Politecnico
di Milano

Coding in Python

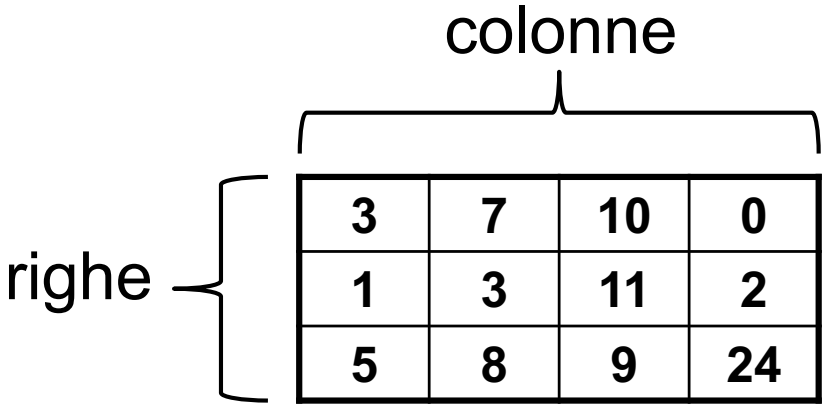
10-14 Giugno 2018

Lezione 3

MATRICI

Excursus Matematico – Le Matrici

- In matematica, una **matrice** è uno schieramento rettangolare di oggetti.
- Servono principalmente a descrivere valori che dipendono da due parametri, e per questo motivo sono ampiamente usate in matematica e in tutte le scienze.



- Ogni elemento è identificato da un indice di riga e uno di colonna, per esempio, a_{12}

Excursus Matematico – Le Matrici

- Se la matrice si chiama A
  - L'elemento posizionato nella riga *i* e nella colonna *j* può essere indicato in vari modi: ad esempio come $A_{i,j}$, o tramite parentesi quadre $A[i,j]$. Si usa talvolta la notazione $A = (a_{i,j})$ per indicare che A è una matrice e che i suoi elementi sono denotati con $a_{i,j}$.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

- Vettori colonna e vettori riga sono tipi particolari di matrici

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & -2 & 0 \\ 4.5 & 0 & 2 \\ 6 & 1 & 5 \end{bmatrix} \quad \begin{bmatrix} 7 \\ 0 \\ \pi \end{bmatrix} \quad \begin{bmatrix} 3 & \frac{7}{2} & -9 \end{bmatrix}$$

Excursus Matematico – Le Matrici

Somma

- Due matrici *A* e *B* possono essere sommate se hanno le stesse dimensioni.
- La loro somma *A* + *B* è definita come la matrice i cui elementi sono ottenuti sommando i corrispettivi elementi di *A* e *B*. Formalmente:

  $(A + B)_{i,j} := A_{i,j} + B_{i,j}$

  Per esempio

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \\ 1+2 & 2+1 & 2+1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \\ 3 & 3 & 3 \end{bmatrix}.$$

Excursus Matematico – Le Matrici

- La **moltiplicazione per uno scalare** è un'operazione che, data una matrice *A* ed un numero *c* (detto *scalare*), costruisce una nuova matrice *cA*, il cui elemento è ottenuto moltiplicando l'elemento corrispondente di *A* per *c*

  $(cA)_{ij} := cA_{i,j}$. Per esempio:

$$2\begin{bmatrix} 1 & 8 & -3 \\ 4 & -2 & 5 \end{bmatrix} = \begin{bmatrix} 2\times 1 & 2\times 8 & 2\times -3 \\ 2\times 4 & 2\times -2 & 2\times 5 \end{bmatrix} = \begin{bmatrix} 2 & 16 & -6 \\ 8 & -4 & 10 \end{bmatrix}.$$

Excursus Matematico – Le Matrici

Prodotto tra matrici AxB

- È definito soltanto se il numero di righe di *B* coincide con il numero *n* di colonne di *A*. Il risultato è una matrice con lo stesso numero di righe di A e lo stesso numero di colonne di B
  - Se A è una matrice mxn e B una matrice nxl, C = A x B sarà una matrice mxl

- L'elemento di posizione (*i*,*j*) in C è dato dalla somma

$$C_{i,j} := A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + \cdots + A_{i,n}B_{n,j}.$$

Excursus Matematico – Le Matrici

Per esempio:

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & -3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 2 & 5 & 1 \\ 0 & -2 & 1 \end{bmatrix} =$$

Moltiplicando una matrice 2×3 per una 3×3 si ottiene una matrice 2×3.

