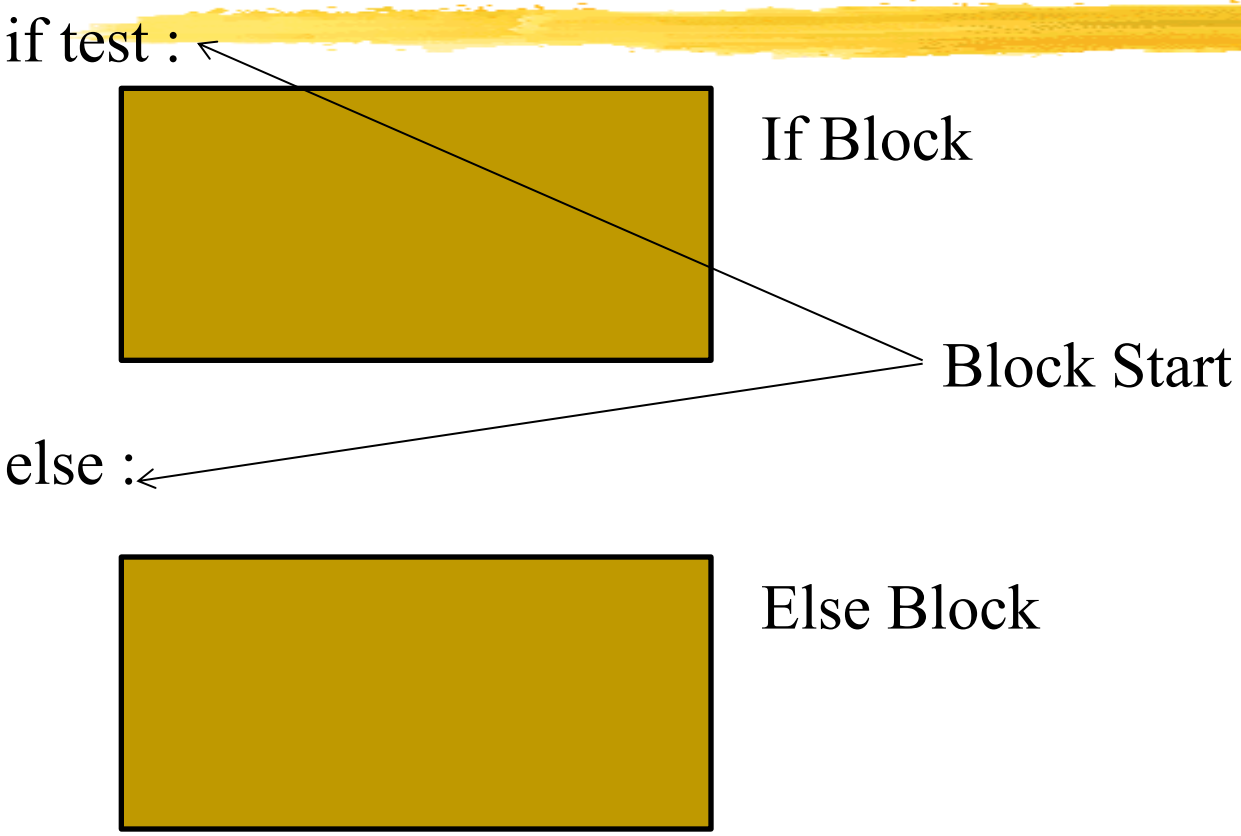
University College Cork
Computer Science
Department

Programming Training

Main Points:

- Problems with repetitions.
- Discuss some important algorithms.

C Selection.



Test if three float form a triangle.

Given three real numbers a, b and c then write a Python program to test whether they form a triangle.

Inputs: a,b,c – float

Output: the answer either yes or not.

