de los materiales del curso de "Programming for Everybody (Python)", creado por Charles Severance y disponible en `https://www.coursera.org/course/pythonlearn`.