1° riga:

$$C_{11} = [(1 \times 1) + (1 \times 2) + (2 \times 0)] = 3$$
$$C_{12} = [(1 \times 1) + (1 \times 5) + (2 \times -2)] = 2$$
$$C_{13} = [(1 \times 1) + (1 \times 1) + (2 \times 1)] = 4$$

2° riga:

$$C_{21} = [(0 \times 1) + (1 \times 2) + (-3 \times 0)] = 2$$
$$C_{22} = [(0 \times 1) + (1 \times 5) + (-3 \times -2)] = 11$$
$$C_{23} = [(0 \times 1) + (1 \times 1) + (-3 \times 1)] = -2$$

Risultato 2×3:

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{bmatrix} = \begin{bmatrix} 3 & 2 & 4 \\ 2 & 11 & -2 \end{bmatrix}$$

Excursus Matematico – Le Matrici

Per esempio:

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & -3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 2 & 5 & 1 \\ 0 & -2 & 1 \end{bmatrix} =$$

Moltiplicando una matrice 2×3 per una 3×3 si ottiene una matrice 2×3.

1° riga:

$$C_{11} = [(1 \times 1) + (1 \times 2) + (2 \times 0)] = 3$$
$$C_{12} = [(1 \times 1) + (1 \times 5) + (2 \times -2)] = 2$$
$$C_{13} = [(1 \times 1) + (1 \times 1) + (2 \times 1)] = 4$$

2° riga:

$$C_{21} = [(0 \times 1) + (1 \times 2) + (-3 \times 0)] = 2$$
$$C_{22} = [(0 \times 1) + (1 \times 5) + (-3 \times -2)] = 11$$
$$C_{23} = [(0 \times 1) + (1 \times 1) + (-3 \times 1)] = -2$$

Risultato 2×3:

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{bmatrix} = \begin{bmatrix} 3 & 2 & 4 \\ 2 & 11 & -2 \end{bmatrix}$$

Excursus Matematico – Le Matrici

Per esempio:

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & -3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 2 & 5 & 1 \\ 0 & -2 & 1 \end{bmatrix} =$$

Moltiplicando una matrice 2×3 per una 3×3 si ottiene una matrice 2×3.

1° riga:

$$C_{11} = [(1 \times 1) + (1 \times 2) + (2 \times 0)] = 3$$
$$C_{12} = [(1 \times 1) + (1 \times 5) + (2 \times -2)] = 2$$
$$C_{13} = [(1 \times 1) + (1 \times 1) + (2 \times 1)] = 4$$

2° riga:

$$C_{21} = [(0 \times 1) + (1 \times 2) + (-3 \times 0)] = 2$$
$$C_{22} = [(0 \times 1) + (1 \times 5) + (-3 \times -2)] = 11$$
$$C_{23} = [(0 \times 1) + (1 \times 1) + (-3 \times 1)] = -2$$

Risultato 2×3:

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{bmatrix} = \begin{bmatrix} 3 & 2 & 4 \\ 2 & 11 & -2 \end{bmatrix}$$

Excursus Matematico – Le Matrici

Matrici quadrate

- Una matrice si dice quadrata se ha lo stesso numero di righe e colonne.
- Una matrice quadrata ha una diagonale principale, quella formata da tutti gli elementi $a_{i,i}$ con indici uguali. La somma di questi elementi è chiamata *traccia*. L'**operazione di trasposizione** trasforma una matrice quadrata A nella matrice A^t ottenuta scambiando ogni $a_{i,j}$ con $a_{j,i}$, in altre parole ribaltando la matrice intorno alla sua diagonale principale.
- Una matrice tale che $a_{i,j} = a_{j,i}$ è una matrice **simmetrica**. In altre parole, A è simmetrica se $A = A^t$.
- Se tutti gli elementi che non stanno nella diagonale principale sono nulli, la matrice è detta **diagonale**.

Verso l'implementazione delle matrici in Python: liste di liste

- Una lista di liste contiene al suo interno una o più liste, esempio
  - lista = ["ciao", 2.0, 5, [10, 20]]
  - print(lista[0]) → `'ciao'`
  - print(lista[1]) → `2.0`
  - print(lista[3]) → `[10, 20]`
- Per accedere al primo elemento della lista interna
  - listaInterna = lista[3]
  - print(listaInterna[0]) → `10`
- Oppure
  - print(lista[3][0]) → `10`

Uso delle liste di liste per rappresentare le matrici in Python

- Consideriamo la matrice

3	7	10	0
1	3	11	2
5	8	9	24

- Possiamo rappresentarla come
  - M = [3, 7, 10, 0], [1, 3, 11, 2], [5, 8, 9, 24]]
- M[1] → `[1, 3, 11, 2]`
- M[1][1] → `3`

Riempimento di una matrice di 0 e visualizzazione


```
numerorighe=3
numerocolonne=4
m=[] #lista vuota
for i in range(numerorighe):
    n=[]
    for j in range(numerocolonne):
        n.append(0)
    m.append(n)
print("m: ", m)
```

