Universidad de Alcalá

FUNDAMENTOS DE LA PROGRAMACIÓN

Prueba de evaluación continua PECT1 – Grado en Ingeniería Informática

SOLUCIONES

1) Cumplimente los objetivos del siguiente subprograma y elija un nombre adecuado para el mismo (1 punto):

```
def subprograma(n):
    """ ¿?? -> ¿??
         OBJ: ??????
        PRE: 101 <= n <= 999
    """
    m=0
    for i in range(1,4):
        d = n%10
        n = n//10
        m = m*10 + d
    return m
```

SOLUCIÓN:

```
int -> int
OBJ: invierte un número de 3 cifras
subprograma => invertido3cifras, invertir_cifras, etc.
```

2) El siguiente código busca calcular cuántos divisores comunes tienen dos números enteros. Complete los huecos para que funcione adecuadamente (1,5 puntos):

```
def mcd(x,y):
    i = ____________________
    while (i<=x and i<=y):
        if(x%i==0 and y%i == 0):
            gcd = _______________
        i += _______________
    return gcd;

def num_comm_div(x, y):
  n = mcd(_______________)
  result = 0
  z = int(n**0.5)
  i = 1
  while( ________________ ):
    if(n % i == 0):
      result += 2
      if(i == n/i):
        result-=1
    i+=1
  return result

print("Number of common divisors: ",num_comm_div(360, 336))
```

SOLUCIÓN:

```
i = 1
gcd = i
```

```
        i += 1
        while ( i <= z )
```

3) Indique cuál sería la salida por pantalla del siguiente código (1,5 puntos):

```
def piirtaa_lippu (n,m):
    """ int, int --> None
    OBJ: this has been removed, sorry for the inconvenience """
    for k in range(m):
        for i in range(1,n+1):
            print (i*'* ')
        for j in range(n-1,0,-1):
            print (j*'* ')

piirtaa_lippu (4,5)
```

SOLUCIÓN:

Muestra en pantalla una bandera triangular de n asteriscos con tantos triángulos como indique el número m.

4) Complete el *docstring* y explique el funcionamiento del siguiente subprograma (1,5 puntos):

```
def hullu_silmukka(k,kk):
    """
        OBJ: Write here!
    """
    for k1 in range (0,k+1):
        if k1==0:
            for kk2 in range (1,kk+1): print('\t', kk2*'@',end='')
        else:
            print (k1*'|', end='\t')
            for kk2 in range (1,kk+1): print(k1*'|','-',kk2*'@',end='\t')
        print()

hullu_silmukka(3,4)
```

SOLUCIÓN:

```
“”” int, int -> None
      OBJ: Muestra por pantalla una tabla como la siguiente:

            @           @@           @@@          @@@@
|       | - @       | - @@       | - @@@       | - @@@@
||     || - @       || - @@       || - @@@     || - @@@@
|||   ||| - @       ||| - @@     ||| - @@@    ||| - @@@@

”””””
```

5) Observe el siguiente código. ¿Es correcto? Si no lo es, enumere todos y cada uno de los errores que crea que contiene especificando claramente si son de tiempo de compilación, de tiempo de ejecución o de otro tipo, y por qué considera usted que es un error (1 punto).

```
def cifras(a):
    x=0
    n=a
    while (n>0):
        if (a%10)>0:
            x=x+1
```

```
            a=a//10
            n=a
        return x

    def cifras2(a):
        x=0
        n=a
        while (n>0):
            if (a%10)>0:
                x=x+1
            a=a//10
            n=a
        return x

    x1 = int(input('Introduce un número: '))
    x2 = int(input('Introduce otro número: '))
    if cifras(x1) > cifras2(x2): print('El primero tiene más cifras')
    else: print('El segundo tiene más cifras')
```

SOLUCIÓN:

- El error más grave es en el diseño del algoritmo, pues resulta evidente su falta de modularidad: es innecesario codificar 2 funciones iguales, basta con invocar tantas veces a una única función como sea necesario.
- No funciona bien porque no cuenta los ceros: error en tiempo de ejecución (bug)
- La variable n es innecesaria, lo cual no es un error, pero sí es mejorable
- No funciona si a es cero: error en tiempo de ejecución (bug)

6) Observe el siguiente código y elija una opción entre las propuestas (0,5 puntos)

```
def lado_triangulo(a,b,c):
    """float,float,float-->float
       OBJ: introducir dos lados y calcular el lado que falta
    """
    if a==0:
        import math
        return ("lado =", math.sqrt(c**2-b**2))
    else return math.sqrt(c**2+b**2))
```

a) **No contiene errores**
b) **Es erróneo, porque el docstring no refleja lo que hace el código**
c) **Es erróneo, porque el retorno en el "if" no concuerda con lo especificado en su *docstring***
d) **El código contiene múltiples errores**

SOLUCIÓN:

d) Tiene múltiples errores

7) En un examen de programación se pide escribir una función que calcule, utilizando el teorema de Pitágoras, el lado que le falta a un triángulo rectángulo. Quien invoca a la función pone un cero en el lado que falta y da valores para los lados que conoce. Alguien implementa esta solución, pero lamentablemente contiene errores. Su papel consiste en detectar y explicar brevemente todos los errores e incorrecciones (1 punto)

```
def pitagoras(a,b,c):
    """float,float,float --> float
    OBJ: calcula el lado del triángulo rectángulo que falta. Las variables a y b
         representan los catetos, c la hipotenusa
    PRE: a,b,c>=0 """
    if not (a>=0 and b>=0 and c>=0): return ('Error!!')
    else:
       if a==0: a = pow(((pow(c,2))-(pow(b,2))),0.5)
       if b==0: b = pow(((pow(c,2))-(pow(a,2))),0.5)
       if c==0: c = pow(((pow(a,2))+(pow(b,2))),0.5)
       return print("Los 3 lados son:",a, ", ", b,"y ",c)
```

SOLUCIÓN:

- Es innecesario codificar una comprobación para algo que se asegura mediante una precondición. En este caso: not (a>=0 and b>=0 and c>=0)
- En lugar de 3 ifs sucesivos debería haberse empleado una estructura selectiva múltiple
- Return print retorna siempre None, no float
- Los retornos en las dos ramas del if son de distinto tipo, uno es str y otro None, y además no son coherentes con el retorno declarado: float.

8) El siguiente programa trata de obtener una aproximación del número π utilizando para ello la serie de Leibniz:

$$\pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + 4/13 - 4/15 \ldots$$

```
1.  iteraciones = int(input("¿Cuántas iteraciones? "))
2.  signo = -1
3.  pi = 0
4.  for n in range(1, iteraciones+1):
5.      pi = pi+4/(n*2+1)*signo
6.      signo += -1
7.  print("Pi: ", pi)
```

Lamentablemente, se ha hecho un seguimiento y se han detectado errores. Se pide:

a) Corregir los errores para que el programa cumpla su función, indicando sólo qué línea(s) contiene(n) el error y reescribiendo dicha(s) línea(s). (1,5 puntos)

b) ¿Qué habría que modificar para que, en lugar de solicitar el número de iteraciones, el programa pare cuando el error de la aproximación con respecto al valor real de π sea menor a 0,001 ? (0,5 puntos)

SOLUCIÓN:

a)

```
2.  signo = 1
5.  pi = pi + 4/(n*2-1)*signo
6.  signo *= -1
```

b)

Eliminar línea 1 y sustituir la 4 así:

```
4.  while (abs (4/(n*2+1)*signo)) > 0.001):
```