Some points:

1. negative numbers do not form triangle
2. a, b, c are triangle sides then

$$a+b>c \text{ and } a+c>b \text{ and } b+c>a$$

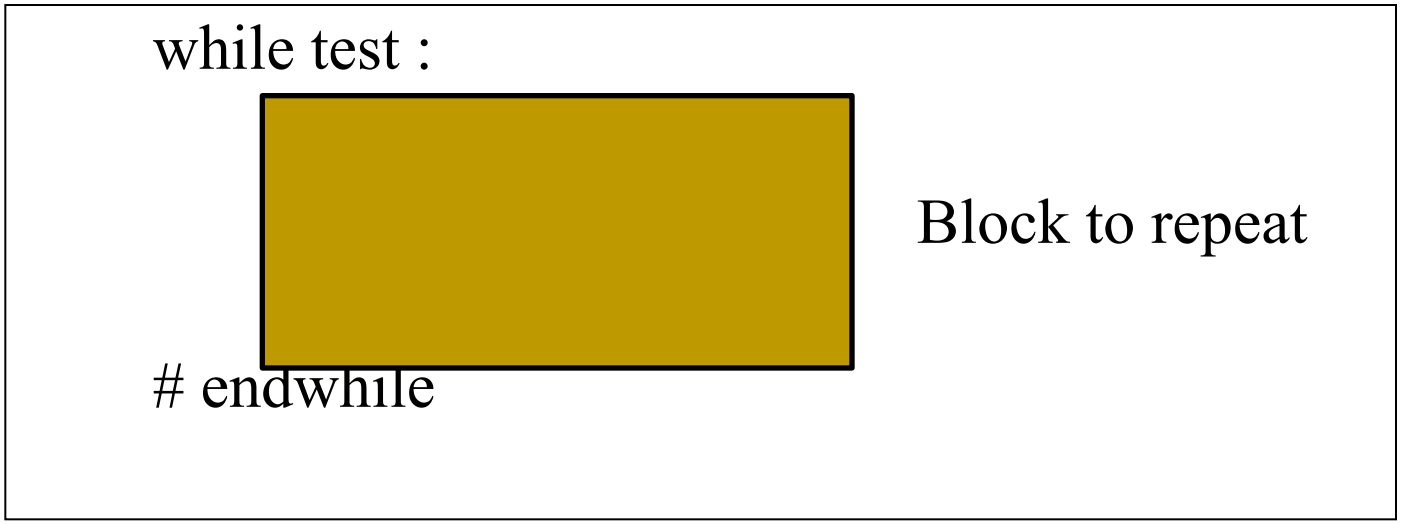
Form a right-angled triangle?

Python 3.4.1: testRightAngleTriangle.py – /Users/sabin/Desktop/python/testRightAngleTriangle.py

```
# import modules
import math

# define function
def testTriangle():

    '''
        This function tests if three float numbers can form a trian
        They form a triangle when they all are positive and satisfy
    '''
    a, b, c = map(int, input('Type a b c:').split())

    if a <= 0 or b <= 0 or c <= 0 :
        print(a, b, c , 'do not form a right-angled triangle')
        return
    # endif

    if a >= b+c or b >= a+c or c >= a+b :
        print(a, b, c , 'do not form a right-angled triangle')
        return
    # endif

    if a**2 == b**2+c**2 or b**2 == a**2+c**2 or c**2 == a**2+b**2
        print(a, b, c , 'form a right-angled triangle')
        return
    else :
        print(a, b, c , 'do not form a right-angled triangle')
        return
    # endif


# end testTriangle
```

C Repetitions.

C repetitions execute repeatedly a statement while a condition holds or for a number of times.



While loop repeats the block while test holds. If test is initially false the the block is not exeecuted.

C Repetitions.

C repetitions execute repeatedly a statement while a condition holds or for a number of times.

for elem in iterator :

Block to repeat

endfor

For loop repeats statements for all elems in iterator.

iterator is a set of elements that can be traversed.

Iterators

- Iterators can be given by:
  - range(n) ➔ all values between 0, 1,…, n-1
  - range(a,b)➔ all values between a, a+1,…,b-1
  - range(a, b, step)