Riempimento di una matrice con valori inseriti dall'utente


```
numerorighe=3
numerocolonne=4
m=[]
for i in range(numerorighe):
    n=[]
    for j in range(numerocolonne):
        x=int(input("Inserire elemento (”
              +str(i)+","+str(j)+")\n"))
        n.append(x)
    m.append(n)
print("m: ", m)
```

Verifica che una matrice quadrata sia simmetrica


```
dimensione=4
m=[[1,2,3,4],[2,3,5,0],[3,5,6,8],[4,0,8,0]]
simmetrica=True
for i in range(dimensione):
    for j in range(dimensione):
        if m[i][j]!=m[j][i]:
            simmetrica=False
if simmetrica :
  print('La matrice e` simmetrica')
else :
 print('La matrice non e` simmetrica')
```

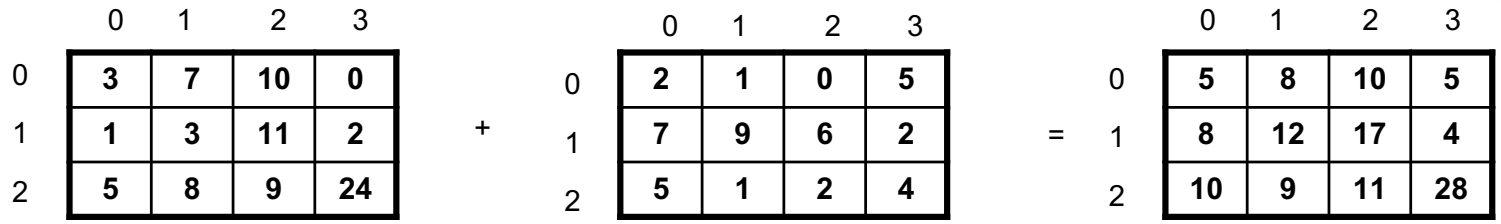
si può migliorare?

Verifica che una matrice quadrata sia simmetrica


```
dimensione=4
m=[[1,2,3,4],[2,3,5,0],[3,5,6,8],[4,0,8,0]]
simmetrica=True
for i in range(dimensione):
    for j in range(i):
        if m[i][j]!=m[j][i]:
            simmetrica=False
if simmetrica :
  print('La matrice e` simmetrica')
else :
 print('La matrice non e` simmetrica')
```

ora?

Somma di matrici



	0	1	2	3
0	3	7	10	0
1	1	3	11	2
2	5	8	9	24

+

	0	1	2	3
0	2	1	0	5
1	7	9	6	2
2	5	1	2	4

=

	0	1	2	3
0	5	8	10	5
1	8	12	17	4
2	10	9	11	28

$$c[i][j] = a[i][j] + b[i][j]$$

```
A = [[3, 7, 10, 0], [1, 3, 11, 2], [5, 8, 9, 24]]
B = [[2, 1, 0, 5], [7, 9, 6, 2], [5, 1, 2, 4]]
C = []
for i in range(len(A)):
    riga = []
    for j in range(len(A[0])):
      riga.append(A[i][j] + B[i][j])
    C.append(riga)
print(C)
```

DICTIONARIES

Dictionaries

- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
  - Associative Arrays - Perl / PHP
  - Properties or Map or HashMap - Java
  - Property Bag - C# / .Net

Dictionaries: characteristics

- Another built-in data type
  - accessed by key (not offset)
  - unordered collection of arbitrary objects
  - variable length, heterogenous, arbitrarily nestable
  - mutable
  - tables of object references (has tables)

Dictionaries: basics


```
>>> D = {'food':'Spam', 'quantity':4, 'color':pink}
>>> D['food']
'Spam'
>>> D['quantity'] +=1
>>> D
{'food':'Spam', 'quantity':5, 'color':'pink'}
>>> len(D)
3
>>> list(D.keys())
['food', 'quantity', 'color']
>>> list(D.values())
['Spam', 5, 'pink']
```

Creating Dictionaries


```
>>> D = {}
>>> D['name'] = 'Bob'
>>> D['job'] = 'developer'
>>> D['age'] = 40
>>> D
{'age':40, 'name':'Bob', 'job':'Developer'}
>>> print(D['name'])
Bob
>>> d1 = dict(name='Bob', job='developer', age=40)
>>> d1
{'age':40, 'name':'Bob', 'job':'Developer'}
```

Changing Dictionaries


```
>>> D
{'eggs':3, 'spam':2, 'ham':1}

>>> D['ham'] = ['grill', 'bake', 'fry']
>>> D
{'eggs':3, 'spam':2, 'ham':['grill', 'bake', 'fry']}

>>> del D['eggs']
>>> D
{'spam':2, 'ham':['grill', 'bake', 'fry']}

>>> D['brunch'] = 'Bacon'
>>> D
{'brunch':'Bacon', 'spam':2, 'ham':['grill', 'bake', 'fry']}
```

