Condiciones

CC1001–Prof. Romain Robbes

Ejercicio precedente & riceta de diseño

```
import  triangulo

help(area_triangulo)
```

Programas necesitan de tomar decisiones

Ejemplo: determinar el tipo de un triangulo de lados a, b, y c

=> equilatero (si a = b = c)
=> isosceles (...)
=> escaleno (...)
=> no es un triangulo (...)

3 números forman un triángulo **si** son positivos y la suma de 2 cualesquiera de ellos es mayor que el 3.

Pero, como podemos tomar decisiones?

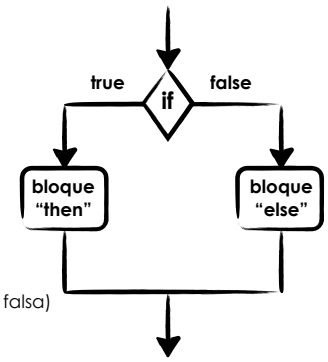
Instrucción **if**

Sintaxis:

```
if condición:
  bloque “then”
else:
  bloque “else”
```

Semántica:

- Si condición se cumple: ejecutar instrucciones1
- Si condición no se cumple (es falsa) ejecutar instrucciones2



El tipo **bool** describe valores de condiciones

constantes:
`True` (verdadero) y `False` (falso)

ejemplos de condiciones:

```
4 < 3
5 == 5
x >= y
x == "hello"
x == True
3 != 4
```

¿ Porqué == ?

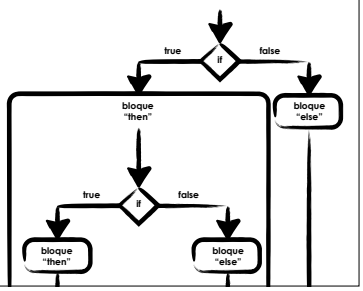
Ejemplo:

```
def max(x, y):
    """
    max(x, y): num num -> num
    devuelve el argumento mayor de los dos
    ejemplo: max(3, 4) devuelve 4
    """
    if x > y:
        return x
    else:
        return y
```

Triangulo equilatero?

```
def tipo_triangulo(a, b, c):
    ...
    if a == b:
        if a == c:
            print "triangulo equilatero"
        else: ...
    else: ...
```

se puede hacer mejor que eso ...



Condiciones compuestas

X and Y

X \ Y	T	F
T	T	F
F	F	F

X or Y

X \ Y	T	F
T	T	T
F	T	F

not X

X	
T	F
F	T

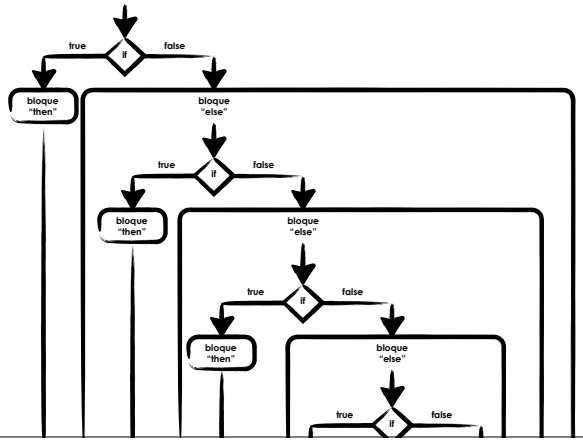
X ∧ Y

X \ Y	T	F
T	F	T
F	T	F

Triangulo equilatero?

```
def tipo_triangulo(a, b, c):
    ...
    if a == b and a == c:
            print "triangulo equilatero"
    else: # isoceles o escaleno o ...
```

Cadenas de "if"



Generalización: Selección múltiple

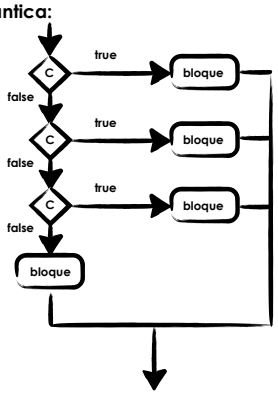
Sintaxis:

```
if cond1:
     insts1
elif cond2:
    insts2
elif ...
...
else:
    insts
```