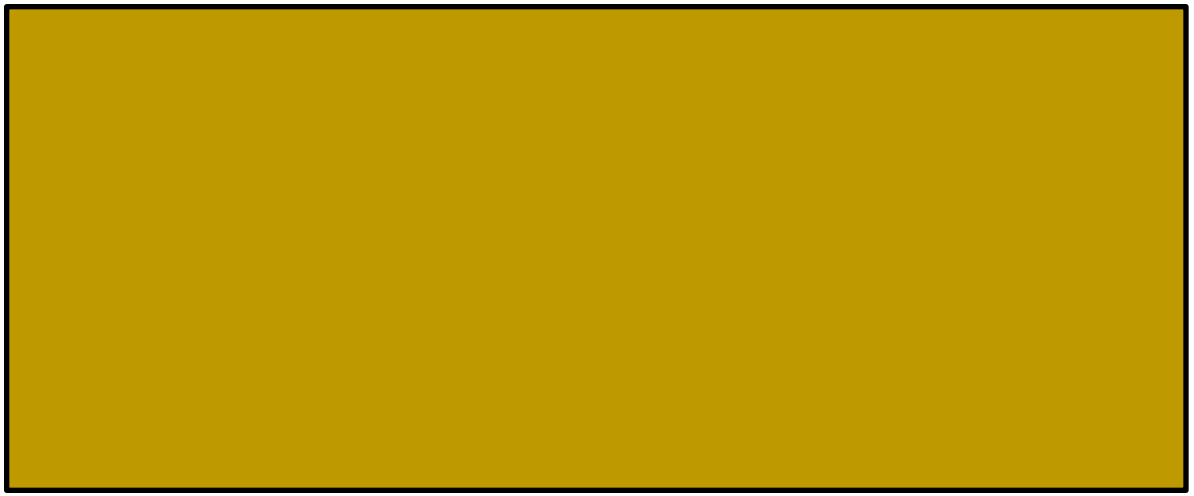
    ➔ all values between a, a+step, a+2step, …
- Lists, strings can also be iterators

Examples of for repetition

```
for letter in str :
    print(letter)

for elem in list :
    sum = sum + elem

for i in range(10) :
    print(i, i**2, i**3)
```

Examples of for repetition

Suppose that there is a list a = [a[i], i=0,…,n-1].



Find pow so that d^pow | n.

Given an int number n and d one of its divisors then find what the of is in n.

Some Hints:

1) d is a divisor of n when n % d == 0
2) Examples:
    1) n = 100 and d = 2 ➔ pow = 2
    2) n = 81 and d = 2 ➔ pow = 0
    3) n = 81 and d = 3 ➔ pow = 4

Find pow so that d^pow | n.

Counting problems use a counting counter *cont*:

- initial value is 0.
- count increases when the event occurs.

Any time when a divisor is found we remove it from the number.

Example: n=72 and k=2.

pow:	0	1	2	3
n :	72	36	18	9
d :	2	2	2	

The repetition takes place while n is divisible by d: n % d ==0

Python 3.4.1: power.py - /Users/sabin/Desktop/python/power.py

```
# import modules
import math

# define functions
def power():

    # input n d
    n = int(input('n='))
    d = int(input('d='))

    # count how many times d can be extracted n
    count = 0

    while n % d == 0 :

        count=count+1
        n = n/d

    #endwhile

    print('power=', count)

# end power
```

Prime number decomposition

If a number n is integer find the prime number decomposition.

We need to find all the prime divisors together with their powers.

Examples:

36 = 2**2 * 3**2

54 = 2**1 * 3**3

1024 = 2**10

Hint:

for a prime divisor p use the previous schme to find its power

possible divisors 2,3,4,5,…, n

Prime number decomposition

Repeat for d=2,3,4,5,…?

- if d is a divisor of n then
  - find the power of d in n.
  - write d and pow

Example: n=72

n	:	72	9	1
d	:	2	3	4
pow	:	3	2	

Repetition ends when n<d.

Python 3.4.1: decompositionpower.py – /Users/sabin/Desktop/python/decompositionpower.py

```
# import modules
import math

def decomposition():

    # input n d
    n = int(input('n='))

    #traverse the possible divisors

    for d in range(2,n+1) :

        if n % d == 0 :

            # count how many times d can be extracted n
            count = 0

            while n % d == 0 :

                count=count+1
                n = n/d

            #endwhile

            print(d, '**' ,count)

        # endif

    # endfor

#end main
```

Find the digits of an integer.

If n is a long number then

last= n % 10 is the last digit.

n = n/10 is the new number from which we remove last.

If we repeat removing the last digit then we can find:

- all the digits of the number.
- number of digits.

Example: n=45672941

last:	1	4	9	2	…
n:	4567294	456729	45672	4567	…

We repeat while the number is not fully done.

Python 3.4.1: digits.py - /Users/sabin/Desktop/python/digits.py

```
import math

def digits():

    # input n
    n = int(input('n='))
    count = 0

    # find the digits
    while n != 0 :

        digit = n % 10
        print(digit)
        n = n//di10
        count = count + 1

    # end while

    print('number of digits', count)

#end digits
```

To do List

1. Read more about while and for from the e-tutorial.
2. Solve the HW problems.