Changing Dictionaries


```
>>> D
{‘eggs’:3, ‘spam’:2, ‘ham’:1}
>>> D2 = {‘toast’:4, ‘muffin’:5}
>>> D.update(D2)
>>> D
{‘eggs’:3, ‘muffin’:5, ‘toast’:4, ’spam’:2, ‘ham’:1}
```

Nesting


```
>>> record = {'name':{'first':'Bob', 'last':'Smith'},
                  'jobs':['developer', 'manager'],
                  'age':40}
>>> record['name']
{'last':'Smith', 'first':'Bob'}
>>> record['name']['last']
'Smith'
>>> record['jobs']
['developer', 'manager']
>>> record['jobs'][-1]
'manager'

>>>record['jobs'].append('janitor')
    {'name':{'first':'Bob',
    'last':'Smith'},'jobs':['developer', 'manager',
    'janitor'],'age':40}
```

The get Method for Dictionaries

- The pattern of checking to see if a key is already in a dictionary and assuming a default value if the key is not there is so common that there is a method called get() that does this for us

```
if name in counts:
   x = counts[name]
else :
   x = 0
```

```
x = counts.get(name, 0)
```

Counting Pattern


```
counts = dict()
print('Enter a line of text:')
line = input('')

words = line.split()

print('Words:', words)

print('Counting...')
for word in words:
    counts[word] = counts.get(word,0) + 1
print('Counts', counts)
```

The general pattern to count the words in a line of text is to split the line into words, then loop through the words and use a dictionary to track the count of each word independently.

Retrieving Lists of Keys and Values

- You can get a list of keys, values, or items (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print(list(jjj))
['jan', 'chuck', 'fred']
>>> print(jjj.keys())
['jan', 'chuck', 'fred']
>>> print(jjj.values())
[100, 1, 42]
>>> print(jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

What is a “tuple”? - coming soon...

Definite loops with two iteration variables

- We loop through the key-value pairs in a dictionary using *two* iteration variables
- Each iteration, the first variable is the key and the second variable is the corresponding value for the key

```
jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
for aaa,bbb in jjj.items() :
    print(aaa, bbb)
```

```
jan 100
chuck 1
fred 42
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

Counting words in a text


```
name = input('Enter file:')
handle = open(name)

counts = dict()
for line in handle:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print(bigword, bigcount)
```

TUPLES

Tuples

- Tuples are sequences
  - ordered collections of arbitrary objects
  - accessed by offset
  - immutable (like strings)
  - fixed length, heterogenous and arbitrarily nestle
  - arrays of object references

Tuples: basics

```
>>> T = (1,4,3,4)
>>> len(T)
4
>>> T + (5,6)
(1,2,4,4,5,6)
>>> T[0]
1
```

```
>>> T.index(4)
1
>>>T.count(4)
2
>>>T[0] = 2
error
```

index returns the smallest position of an element

count returns the number of occurrences of an element

immutable

Syntax Issues

- Parentheses are used to enclose both expressions and tuples in Python
- Commas are used to separate elements in a tuple, so I can use them to distinguish between expressions and tuples (i.e., commas have no place in expressions)
- But how do I distinguish between the two when I have only one element in the tuple?!?!
  - Use the comma!

```
>>> x = (40)
>>> x
40
```

```
>>> x = (40,)
>>> x
(40,)
```

How can I sort a tuple?


```
>>> T = (‘cc’, ‘aa’, ‘dd’, ‘bb’)
>>> tmp = list(T)
>>> tmp = sorted(tmp)
>>> tmp
[‘aa’, ‘bb’, ‘cc’, ‘dd’]

>>> T = tuple(tmp)
>>> T
(‘aa’, ‘bb’, ‘cc’, ‘dd’)
```

Why do we have both lists and tuples?

- It's about mutability vs. immutability
- The immutability of tuples provides integrity
  - a tuple will not be changed through another reference
  - similar to constants…

SETS

Sets

- Unordered collections of unique and immutable objects

```
>>> X = set('spam')
>>> Y = {'h', 'a', 'm'}
>>> Z = X, Y
>>> Z
({'m', 'a', 'p', 's'},
{'m', 'a', 'h'})
>>> X & Y
{'m','a'}
>>> X | Y
{'m', 'h', 'a', 'p', 's'}
>>> X-Y
{'p', 's'}
>>> X > Y
False
>>> X > X - Y
True
```

Filtering out duplicates

