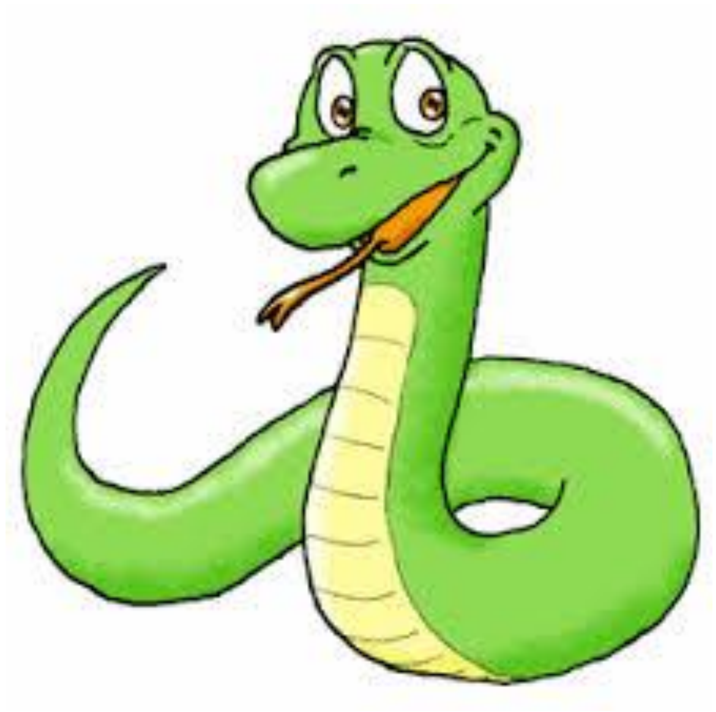
Application of GIS with Python

Chapter 3 : Control statements and repetition



https://www.python.org/

http://www.tutorialspoint.com/python/

Table of Contents

Control Statements & Iteration

- Controls the flow of statements during execution
- **Control statements**:
    - conditional statements(if, if…else etc), break, continue.
- Programs are used for repetitive tasks.
- Repetition of a task is called **iteration**. In programs, iterations are performed in statements that are called **loops**.

if Statements

- The simplest form is the *if statement* of **conditional statement**.

```
if x > 0:
        print 'x is positive'
```

$\Big\}$ compound statement

- The <u>boolean expression</u> after the if statement is called the condition. If it is true, then the indented statement gets executed. If not, nothing happens.
- Structure: Header followed by an indented block (compound statement)

Pass statement?

if…else

- A second form of the if statement is alternative execution, in which there are two possibilities and the condition determines which one gets executed.

```
if x%2 == 0:
        print 'x is even'
else:
        print 'x is odd’
```

- one of the alternatives will be executed; the alternatives are called branches

Nested if

- One conditional can also be nested within another

```
if x == y:
    print 'x and y are equal'
else:
    if x < y:
        print 'x is less than y'
    else:
        print 'x is greater than y‘
```

outer

inner

- One branch contains other conditional branches

Chained conditionals (with elif)

- ➢For more than one alternatives need more than two branches for the condition; chained conditional works here

```
 if x == 3:
          print “X equals 3.”
elif x == 2:
          print “X equals 2.”
else:
          print “X equals something else.”
```

3 branches

Loops

- Loops perform iteration
- Python knows two types of loops:
  - the while loop, and
  - the for loop

while loop

1. A while statement starts with the keyword while. This keyword is always followed by:
2. a condition
3. a colon :
4. a body of statements
5. these statements are indented!

While

- The statements of the while loop are executed repeatedly as long as the condition is True.

```
while condition :
        statement
        statement
```

- The simplest while loop looks like this:

```
>>> while True :
                pass
```

- This loop runs forever, because the condition is always True!
- It can only be stopped by hitting CTRL+C:

