INF/LAB 111

M.sc. Jorge Terán Pommier

2019

Universidad Mayor de San Andrés

Contenido - Tema 8

1. Funciones y procedimientos
2. Aplicar estos conceptos a programación
3. Funciones
4. Ámbito/alcance de las variables
5. Alcance local y global
6. Funciones como argumentos
7. Recursión
8. Funciones Lambda

Funciones y procedimientos

Buena programación

- Mas código no necesariamente es una buena estrategia
- Un buen programador se mide por la funcionalidad
- Introducimos **funciones**
- Mecanismo para incorporar **descomposición** y **abstracción**

Ejemplo -Proyector

- Un proyector es una caja negra
- No se como funciona
- Conozco la interfaz entrada / salida
- Conecto cualquier dispositivo electrónico que pueda conectarse con esa entrada
- La caja negra proyecta la imagen amplificada a la pared
- La idea de abstracción no necesito conocer como funciona un proyecto para utilizarlo

Descomposición

- Una imagen grande que proyecta los juegos olímpicos se descompone en diferentes tareas para diferentes proyectores
- Cada proyector recibe una entrada y produce una salida separada
- todos los proyectores trabajan juntos para producir una imagen más grande
- la idea detrás de descomposición diferentes dispositivos trabajan juntos para obtener un objetivo final

Aplicar estos conceptos a programación

Crear una estructura con descompisición

- en el ejemplo del proyector, diferentes dispositivos
- en programación dividimos el código en módulos
  - son auto contenidos
  - se utilizan para dividir el código
  - la intención es que sean reusables
  - mantener el código organizado
  - mantener el código coherente
- En este curso llevamos descomposición con funciones, en cursos siguientes aprenderán con clases

Suprimir detalles con abstracción

- En el ejemplo de proyector es suficiente tener las intrucciones de uso, no hay que saber construirlo
- En programación piense, un pedaso de código como una caja negra
  - no puede ver los detalles
  - no necesita ver los detalles
  - no quiere ver los detalles
  - ocultar detalles tediosos de programar
- obtener abstracción mediante especificación de funciones o docstrings

Funciones

Funciones

- Escriba pedazos/partes de codigo, llamados funciones
- Las funciones en un programa no se ejecutan hasta que son llamadas o invocadas por un programa
- Características de una función:
  - tiene un nombre
  - tiene parámetros (0 o mas)
  - tiene un docstring (opcional pero recomendado)
  - tiene un cuerpo
  - devuelve algo

Como llamar/invocar una función

```
def es_par( i ):
        """
        Input: i, un entero positivo
        Returns True si i es par, sino False
        """
        print('llamo a es_par')
        return i%2 == 0
es_par(3)
```

- identifique: keyword, nombre de la función, argumentos, especificación docstring, cuerpo de la función, llamada a la función
- palabra clave para devolver un resultado. ¿ Donde se evalúa?

Ámbito/alcance de las variables

Buena programación

```
def f( x ):
        x = x + 1
        print('en␣f(x):␣x␣=', x)
        return x
x = 3
z = f( x )
```

- el parámetro formal se convierte en parámetro actual cuando la función se llama
- el programa principal asigna un valor a x, luego guarda el resultado de la función en z.
- un nuevo ámbito se crea cuando se ingresa a la función
- ámbito es asignar nombres a objetos

Alcance local y global

Alcance de las variables

```
def f( x ):
        x = x + 1
        print('en␣f(x):␣x␣=', x)
        return x
x = 3
z = f( x )
```

- Describa el las variables alcance local y global

Advertencia ausencia de return

```
def es_par( i ):
        print('llamo a es_par)
        i %2 == 0
is_even(3)
```

- Python devuelve None si no se uso return
- representa la ausencia de valor

print vrs return

return	print
• return solo tiene sentido dentro de la función • solo se ejecuta un return en la función • el código que sigue a la instrucción return no se ejecuta • tiene un valor asociado con el que invoca a la función	• print puede ser usado fuera de las funciones • puede usar múltiples instrucciones print en las funciones • el código des pues de un print se ejecuta en una función • tiene una valor asociado que es enviado a la consola

Funciones como argumentos

Funciones como argumentos

PROGRAMA

```
def func_a():
    print ('dentro de func_a')
def func_b(y):
    print ('dentro de func_b')
    return y
def func_c(z):
    print ('dentro de func_c')
    return z()
print (func_a())
print (5 + func_b(2))
print (func_c(func_a))
```

Salida

```
dentro de func_a
None
dentro de func_b
7
dentro de func_c
dentro de func_a
None
```

Ejemplo de alcance

- Una función puede acceder a variables definidas fuera de la función
- No puede modificar variables definidas fuera de la función, sin embargo puede utilizar variables globales

```
def f(y):
        x = 1
        x += 1
        print(x)
x = 5
f(x)
print(x)
```

¿Cual es el valor final de x?

```
def g(y):
        print(x)
        print(x +
            1)
x = 5
g(x)
print(x)
```

¿Puede acceder al valor de x?

```
def h(y):
        x += 1
x = 5
h(x)
print(x)
```

¿ Cual es el error?