```
>>> list(set([1,2,1,3,1]))
[1,2,3]
```

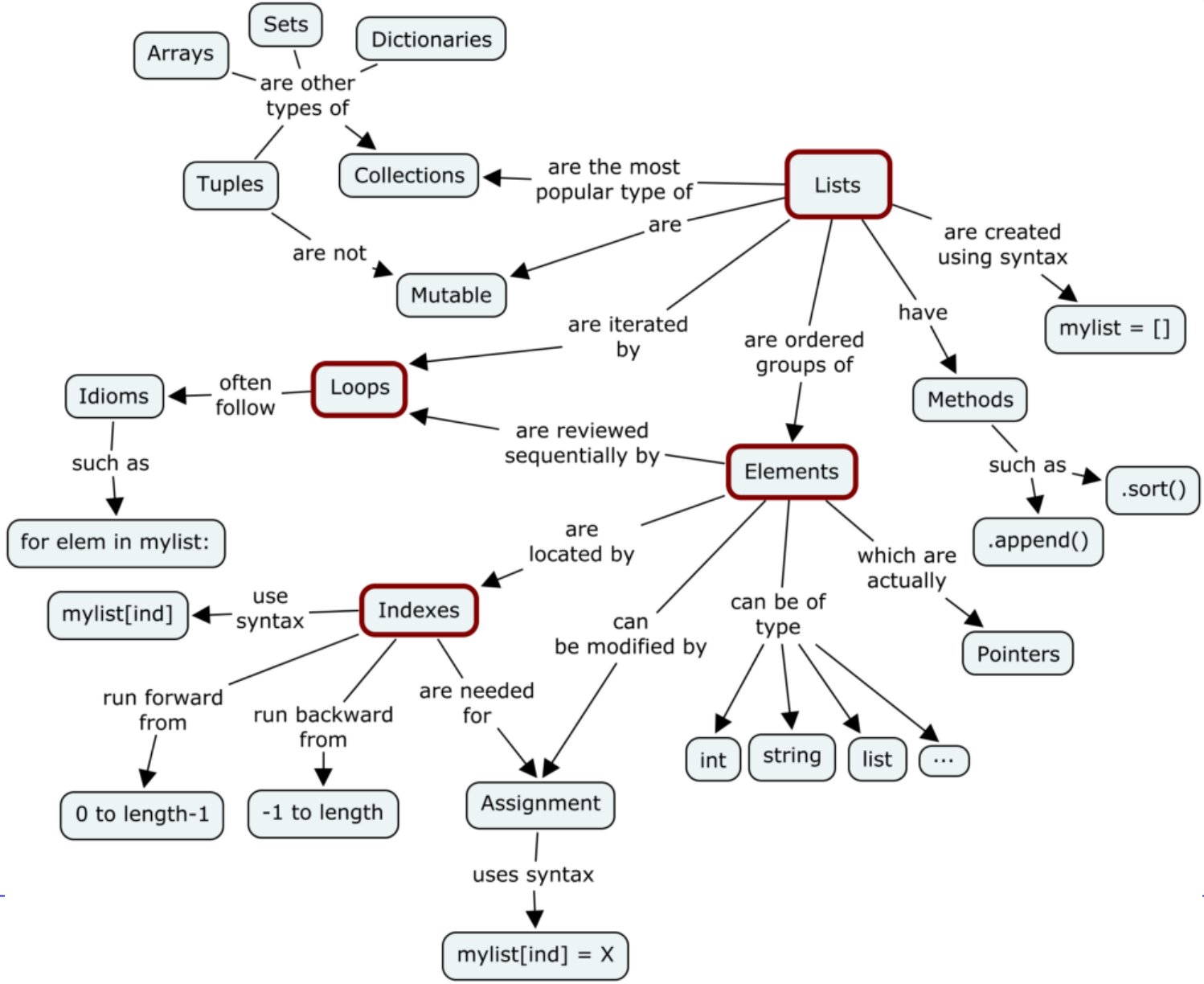
Finding Differences

```
>>>set('spam') - set('ham')
{'p', 's'}
```

Order-neutral equality

```
>>> set('spam') == set('asmp')
True
```

Summary


Sets
Arrays
Dictionaries
are other types of
Tuples
Collections
are the most popular type of
Lists
are
are not
Mutable
are created using syntax
mylist = []
are iterated by
are ordered groups of
have
Idioms
often follow
Loops
Methods
are reviewed sequentially by
Elements
such as
such as
.sort()
for elem in mylist:
.append()
are located by
which are actually
mylist[ind]
use syntax
Indexes
Pointers
can be modified by
can be of type
run forward from
run backward from
are needed for
int
string
list
...
0 to length-1
-1 to length
Assignment
uses syntax
mylist[ind] = X

FUNCTIONS

Prologue

- Remember this piece of code?

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

Functions!

Why do we use functions?

- Reusability
  - As programmers we have to work less
- Abstraction
  - Express complex operations in a compact form
  - The program becomes more readable, compare the two pieces of code below…