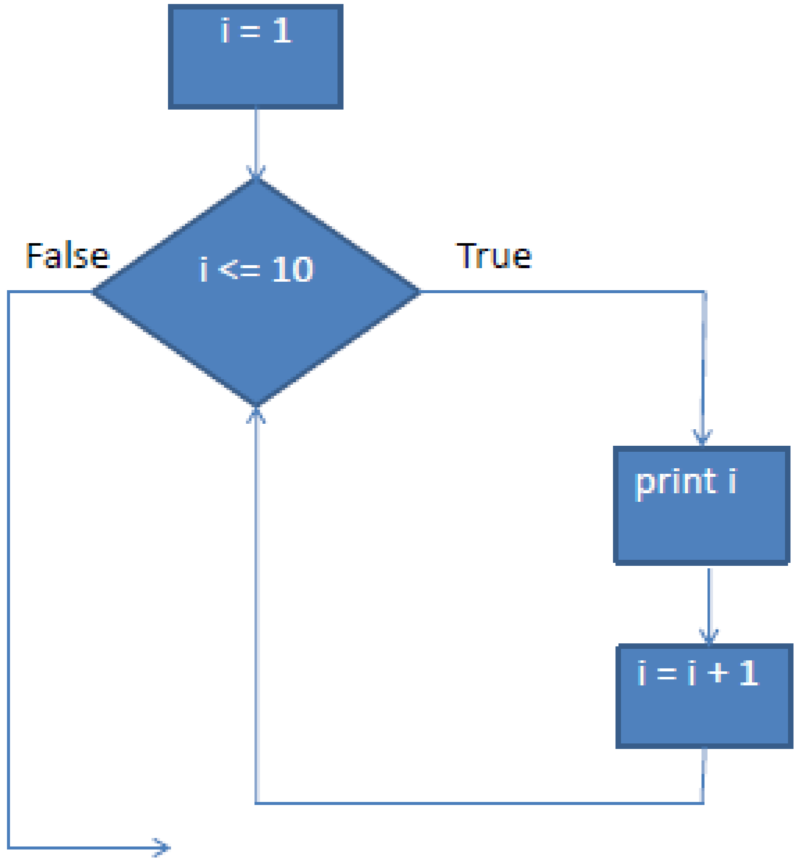
Counting

Here's a while loop that counts to 10:

```
>>> i = 1            #initialization
>>> while i <= 10 :  #condition
            print i
#statement
            i = i + 1    #update
```

```
1
2
3
4
5
6
7
8
9
10
```

- ➢ The variable I controls the loop and is called the loop variable



countdown function

```
6
Fire!
5
Fire!
4
Fire!
3
Fire!
2
Fire!
1
Fire!
```

```
>>> def countdown(n):
        i = n
        while i > 0:
            print i
            i = i - 1
        print ‘Fire!’
>>> countdown(6)
```

break

➢It is possible to halt execution of a loop by using a break statement.

```
>>> i = 1
>>> while True:
        print i,
        if i >= 10:
            break
        print '*',
        i = i + 1
```

- In most cases, a break can be avoided!

```
>>>i = 1
>>>while i < 10:
                    print i, '*',
                    i = i + 1
        print i
```

Square root

We present a recipe (algorithm) for the approximation of the square root of a:

1. start with some value x
2. calculate a new approximation y by applying the formula: $y=(x+a/x)/2$
3. if x is equal to y, then x is the sq. root
4. assign y to x
5. goto step 2

Square root

```
>>> def square_root(a):
    x = 0
    y = a
    while True:
        x = y
        y = (x + a / x) / 2
        if x == y:
            break
        x = y
    return x
```

```
>>> def square_root(a):
        x = 0
        y = a
        while x!=y:
            x = y
            y = (x + a / x) / 2
        return x
```

Without break

for loop

- Python's for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the **sequence**.
- In Python, the for statement looks like this:
  - for variable in sequence :
    - statement
    - statement

- Python has a number of built-in types for collection of things, called sequences:
  - String: stores characters
  - List: stores values
  - Tuple: stores constants
  - Array: stores values of one type
- The data of these objects is typically accessed in a linear way, using iteration.

for loop

```
>>> s= "hellow world"
>>> for c in s:
        print c
```

```
h
e
l
l
o
w

w
o
r
l
d
```

```
>>> # Measure some strings:
a = ['cat', 'window', 'defenestrate']
>>> for x in a:
                    print x, len(x)
```

```
cat 3
window 6
defenestrate 12
```

Nested loop

```
>>> for y in 'ABCDE':                     } Outer loop
        for x in '12345':     } Inner loop
            print y+x,
        print
```

```
A1 A2 A3 A4 A5
B1 B2 B3 B4 B5
C1 C2 C3 C4 C5
D1 D2 D3 D4 D5
E1 E2 E3 E4 E5
```

Assignment 3:

1. What do you understand by control statement and iterations? Describe different forms of control statement and iteration in detail and also express your view on when and how different forms are used with suitable python example.