python™

CUSOL (Comunidad Universitaria de Software Libre)

Operador Modulo

El operador módulo trabaja sobre enteros y devuelve el resto cuando el primer operando se divide por el segundo.

En Python, el operador de módulo es un signo de porcentaje %. La sintaxis es la misma que para otros operadores, ejemplo:

```
>>> cociente = 7 / 2
>>> print cociente
3
>>> residuo = 7 % 2
>>> print residuo
1
```

Operador Boolean

Una expresión Boolean es aquella que puede ser **True** (verdadera) o **False** (falsa). Para ella se usa el operador **==** para comparar dos operandos, por ejemplo:

```
>>> 5==5
True
>>> 5==6
False
```

Los valores True o False no son cadenas, son un tipo espacial llamada ‘bool’.

```
>>> type(True)
<type ‘bool’>
```

Operadores Relacionales

Hay que tener en cuenta que el operador **==** solo es uno de los operadores relacionales, también existen:

```
x  !=   y                  # x diferente de      y
x  >    y                  # x mayor que         y
x  <    y                  # x menor que         y
x  >=  y                # x mayor igual que   y
x  <=  y                # x menor igual que   y
```

Operadores Lógicos

Existen tres operadores lógicos **and, or, not.**

Por ejemplo:

`x > 0 and x < 10` el resultado de esto será **True** si x es un número entre 0 y 10.

`n % 2  or  n % 3`, **True** para aquellos valores de n que sean múltiplos de 2 o 3.

`not ( x > y )` Será **True** para los casos donde y sea mayor o igual a x

Condicionales

A la hora de programar habrá momentos donde tengamos la necesidad de cambiar el comportamiento del programa al cumplir ciertas condiciones. Los condicionales **(if)** nos da la capacidad de hacerlo, por ejemplo:

```
if x>0:
    print “x es positivo”
```

Si x cumple la condición, nuestro programa seguirá ejecutando lo que está dentro de esta condición. Hay que tener en cuenta a la hora de escribir el código la **indentación.**

Condicionales

Es normal que nos preguntemos qué pasa si no cumple la condición?

En este caso podemos indicar que queremos hacer en caso de que no se cumpla la condición con un **else,** así:

```
if x > 0:
    print “x es positivo”


else:
    print “x no es positivo”
```

Condicionales

También podemos encontrar la necesidad de probar **varias** condiciones, pero usar muchos **if** no es eficiente, por ello es bueno conocer el **elif,** el cual nos permitirá verificar más condiciones en caso de que no se haya cumplido alguna anterior

```
if x > 0:
    print “x es positivo”
elif x < 0
    print “x es negativo”
else
    print “x es igual a 0”
```

Se excluyen entre sí.

Recursión

No es algo malo que un función llame a otra, tampoco que se llame a sí mismo, puede no ser muy obvio, pero es algo mágico que un programa puede hacer, ejemplo:

```
def cuentaAtras(n):
    if n <= 0:
        print ‘Despegue!’
    else:
        print n
        cuentaAtras(n-1)
```

Podemos observar que cuando n sea igual o menor que cero el programa imprimirá ‘Despegue!’

Ejercicios

Teorema de fermat página 49.

Exercise 5.3. *Fermat's Last Theorem says that there are no integers a, b, and c such that*

$$a^n + b^n = c^n$$

for any values of n greater than 2.

1. *Write a function named* `check_fermat` *that takes four parameters—*`a`, `b`, `c` *and* `n`*—and that checks to see if Fermat's theorem holds. If n is greater than 2 and it turns out to be true that*

   $$a^n + b^n = c^n$$

   the program should print, "Holy smokes, Fermat was wrong!" Otherwise the program should print, "No, that doesn't work."

2. *Write a function that prompts the user to input values for* `a`, `b`, `c` *and* `n`*, converts them to integers, and uses* `check_fermat` *to check whether they violate Fermat's theorem.*

Exercise 5.4. *If you are given three sticks, you may or may not be able to arrange them in a triangle. For example, if one of the sticks is 12 inches long and the other two are one inch long, it is clear that you will not be able to get the short sticks to meet in the middle. For any three lengths, there is a simple test to see if it is possible to form a triangle:*

> *If any of the three lengths is greater than the sum of the other two, then you cannot form a triangle. Otherwise, you can. (If the sum of two lengths equals the third, they form what is called a "degenerate" triangle.)*

1. *Write a function named* `is_triangle` *that takes three integers as arguments, and that prints either "Yes" or "No," depending on whether you can or cannot form a triangle from sticks with the given lengths.*

2. *Write a function that prompts the user to input three stick lengths, converts them to integers, and uses* `is_triangle` *to check whether sticks with the given lengths can form a triangle.*

Ejercicio Recursividad

Escriba una función que calcule el factorial de un número de **manera recursiva, ojo! No usando `for` ni `while`.**

El **factorial** de un entero positivo *n*, el **factorial de *n*** o ***n* factorial** se define en principio como el producto de todos los números enteros positivos desde 1 (es decir, los números naturales) hasta *n*. Por ejemplo el factorial de 5 se calcula:

5! = 1x2x3x4x5 = 120

Se puede definir de manera recursiva:

$$n! = \begin{cases} 1 & \text{si, } n = 0 \\ (n-1)! \times n & \text{si, } n > 0 \end{cases}$$

Solución página 56. (Think Python 2.7)

Ciclos de iteración

while: Evalúa una condición y si se cumple se ejecuta lo que está indentado.

Si no se cumple, se rompe el ciclo.

```
def countdown(n):
    while n > 0:
        print n
        n = n-1
    print 'Blastoff!'
```

En el anterior código se actualiza una variable n en cada **iteración.**

Break!

A veces no sabemos cuando romper los ciclos y queremos controlar que no itere para siempre, usamos la palabra clave **break.**

```
while True:
    line = raw_input('> ')
    if line == 'done':
        break
    print line

print 'Done!'
```

Esta es una manera de hacer el conocido “do while” en python.

Ejercicio con while (Tarea)

Página 69 Think Python 2.7

Exercise 7.5. *The mathematician Srinivasa Ramanujan found an infinite series that can be used to generate a numerical approximation of* π*:*

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Write a function called `estimate_pi` *that uses this formula to compute and return an estimate of* π*. It should use a* `while` *loop to compute terms of the summation until the last term is smaller than* `1e-15` *(which is Python notation for* 10^{-15}*). You can check the result by comparing it to* `math.pi`.