

PRACTICO 5 - HASKELL

- 1) `maximo :: (Integer, Integer) -> Integer`
`maximo (a,b) | a>b = a`
`| b>a = b`
`| otherwise = error "Los dos argumentos son iguales"`

- 2) `par :: Integer -> Bool`
`par a | a `mod` 2==0 = True`
`| otherwise = False`

- 3) `max3 :: (Integer, Integer, Integer) -> Integer`
`max3 (a,b,c) | a>b && a>c = a`
`| a==b && a>b = a`
`| b>a && b>c = b`
`| b==c && b>a = b`
`| c>a && c>b = c`
`| a==c && a>b = c`
`| a==b && b==c = a`

`max_3 :: (Float, Float, Float) -> Float`
`max_3 (a,b,c) | a>b && a>c = a`
`| a==b && a>b = a`
`| b>a && b>c = b`
`| b==c && b>a = b`
`| c>a && c>b = c`
`| a==c && a>b = c`
`| a==b && b==c = a`

- 4) `sgn :: Int -> Int`
`sgn a | a>0 = 1`
`| a<0 = -1`
`| a==0 = 0`

- 5) `abs :: Int -> Int`
`abs a | a>0 = a`
`| a<0 = -a`
`| a==0 = 0`

`abs2 :: Integer -> Integer`
`abs2 a = (sgn a)*a`

- 6) `bisiesto :: Int -> Bool`
`bisiesto a | a `mod` 4==0 && a `mod` 100/=0 || mod a 400 == 0 = True`
`| otherwise = False`

- 7) `lados_triangulo :: (Float, Float, Float) -> Bool`
`lados_triangulo (a,b,c) | max_3 (a,b,c) < a + b + c - max_3 (a,b,c) = True`
`| otherwise = False`

`lados_triangulo2 :: (Float, Float, Float) -> String`

lados_triangulo2 (a,b,c) | max_3 (a,b,c) < a + b + c - max_3 (a,b,c) = "CON ESTAS MEDIDAS SE PUEDE FORMAR UN TRIANGULO"

| otherwise = "IMPOSIBLE FORMAR UN TRIANGULO CON

ESTAS MEDIDAS"

clasif_triangulo :: (Float, Float, Float) -> String

clasif_triangulo (a,b,c) | lados_triangulo (a,b,c)==True && a/=b && a/=c && b/=c = "El triangulo es escaleno"

| lados_triangulo (a,b,c)==True && a==b && b==c = "El triangulo es equilatero"

| lados_triangulo (a,b,c)==True && otherwise = "El triangulo es isosceles"

| otherwise = "No es posible construir un triangulo con estas medidas"

8) es_rectangulo :: (Integer, Integer, Integer) -> Bool

es_rectangulo (a,b,c) | max3 (a,b,c)*max3 (a,b,c) == a*a + b*b + c*c - max3 (a,b,c)*max3 (a,b,c) = True

| otherwise = False

es_rectangulo2 :: (Float, Float, Float) -> String

es_rectangulo2 (a,b,c) | lados_triangulo (a,b,c)==True && max_3 (a,b,c)*max_3 (a,b,c) == a*a + b*b + c*c - max_3 (a,b,c)*max_3 (a,b,c) = "El triangulo es rectangulo"

| lados_triangulo (a,b,c)==True = "El triangulo no es equilatero"

| otherwise = "No se puede formar triangulo"

9) area_circulo :: Float -> Float

area_circulo r | r>0 = r*r*22/7

| r==0 = 0

| otherwise = error "El radio debe ser un numero positivo"

10) num_raices :: (Float, Float, Float) -> String

num_raices (a,b,c) | b*b-4*a*c > 0 = "La ecuacion polinomica tiene 2 raices reales diferentes"

| b*b-4*a*c == 0 = "La ecuacion polinomica tiene 1 raiz real doble"

| b*b-4*a*c < 0 = "La ecuacion polinomica no tiene raices reales"

11) f :: (Integer, Integer) -> Integer

f(0,0) = 1

f(0,1) = 2

f(0,y) = y + 2

f(x+1,0) = 1

f(x+1,y+1) = f(x, f(x+1,y))

f(3,2)

f(2, f(3,1))

f(2, f(2, f(3,0)))

f(2, f(2, 1))

f(2, f(1, f(2, 0)))

f(2, f(1, 1))

f(2, f(0, f(1, 0)))

f(2, f(0,1))

f(2, 2)

f(1, f(2,1))

f(1, f(1, f(2,0)))

f(1, f(1, 1))

f(1, f(0, f(1,0)))

f(1, f(0, 1))

f(1, 2)

f(0, f(1,1))

f(0, f(0, f(1,0)))

f(0, f(0, 1))

f(0, 2)

4

- 12) (`|||`) :: `Bool -> Bool -> Bool`
`x ||| y` | `x==False && y==True = True`
| `x==True && y==False = True`
| `otherwise = False`
- 13) (`|||`) :: `Bool -> Bool -> Bool`
`x ||| y` | `(x || y) && not(x && y) == True = True`
| `otherwise = False`
- 14) `max2` :: `Integer -> Integer -> Integer`
`max2 a b` | `a>=b = a`
| `b>a = b`
`max3` :: `Integer -> Integer -> Integer -> Integer`
`max3 a b c` | `max a b >= c = max a b`
| `c > max a b = c`