Funciones como argumentos

PROGRAMA

```
def g(x):
        def h():
                x = 'abc'
        x = x + 1
        print('g: x =', x)
        h()
        return x
x = 3
z = g(x)
```

Describa las variables, el alcance global, el alcance en g(x) y h(x). ¿ Como se llega al resultado final?

Ejemplo de diseño incremental

- Se ingresan por teclado dos puntos que son el centro de un circulo y en punto en su circunferencia
- Calcular el área de dicho circulo.
- Utilizar descomposición y composición para desarrollar el programa
  - Crear una función distancia
  - Crear función área
  - Componer para crear función área de circulo

Recursión

Recursión

- Las técnicas de programación basadas en ciclos se denominan programas iterativos.
- Los programas que describen acciones repetitivas basadas en llamarse a si mismo, se denominan programas recursivos.

Recursión

Los problemas recursivos los representamos con ecuaciones de recurrencia, y de la misma forma en la que se presentan estas ecuaciones los programas recursivos deben tener dos partes:

Caso base El caso base debe ser tan simple que puede ser resuelto sin una llamada recursiva.

Caso recursivo El caso recursivo involucra convertir el problema a otro más simple que puede ser resuelto por una llamada recursiva, es decir llamandose a si mismo.

Modelo para construir programas recursivas

```
if parametros == algun_valor:
    # caso base que termina la recursión
    .........
else:
    # caso recursivo
    .........
```

Cálculo del factorial

Definición:

$$factorial(n) = \begin{cases} 1 & \text{si } n = 0 \\ n * factorial(n-1) & \text{si } n > 0 \end{cases} \quad (1)$$

Demostrar que factorial de cero es uno.

Invertir los datos de entrada

- Supongamos que tenemos un archivo de texto que esta formado por palabras cada una en una línea. Se quiere imprimir estas palabras en orden inverso.
- Para resolver esto podemos guardar las palabra en una estructura de datos que haga el trabajo. Por ejemplo en una estructura de pila.
- Sin explícitamente utilizar ninguna estructura de datos podemos resolver el problema utilizando la recursión.
- Escriba el programa que haga esto.

Máximo común divisor

Uno de los algoritmos más antiguos proviene de Euclides. Este algoritmo es para calcular el máximo común divisor de dos números positivos, y esta definido por la ecuación de recurrencia 2:

$$MCD(a,b) = \begin{cases} a & \text{si } b = 0 \\ MCD(b, a\,\%\,b) & \text{Otros casos} \end{cases} \tag{2}$$

Ejercicios

- Escriba un programa que escriba la representación octal de un número. Para la descomposición debe utilizar recursión.
- Escriba un programa que escriba la representación binaria de un número de 16 bits. Para la descomposición debe utilizar recursión.
- Escriba un programa recursivo que imprima la suma de los dígitos de un número.

Variables globales

```
def fib( x ):
        global numllamadas
        numllamadas+=1
        if x==0 or x==1:
                return 1
        else :
                return fib(x-1)+fib(x-2)
def testFib(n):
        for i in range(n):
                global numllamadas
                nunllamadas=0
                print('fib ',i,'=',fib(i),end=' ')
                print('Numero de llamadas recursivas',
                      numllamadas)
numllamadas=0
testFib(10)
```

- Cuente el numero de llamadas recursivas que se requieren para hallar un numero de Fibonacci
- Para esto creamos una variable global

Funciones Lambda

Funciones Lambda

Sintaxis:

```
lambda argumentos : expresion
```

Ejemplo:

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

Porque usar funciones Lambda

El poder de lambda se muestra mejor cuando se usa como una función anónima dentro de otra función.

Digamos que tiene una definición de función que toma un argumento, y ese argumento se multiplicará con un número desconocido:

```
def mifunc(n):
  return lambda a : a * n

midouble = mifunc(2)

print(midouble(11))
```

```
mitriple = mifunc(3)

print(mitriple(11))
```

Ejemplos

Por ejemplo para hacer x**2 podemos crear la siguiente función:

```
def cuad(x):
  return x * x
```

También es posible hacer:

```
cuad=lambda x: x*x
```

Ejemplos

Para convertir una lista de números a sus cuadrados

```
lista=[1,2,3,4]
def cuad(x):
  return x * x
listaCuad=map(cuad,lista)
```

También es posible hacer:

```
lista=[1,2,3,4]
cuad=lambda x: x*x
listaCuad=map(cuad,lista)
```

Y finalmente:

```
lista=[1,2,3,4]
listaCuad=map(lambda x: x*x,lista)
```

Ejemplos Fibonacci

Se puede definir una función abstracta de Fibonacci

```
def fib():
  a,b = 0,1
  while True:
    yield a
    a, b = b, a + b
```

La instrucción *yield* establece la variable retorno.

Para listar los números Fibonacci:

```
for valor in fib():
  print(valor)
  if valor> 100:
    break
```

Se necesita este if y break para que no de vueltas infinitamente.

Ejemplos Fibonacci

Esta función puede generar números en secuencia uno a la vez

```
def fib():
  a,b = 1,0
  while True:
    yield a
    a, b = b, a + b
```

La instrucción *next* permite hallar el siguiente de la secuencia.

Para listar los números Fibonacci y /o crear una lista

```
f=fib()
for i in range(10):
    print(next(f))

l=[next(f) for x in range(10)]
print(l)
```