





Funciones import

Clase #11

IIC1103 – Introducción a la Programación

El plan de hoy es...

- Tarea 1 publicada!
- Menti. Hoy con la colaboración del  
- Más sobre funciones!!

Menti



Recuerdo de la clase pasada...

definir
función

nombre de la función

parámetros/argumentos

• **def** segundos(hora):

•
s = hora%100
m = (hora//100)%100
h = (hora//10000)%100
return s+m*60+h*60*60

• variable **LOCAL**!

- instrucciones internas a la función (ojo: deben estar indentadas)
- retornar resultado (con instrucción return)

Recuerdo de la clase pasada...

```
import math
```

instrucciones import

```
def area_circulo(r):  
    return math.pi*r*r  
  
def area_triangulo(a,b,c):  
    s=(a+b+c)/2  
    return math.sqrt(s*(s-a)*(s-b)*(s-c))
```

definiciones de funciones
que usaremos más abajo

```
a=float(input("a? "))  
b=float(input("b? "))  
c=float(input("c? "))  
radio = max(a,b,c)/2  
print("area = "+str(area_circulo(radio)-area_triangulo(a,b,c)))
```

programa principal:
llamadas a funciones

Problema #0 (clase pasada)

- Escribe una función `sumatoria(n)` que entregue la suma de los números desde 1 a `n`. Si `n` es negativo, debe entregar 0.
- Ejemplos:
 - `x = sumatoria(4)` #x vale 10
 - `y = sumatoria(-4)` #y vale 0

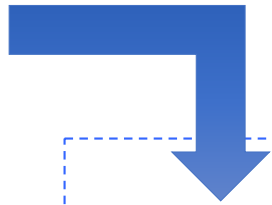
Funciones

- Reciben argumentos/parámetros
- Hacen cálculos
- Retornan/devuelven algún valor



Dentro de la función...

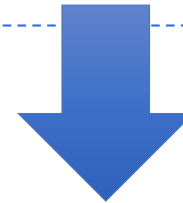
x



sumatoria

Instrucciones usando x
+ cualquier otra
instrucción necesaria

return resultado



resultado

Dentro de la función...

definir

función

nombre de la función

parámetros/argumentos

• `def sumatoria(x):`

- ```
s = 0
for i in range(1,x+1):
 s+=i
return s
```

- variable **LOCAL!**

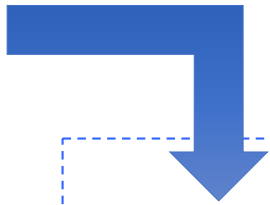
- instrucciones internas a la función (ojo: deben estar indentadas)
- retornar resultado (con instrucción `return`)

## Problema #0b

- Escribe una función `sumatoriap(n)` que **imprima** la suma de los números desde 1 a `n`. Si `n` es negativo, debe **imprimir** 0.
- Ejemplos:
  - `sumatoriap(4)` #imprime 10
  - `sumatoriap(-4)` #imprime 0

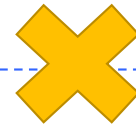
Dentro de la función...

x



sumatoriap

Instrucciones usando x  
+ cualquier otra  
instrucción necesaria



No retorna

# Diferencia entre return y print

- Ver: cómo llamar a sumatoria y a sumatoriap
- Ver: cómo usar el resultado de sumatoria y de sumatoriap

# Problema #1

- Escribe una función `digito(n,i)` que entregue el *i*ésimo dígito del número *n*
- Ejemplos:
  - `x = digito(1567,3)` #x vale 6
  - `y = digito(34,1)` #y vale 3

# Funciones

- Reciben argumentos/parámetros
- Hacen cálculos
- Retornan/devuelven algún valor



# Dentro de la función...

definir

función

nombre de la función

parámetros/argumentos, separados por comas

```
• def digito(x,i):
```

```
• ...código...
 return valor
```

• variable **LOCAL**!

- instrucciones internas a la función (ojo: deben estar indentadas)
- retornar resultado (con instrucción return)

# Problema 2, o, dividir para conquistar

## PRIMOS RELATIVOS

Dos números naturales se llaman primos relativos si el máximo común divisor entre ellos es 1.

Los números 6 y 9 NO son primos relativos ya que los divisores de 6 son 1, 2, 3 y 6. Los divisores de 9 son 1, 3 y 9. Por lo tanto el máximo común divisor es 3.

Los números 9 y 14 son primos relativos ya que los divisores de 9 son 1, 3 y 9, mientras que los divisores de 14 son 1, 2, 7 y 14. Por lo tanto el máximo común divisor es 1.