`max4` :: `Integer -> Integer -> Integer -> Integer -> Integer`
`max4 a b c d` | `max2 a b >= max2 c d = max2 a b`
| `max2 c d > max a b = max2 c d`
- 15) `tresDiferentes` :: `Integer -> Integer -> Integer -> Bool`
`tresDiferentes a b c` | `a/=b && a/=c && b/=c = True`
| `otherwise = False`
- 16) `cuatrolguales` :: `Integer -> Integer -> Integer -> Integer -> Bool`
`cuatrolguales a b c d` | `a==b && b==c && c==d = True`
| `otherwise = False`
- 17) `media3` :: `Float -> Float -> Float -> Float`
`media3 a b c = (a+b+c)/3`
- 18) `sobreMedia3` :: `Float -> Float -> Float -> String`
`sobreMedia3 a b c`
| `a>media3 a b c && b>media3 a b c = "Exactamente 2 valores son mayores que la media"`
| `a>media3 a b c && c>media3 a b c = "Exactamente 2 valores son mayores que la media"`
| `b>media3 a b c && c>media3 a b c = "Exactamente 2 valores son mayores que la media"`
| `a==b && b==c && a==media3 a b c = "Los 3 valores son iguales a la media"`
| `otherwise = "Solamente 1 valor es mayor que la media"`
- 18) `divideA` :: `Integer -> Integer -> Bool`
`divideA x y` | `mod x y == 0 = True`
| `otherwise = False`
- 19) `maximo2` :: `Int -> Int -> Int`
`maximo2 x y = div ((x+y) + abs(x-y)) 2`
- 20) `maximo2` :: `Integer -> Integer -> Integer`
`maximo2 x y = div ((x+y) + abs(x-y)) 2`

`maximo3` :: `Integer -> Integer -> Integer -> Integer`
`maximo3 x y z = maximo2 (maximo2 x y) z`

```
minimo2 :: Integer -> Integer -> Integer
minimo2 x y = div ((x+y) - abs (x-y)) 2
```

```
minimo3 :: Integer -> Integer -> Integer -> Integer
minimo3 x y z = minimo2 (minimo2 x y) z
```

```
maximo4 :: Integer -> Integer -> Integer -> Integer -> Integer
maximo4 x y z a = maximo2 (maximo2 x y) (maximo2 z a)
```

```
21) entre0y9 :: Integer -> Bool
    entre0y9 x = (0 <= x) && (x <= 9)
```

```
22) esMultiploDe3 :: Integer -> Bool
    esMultiploDe3 a | mod a 3==0 = True
                   | otherwise = False
```

```
23) descomponer :: Integer -> (Integer, Integer, Integer)
    descomponer a = (div a 3600, div (a - (div a 3600)*3600) 60, a - (div a 60)*60)
```

```
descomponer :: Integer -> (Integer, Integer, Integer)
descomponer a = (h,m,s)
    where h = div a 3600
          m = div a 60 - h*60
          s = a - m*60 - h*3600
```

```
24) incTupla3 :: (Integer, Integer, Integer) -> (Integer, Integer, Integer)
    incTupla3 (a,b,c) = (a+1,b+1,c+1)
```

```
25) ordena3 :: Integer -> Integer -> Integer -> (Integer, Integer, Integer)
    ordena3 a b c = (minimo3 a b c, a + b + c - minimo3 a b c - maximo3 a b c, maximo3 a b c)
```

```
ordena3 :: (Integer, Integer, Integer) -> (Integer, Integer, Integer)
ordena3 (a,b,c) | max3 (a,b,c)==a && b>c = (c,b,a)
                | max3 (a,b,c)==a && c>b = (b,c,a)
                | max3 (a,b,c)==a && c==b = (b,c,a)
                | max3 (a,b,c)==b && a>c = (c,a,b)
                | max3 (a,b,c)==b && c>a = (a,c,b)
                | max3 (a,b,c)==b && c==a = (a,c,b)
                | max3 (a,b,c)==c && a>b = (b,a,c)
                | max3 (a,b,c)==c && b>a = (a,b,c)
                | max3 (a,b,c)==c && b==a = (a,b,c)
```

```
26) esCapicua :: Integer -> String
    esCapicua a | div a 1000 > 0 && div a 1000 < 10 && (div a 1000) == (a - (div a 10)*10) &&
                (div a 100) - (div a 1000)*10 == (div a 10) - (div a 100)*10 = "El numero es capicua"
                | div a 1000 == 0 || div a 1000 > 9 = "El numero ingresado no es de 4 cifras"
                | otherwise = "El numero no es capicua"
```

27) A cargo del lector

Se agradece al estudiante Domingo Pérez del grupo 1º interior del Profesorado de informática por su aporte a las respuestas.