```
str = 'my home address'
count = 0
for x in str:
   count = count+1
print(count)
```

```
str = 'my home address'
print(len(str)
```

- Functions also allow us to extend a language with new operations

How do we create a new function in Python?

- Function definition

Function name

Parameters

```
def times(x,y) :
    return x * y
```

Return statement: is associated to an expression to compute the value returned to the caller

- Function Usage

Variable that will store the return value

Arguments

```
k = times(2,4)
print(k)          8

j = times('No', 3)
print(j)          'NoNoNo'
```

Preliminary hints on program structure

- Functions can be defined anywhere in a Python program
- Suggested approach
  - Clearly distinguish between functions and main script part
  - If a function is meant to be used also in other scripts, write it in a separate file

```
#first.py
def times(x,y) :
    return x * y
```

```
#second.py
import first
k = first.times(2,4)
print(k)

j = first.times('No', 3)
print(j)
```

Polymorphism in Python

- A single function can be generally applied to a wide variety of objects, as long as they implement the correct interface
- We say that every operation is polymorphic in Python

```
def intersect(seq1, seq2):
    res =[]
    for x in seq1:
        if x in seq2:
          res.append(x)

    return res
```

```
>>> s1 = "SPAM"
>>> s2 = "SCAM"
>>> intersect(s1, s2)
['S', 'A', 'M']
>>>  x = intersect([1, 2, 3], (1, 4))
>>>  x
[1]
```

Scopes

- Where we define a variable will determine its scope of visibility
  - Where can I access a variable from?
  - By default, all names assigned inside a function are associated with that function's namespace, and no other
    - They can only be seen from within the function
    - They do not clash with names defined outside the function
  - Variables can be defined in 3 different places
    - Inside a def -> local to a function
    - Inside an enclosing def -> nonlocal to nested functions
    - outside of all def -> global to the entire file

Summarising Scopes

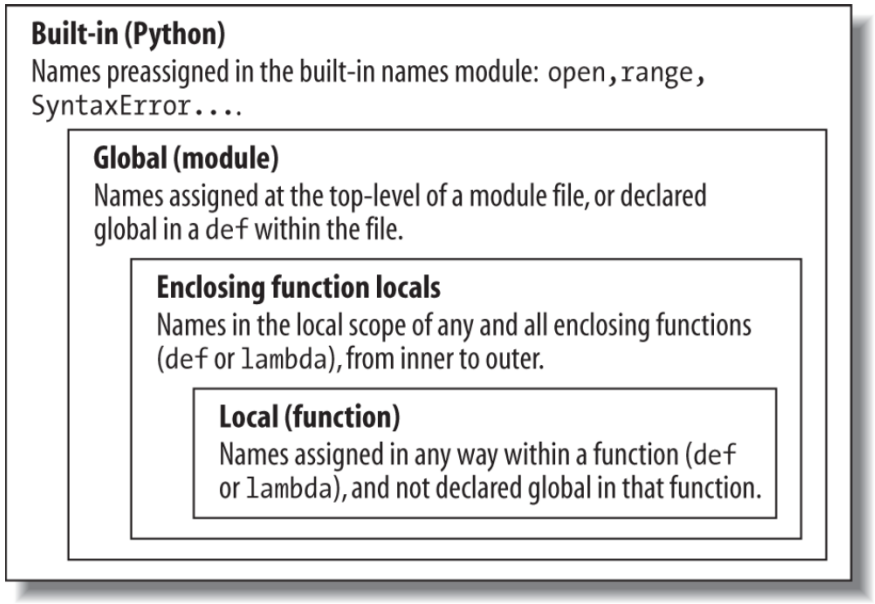
The enclosing module defines a global scope

The global scope spans a single file only

Assigned names are local unless explicitly declared nonlocal or global

All other names are enclosing function locals, globals, or built-ins

Each call to a function creates a new local scope



Keep in mind the LEGB rule !!!
local, enclosed, global, built-in

Using the global keyword


```
X = 88

def func():
    global Y
    Y = 99
    print(X)
    J = 24

func()

print(X)
print(Y)
#J here is not visible
```

X is global, i.e., visible in the whole file

Y is global as well as I have used the keyword global

J is local to the function

Try to minimise the use of global variables even if they make your life easier…

Minimize cross-file changes


```
#first.py
X = 99
```

```
#second.py
import first
print(first.X)
first.X = 88
```

When you import a module you get access to all its global variables!

The coupling becomes too strong!!!

Whoever is responsible for first.py might not know what you are doing with it!!!

Doing it the right way…