<http://www.sectormatematica.cl>

- Escribe todos los pares de **primos relativos** entre 2 y 20
  - Asume que existe una función `primosrelativos(x,y)` que entrega True/False dependiendo de si x, y son primos relativos

```
>>>
2 y 3 son primos relativos
2 y 5 son primos relativos
2 y 7 son primos relativos
2 y 9 son primos relativos
2 y 11 son primos relativos
2 y 13 son primos relativos
2 y 15 son primos relativos
2 y 17 son primos relativos
2 y 19 son primos relativos
3 y 4 son primos relativos
3 y 5 son primos relativos
3 y 7 son primos relativos
3 y 8 son primos relativos
3 y 10 son primos relativos
3 y 11 son primos relativos
3 y 13 son primos relativos
3 y 14 son primos relativos
3 y 16 son primos relativos
3 y 17 son primos relativos
```



# Solución

- #Programa Principal
- i=2
- while i<=20:
- j=i+1
- while j<=20:
- if primosrelativos(i,j):
- print(str(i)+" y "+str(j)+" son primos  
relativos")
- j+=1
- i+=1

# Problema #2-b

## PRIMOS RELATIVOS

Dos números naturales se llaman primos relativos si el máximo común divisor entre ellos es 1.

Los números 6 y 9 NO son primos relativos ya que los divisores de 6 son 1, 2, 3 y 6. Los divisores de 9 son 1, 3 y 9. Por lo tanto el máximo común divisor es 3.

Los números 9 y 14 son primos relativos ya que los divisores de 9 son 1, 3 y 9, mientras que los divisores de 14 son 1, 2, 7 y 14. Por lo tanto el máximo común divisor es 1.

<http://www.sectormatematica.cl>

- Escribe la función `primosrelativos(x,y)`

```
>>>
2 y 3 son primos relativos
2 y 5 son primos relativos
2 y 7 son primos relativos
2 y 9 son primos relativos
2 y 11 son primos relativos
2 y 13 son primos relativos
2 y 15 son primos relativos
2 y 17 son primos relativos
2 y 19 son primos relativos
3 y 4 son primos relativos
3 y 5 son primos relativos
3 y 7 son primos relativos
3 y 8 son primos relativos
3 y 10 son primos relativos
3 y 11 son primos relativos
3 y 13 son primos relativos
3 y 14 son primos relativos
3 y 16 son primos relativos
3 y 17 son primos relativos
```

# Solución

- **def** primosrelativos(x,y):  
    n=2  
    **while** n<=min(x,y):  
        **if** x%n==0 **and** y%n==0:  
            **return** False  
        n=n+1  
    **return** True

# Solución completa

```
def primosrelativos(x,y):
 n=2
 while n<=min(x,y):
 if x%n==0 and y%n==0:
 return False
 n=n+1
 return True
#Programa Principal
i=2
while i<=20:
 j=i+1
 while j<=20:
 if primosrelativos(i,j):
 print(str(i)+" y "+str(j)+" son primos relativos")
 j+=1
 i+=1
```

# Solución completa

programa.py

```
def primosrelativos(x,y):
 n=2
 while n<=min(x,y):
 if x%n==0 and y%n==0:
 return False
 n=n+1
 return True

#Programa Principal
i=2
while i<=20:
 j=i+1
 while j<=20:
 if primosrelativos(i,j):
 print(str(i)+" y "+str(j)+" son primos relativos")
 j+=1
 i+=1
```

## O, definiendo un módulo propio

util.py

```
def primosrelativos(x,y):
 n=2
 while n<=min(x,y):
 if x%n==0 and y%n==0:
 return False
 n=n+1
 return True
```

programa.py

```
import util
#Programa Principal
i=2
while i<=20:
 j=i+1
 while j<=20:
 if util.primosrelativos(i,j):
 print(str(i)+" y "+str(j)+" son primos relativos")
 j+=1
 i+=1
```

# Opciones de import

util.py

```
def primosrelativos(x,y):
 n=2
 while n<=min(x,y):
 if x%n==0 and y%n==0:
 return False
 n=n+1
 return True
def suma(x,y):
 return x+y
```

- `import util`
- `print(util.primosrelativos(5,7))`
  
- `from util import primosrelativos`
- `print(primosrelativos(5,7))`
  
- `from util import *`
- `print(primosrelativos(5,7))`
- `print(suma(5,7))`

# Opciones de import

- `import util`
- `print(util.primosrelativos(5,7))`
- `import util as u`
- `print(u.primosrelativos(5,7))`
- `from util import primosrelativos`
- `print(primosrelativos(5,7))`
- `from util import *`
- `print(primosrelativos(5,7))`
- `print(suma(5,7))`

util.py

```
def primosrelativos(x,y):
 n=2
 while n<=min(x,y):
 if x%n==0 and y%n==0:
 return False
 n=n+1
 return True
def suma(x,y):
 return x+y
```



# Resumen de hoy

definir  
función

nombre de la función

parámetros/argumentos

• def area\_triangulo(a,b,c):

s=(a+b+c)/2

return math.sqrt(s\*(s-a)\*(s-b)\*(s-c))

• variable **LOCAL!**

- instrucciones internas a la función (ojo: deben estar indentadas)
- retornar algo (con instrucción return). En el momento del return se sale de la función y vuelve al lugar donde fue llamada

```
import util -> llamada: util.primosrelativos(5,7)
from util import primosrelativos -> llamada: primosrelativos(5,7)
from util import * -> llamada: primosrelativos(5,7)
```