In [5]:

```
# instructiuni repetitive: for, etc.
# Fibonacci series:
# the sum of two elements defines the next

# concepte importante: atribuire multipla

# the expressions on the right-hand side are all evaluated first
# before any of the assignments take place
# the right-hand side expressions are evaluated from the left to the right

a, b = 0, 1

# any non-zero integer value is true, zero is false
# the condition may also be a string or list value, in fact any sequence
# anything with a non-zero length is true, empty sequences are false
# comparison operators: <, >, ==, <=, >=, !=

while b < 10:
    #print(b)
    print(b, end=', ')
    a, b = b, a+b

# the body of the loop is indented - for grouping statements

# the keyword argument "end" can be used
# to end the output with a different string
```

```
1, 1, 2, 3, 5, 8,
```

In [1]:

```
x = int(input("Please enter an integer: "))
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:                    # short for "else if"
    print('Zero')
elif x == 1:
    print('Single')
else:                           # optional
    print('More')
```

```
Please enter an integer: 0
Zero
```

In [6]:

```
# Measure some strings:
words = ['cat', 'window', 'door']
for w in words:
    print(w, len(w))
```

```
cat 3
window 6
door 4
```

In [7]:

```
for i in range(5):
    print(i)

# the given end point is never part of the generated sequence
```

```
0
1
2
3
4
```

In [14]:

```
a = ['Mary', 'had', 'a', 'little', 'cat']
for i in range(len(a)):
    print(i, a[i])

# the object returned by range() behaves as if it is a list, but in fact it isn't
# it is an object which returns the successive items of the desired sequence
# when you iterate over it, but it doesn't really make the list,
# thus saving space

# we say such an object is iterable
# the "for" statement is an iterator
```

```
0 Mary
1 had
2 a
3 little
4 cat
```

In [16]:

```
list(range(5))
# the function "list()" is another iterator
# it creates lists from iterables
```

Out[16]:

```
[0, 1, 2, 3, 4]
```

In [18]:

```
# loop which searches for prime numbers
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print(n, 'equals', x, '*', n//x)
            break
    else:   # else clause on the "for" loop, NOT the "if" statement
        # loop fell through without finding a factor
        print(n, 'is a prime number')

# loop statements may have an else clause
# it is executed when the loop terminates through exhaustion
# of the list (with "for") or when the condition
# becomes false (with "while"), but not when
# the loop is terminated by a "break" statement
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

In [19]:

```
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found a number", num)

# the "continue" statement continues
# with the next iteration of the loop
```

```
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

In [32]:

```
# the keyword "def" introduces a function definition
# it must be followed by the function name
# and the parenthesized list of formal parameters
# the statements that form the body of the function are indented

def fib(n):    # write Fibonacci series up to n
    print("Print a Fibonacci series up to", n)
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

# Now call the function we just defined:
fib(2000)
fib(50)

# even functions without a "return" statement do return a value
# this value is called "None" (it's a built-in name)
print(fib(50))
```

```
Print a Fibonacci series up to 2000
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
Print a Fibonacci series up to 50
0 1 1 2 3 5 8 13 21 34
Print a Fibonacci series up to 50
0 1 1 2 3 5 8 13 21 34
None
```

In [34]:

```
def fib2(n):  # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)    # calls a method of the list object "result"
        a, b = b, a+b
    return result

f100 = fib2(100)    # call it
f100                # write the result
```

Out[34]:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

In [ ]:

```
# Coding style
# Use 4-space indentation, and no tabs.
# Wrap lines so that they don’t exceed 79 characters.
# When possible, put comments on a line of their own.
# Use spaces around operators and after commas.
# Name your classes and functions consistently;
# the convention is to use "CamelCase" for classes
# and "lower_case_with_underscores" for functions and methods.
```