```
# first.py
X = 99
def setX(new):
    global X
    X = new
```

```
# second.py
import first
first.setX(88)
```

Scopes and Nested Functions

LEGB - **enclosed**

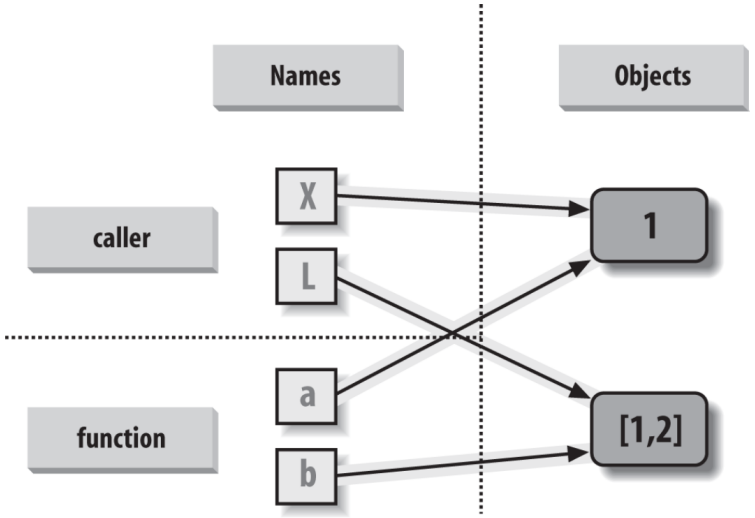
```
X = 99

def f1():
    X = 88
    def f2():
        print(X)
    f2()
f1()
```

Arguments - example

```
def changer(a,b):
  a = 2
  b[0] = 'spam'

>>> X = 1
>>> L = [1, 2]
>>> changer(X, L)
>>> X, L
(1, ['spam', 2])
```



Immutable vs mutable objects

Arguments

- Arguments are passed by automatically assigning objects to local variable names
- Assigning to argument names inside a function does not affect the caller
- Changing a mutable object argument in a function may impact the caller
- **Immutable arguments** are effectively passed "**by value**"
- **Mutable arguments (lists/dictionaries) are effectively passed "by pointer"**

Avoiding Changes

Changes to the objects are often necessary!

Passing objects using pointers can also be much more efficient than passing around the actual data.

However, there may be times when I want to avoid change. In this case I need to pass a **copy** of the data

```
>>> L = [1,2]
>>> changer(X, L[:])
```

The **:** operator creates a copy!!!

The copying can also be achieved directly within the function

Output Params and Multiple Values

Multiple output parameters can be packed into a tuple, or a different collection type

```
>>> def multiple():
                x = 2
                y = [3,4]
                return x, y

>>> X,L = multiple()
>>> X,L
2, [3,4]
```

Exercise

- Modify the pangram script to write it as a function that, given a text, returns True if the text is a pangram.

Solution


```
def pangram(text):
        for letter in [chr(ord('a')+i) for i in range(26)]:
                if letter not in text:
                        return False
        else:
                return True

text1 = 'The quick brown fox jumps over the lazy dog'
text2 = 'not a pangram'
if pangram(text1)==True:
    print('text ' + text1 + ' is a pangram' )
else :
    print('text ' + text1 + ' is not a pangram' )

if pangram(text2)==True:
    print('text ' + text2 + ' is a pangram' )
else :
    print('text ' + text2 + ' is not a pangram' )
```

Exercise

- Write a program that reads a text from a file and creates a list with all words in the text that are palindromes
  - Focus on creating a nice structure for your program
  - Hint: you can use word[::-1] to revert a word

A possible solution


```
def getPalindromes(text):
        found = []
        words = text.split()
        for word in words:
                if word == word[::-1]:
                        found.append(word)
        return found

