

▼ Clase 4: Expresiones y Funciones Condicionales

(Leer Capítulo 5 del Apunte)

Valores Booleanos

Existen expresiones que al ser evaluadas retornan valores:

- `True` (VERDADERO) o
- `False` (FALSO),

estos valores son de tipo `bool` (booleanos).

Ya las conocemos de las matemáticas en donde hablamos de *proposiciones verdaderas o falsas*.

Para un par de números `x` e `y` sólo existen 3 opciones posibles:

```
x == y # x es igual a y
x < y  # x es menor a y
x > y  # x es mayor a y
```

A lo anterior también podemos agregar:

```
x <= y # x es menor igual a y
x >= y # x es mayor igual a y
```

Podemos probar cómo se comportan estas expresiones en python:

```
6 < 5
```



False

```
5 < 6
```



True

```
4 == 5
```



False

▼ Condiciones compuestas

También existen condiciones compuestas que son conectores lógicos:

- **and**, es True si las expresiones que conecta son ambas True
- **or**, es True si al menos una de las expresiones que conecta son True
- **not** niega el resultado de la expresión que viene inmediatamente a continuación

```
x == y and y < z # es verdadero si x==y es True e y<z es True
x == y or y < z # es verdadero si x==y es True o y<z es True
not x == y # es verdadero si x==y es False (la negación de falso es verdadero)
```

```
x=5
y=5
z=6
x==y and y<z
```

 True

Ahora cambiamos sólo el valor de z:

```
z=4
x==y and y<z
```

 False

Ahora probamos **or** con las mismas expresiones y sólo debería bastar que una de las expresiones fuera **True**:

```
x==y or y<z
```

 True

```
True or False
```

 True

```
True and False
```

 False

▼ Funciones booleanas

También es posible crear funciones cuyo resultado sea del tipo bool (`True` o `False`). Por ejemplo, creamos una función que dice si un valor `n` es igual a 5:

```
# esIgualA5: num -> bool
# determina si n es igual a 5
# ej: esIgualA5(5) devuelve True y para cualquier otro numero False
def esIgualA5(n):
    return n == 5
# test
assert esIgualA5(5)
assert not esIgualA5(11) # como esIgualA5(11) es False, se antepone not
                        # para que sea True y funcione el assert
```

```
esIgualA5(10)
```



False

Ahora hagamos una función que retorne True si un valor está entre 5 y 6:

```
# estaEntre5y8: int -> bool
# indica si entero n está entre los valores 5 y 8 (sin incluirlos)
# ej: estaEntre5y8(6) devuelve True, también para 7, y el resto False
def estaEntre5y8(n):
    return n>5 and n<8
# test
assert estaEntre5y8(6)
assert estaEntre5y8(7)
assert not estaEntre5y8(5)
```

▼ Condiciones

si pregunta es verdadero **entonces** se ejecuta respuesta


en python:

```
if (pregunta):
    respuesta
```


```
x=5
y=5

if x==y:
    print('son iguales')
```


```
print ( 'son iguales: ' )
```

 son iguales!

```
x = 5
y = 1
if (x > y):
    print (x, 'es mayor que', y)
```

 5 es mayor que 1

```
x = int(input('ingrese entero x '))
y = int(input('ingrese entero y '))
if (x > y):
    print (x, 'es mayor que', y)
```

 ingrese entero x 8
ingrese entero y 5
8 es mayor que 5

▼ ¿Cómo hago un programa que responda lo correcto en cualquier caso?

- **si** pregunta **entonces** respuesta
- **sino** pregunta **entonces** respuesta
- ...
- **si no ocurre nada de lo anterior entonces** respuesta

en python:

```
if (pregunta):
    respuesta
elif(pregunta):
    respuesta
# (...) elif puede repetirse todas las veces que necesitemos
else:
    respuesta
```

Las condiciones siempre se evalúan en ORDEN

Ejemplo:

```
x = int(input('ingrese un entero x '))
y = int(input('ingrese un entero y '))
if (x > y):
```

```

    print (x, 'es mayor que', y)
elif (x < y):
    print (x, 'es menor que', y)
else:
    print (x, 'es igual que', y)

```

podemos omitir el ultimo elif, y reemplazarlo por un else, o incluso todos los elif



```

ingrese un entero x 5
ingrese un entero y 3
5 es mayor que 3

```

▼ Otro ejemplo:

Ejemplo: Módulo jaliscoCachipun.py (debemos crearlo en un archivo aparte con el nombre del jaliscoCachipun.py)

```

from jaliscoCachipun import ganarCachipun

# al importar usando from, no necesitamos poner jaliscoCachipun.ganarCachipun(parametro)
# solo llamamos directamente a ganarCachipun(parametro)

print ('Juego del Jalisco Cachipun')
jugada = input ('Ingrese su jugada (piedra, papel o tijera) ')
print ('Yo te gano con ' + ganarCachipun(jugada))

```



```

Juego del Jalisco Cachipun
Ingrese su jugada (piedra, papel o tijera) papel
Yo te gano con tijera

```

El módulo jaliscoCachipun.py consiste de:

```

# jaliscoCachipun: str -> str
# entrega la jugada ganadora del cachipun dada una entrada valida
# ejemplo: jaliscoCachipun('tijera') debe retornar 'piedra'
def ganarCachipun(jugada):
    if jugada == 'tijera':
        return 'piedra'
    elif jugada == 'papel':
        return 'tijera'
    else:
        return 'papel'

# test
assert ganarCachipun('tijera')== 'piedra'

```

Ejemplo: Ver si un entero es 'par'

```
#par: int -> bool
#True si un entero es par (False si no)
#ej: par(4) debe ser True, par(15) debe ser False
def par(x):
    return x%2==0

assert par(4) #par(4)==True
assert not par(15) #par(15)==False
```

```
par(5)
```

 False

Propuesto:

- Hacer un programa que dado un año N indique si este es bisiesto o no.

▼ Receta de Diseño de funciones condicionales

Para diseñar una función condicional debemos:

1. Identificar cada una de las situaciones posibles (por ejemplo, para funciones numéricas podemos dibujar una recta e identificar intervalos)
2. Dar ejemplos de uso de la función, un ejemplo por cada situación. Incluir casos de borde.
3. Cuerpo de la función: diseñar condiciones:

```
if(...):
    ...
elif(...):
    ...
else():
```

4. Simplificar condiciones

▼ Programa saludo.py

Saludo es un programa que recibe la hora y saluda según corresponda a la hora del día (Buenos días, Buenas tades y Buenas noches):

```
# saludo: int -> str
# Determinar el saludo adecuado a la hora del día 1 <= hora <= 24 # ejemplos:
```

```
# Determinar el saludo adecuado a la hora del día 1 <= hora <= 24 # ejemplos.
# saludo(11) debe devolver Buenos días!
# saludo(15) debe devolver Buenas tardes!
def saludo(hora):
    ## condiciones
    if (hora <12):
        return 'Buenos días!'
    elif (hora <21):
        return 'Buenas tardes!'
    else:
        return 'Buenas noches!'
# test:
# probar condiciones de BORDE (o límites): 1hrs, 12hrs, 21hrs, 24hrs
assert saludo(1) == 'Buenos días!'
assert saludo(11) == 'Buenos días!'
assert saludo(12) == 'Buenas tardes!'
assert saludo(15) == 'Buenas tardes!'
assert saludo(21) == 'Buenas noches!'
assert saludo(23) == 'Buenas noches!'
assert saludo(24) == 'Buenas noches!'
```