Ejemplo:

```
n=random.randint(1,3)
if n==1:
  print“piedra”
elif n==2:
  print “papel”
else:
  print “tijeras”
```

Semantica:



Determinar tipo de triangulo a, b, c

```
print "Determinar tipo de triangulo de lados a,b,c"
a=input("a?")
b=input("b?")
c=input("c?")

if a<=0 or b<=0 or c<=0 or a+b<c or a+c<b or b+c<a:
    print "no forman un triángulo"
elif a==b and b==c:
    print "equilatero"
elif a==b or a==c or b==c:
    print "isosceles"
else:
    print "escaleno"
```

Sobre estilo: no mezclar interfaz y codigo de funcionalidad; separar funcionalidad

```
def tipo_triangulo(a, b, c):
    if es_triangulo(a, b, c):
        if es_equilatero(a, b, c):
            return "equilatero"
        elif es_isosceles(a,b, c):
            return "isosceles"
        else:
            return "escaleno"
    else: return "no triangulo"

print "Determinar tipo de triangulo de lados a,b,c"
a=input("a?")
b=input("b?")
c=input("c?")

print tipo_triangulo(a, b, c)
```

Había una error en el codigo!

```
print "Determinar tipo de triangulo de lados a,b,c"
a=input("a?")
b=input("b?")
c=input("c?")

if a<=0 or b<=0 or c<=0 or a+b<c or a+c<b or b+c<a:
    print "no forman un triángulo"
elif a==b and b==c:
    print "equilatero"
elif a==b or a==c or b==c:
    print "isosceles"
else:
    print "escaleno"
```

La nueva receta de diseño

```
def es_equilatero(a, b, c):
        """
        es_equilatero: num num num -> bool
        devuelve True si los 3 lados hacen
        un triangulo equilatero
        ejemplos:
        es_equilatero(1, 2, 3) == False
        es_equilatero(2, 2, 2) == True
        """
        return a == b and a == c

assert not es_equilatero(1, 2, 3)
assert es_equilatero(2, 2, 2)
```

Nombre descriptivos: por foncion, parametros, y variables
Contrato: typos de argumentos y valor de returno
Propósito: qué hace la función **(no como)**
Ejemplos: ejemplos de uso de la función, y resultado esperado
Tests: saber si la función es **correcta**
Cuerpo: instruciones para hacer su computacion (**en ultimo**)

El keyword **assert**

```
assert True #no hace nada
assert False #hace una error
```

Es muy importante saber que hay una error el mas temprano possible

La receta de disegno es obligatoria!

Parece demasiado por ejemplos chicos, pero sirve a acostumbrarse.

Una vez que el codigo crece, se hace muy util para tener un bueno disegno, y para evitar errores.

Si Uds no siguen la receta de diseño, van a perder puntos.

La receta de diseño es inspirada del libro "How to design programs" (muy recomendado!). Se encontra gratis a:

http://htdp.org/

Antes de hacer el ejercicio, algo sobre disciplina

No puedo responder a todas las preguntas, entonces no respondo a ninguna.

Si Ud. tiene una pregunta, pensar mas y mirar el "handout". La capacidad a buscar de su mismo es muy importante en computación (y en la vida).

No se puede levantar de su silla. No se puede hablar tampoco, por el ruido que hace y la molestia por otras clases.

Los ejercicios son hechos para apprender. Tienen un peso muy bajo por las notas. Entonces no es un problema si uno no puede hacer uno perfectamente.

Al final del tiempo, devolver las hojas por adelante.

Ejercicio: comparison de floats

Ahora no se puede hacer facilmente testing de floats.

Desarollar una función cerca(x, y, epsilon)

```
cerca(2, 2.05, 0.1) == True
cerca(2, 2.05, 0.01) == False
```

Se puede usar asi:

```
assert cerca(area_circulo(3), 28.26, 0.01)
```

seguir la receta de diseño