filename = input('insert the name of your file')
f = open(filename, 'r')
txt = f.read()
f.close()
palindromes = getPalindromes(txt)
print(palindromes)
```

ESERCIZI SU MATRICI

Esercizio

- Scrivere un programma che riempia una matrice 3x4 chiedendo all'utente di inserire gli elementi, ma inserendo nella matrice solo gli elementi pari.
- Il programma termina quando la matrice è piena.

Esercizio

- Scrivere un programma che, data una matrice, ne calcola la trasposta

Esercizio

- Scrivere un programma che chiede all'utente di inserire una matrice 3x4, poi (dopo aver terminato la fase di inserimento) copia gli elementi dispari in una seconda matrice 3x4 senza lasciare buchi, se non in fondo.
- Gli elementi in fondo (i "buchi") siano messi a zero.

Esercizio

- Scrivere un programma che chiede all'utente di inserire una matrice NxN con elementi tutti diversi. Se l'utente inserisce un numero già inserito il programma lo avvisa dell'errore e chiede nuovamente di inserire l'elemento.

Esercizio

- Si scriva un programma che chiede all'utente di riempire una matrice, un intero l (che deve essere un intero positivo maggiore di 1) e stampa OK se in m è presente almeno una sequenza orizzontale, verticale o diagonale, di lunghezza l, di elementi che crescono o diminuiscono linearmente (cioè in cui la differenza tra due elementi successivi è costante).
- Esempi di sequenze lineari:
  - 1 2 3 4 (lunghezza 4, differenza costante 1)
  - 4 3 2 1 (lunghezza 4, differenza constante -1)
  - 5 5 5 5 5 5 5 (lunghezza 7, differenza costante 0)

Esercizio

- Una matrice quadrata Mat di dimensioni NxN è diagonalmente dominante se la somma dei valori assoluti degli elementi su ciascuna riga, escluso l'elemento sulla diagonale principale, è minore del valore assoluto dell'elemento corrispondente sulla diagonale principale.
- Scrivere un programma che chiede all'utente di inserire i valori di una matrice e stampa «Dominante» se la matrice è diagonalmente dominante, «Non dominante» altrimenti.
- Si usi la funzione abs(n) che restituisce il valore assoluto dell'n ricevuto come parametro.

Esercizio

- Considerata una matrice A di N x M interi, definiamo *claque* una sottomatrice 2 x 2 in cui la somma algebrica dei valori di una diagonale sia pari a quella dell'altra diagonale. In figura sono evidenziate le claque.
- Si scriva un programma che acquisisce una matrice N x M stampa il numero di claque della matrice.

4	-1	7	0	0
-4	-9	-1	0	0
2	8	16	1	4
-1	7	5	2	5

Esercizio

- Scrivere un programma che chiede all'utente di inserire una matrice NxN e stampa gli elementi di tale matrice secondo un ordinamento a spirale, partendo dalla cornice più esterna e procedendo verso l'interno.

ALTRI ESERCIZI

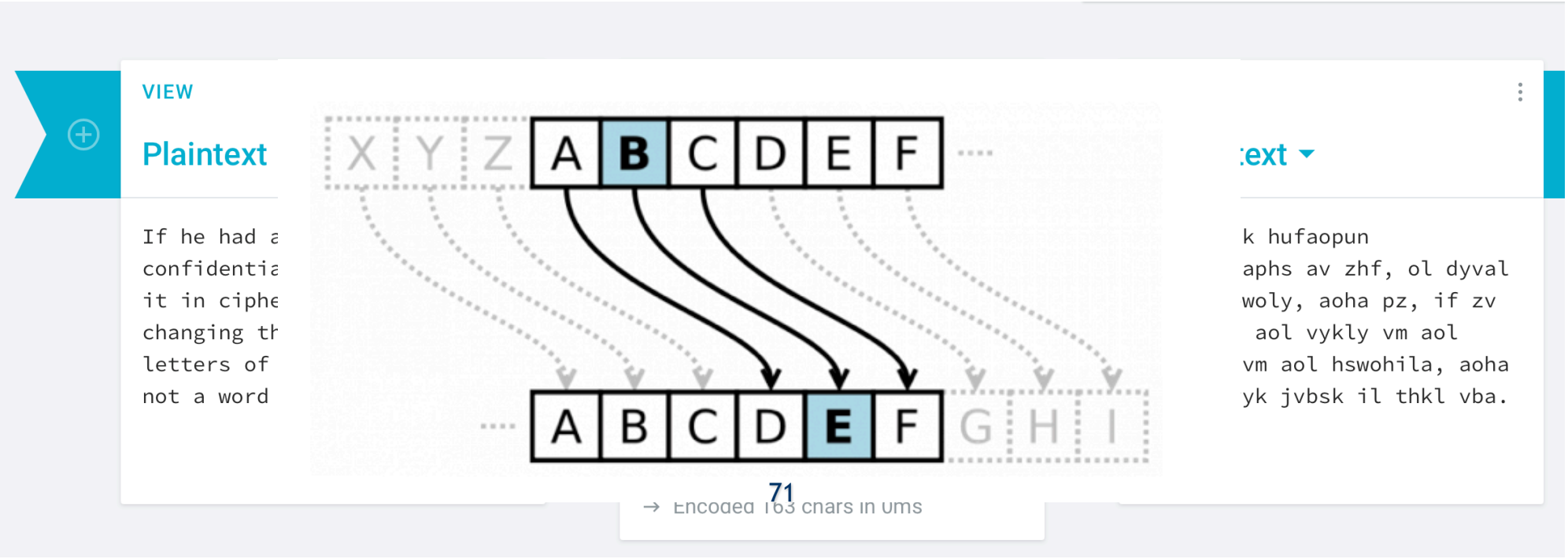
Impiegati

- Riscrivi l'esempio del calcolo del salario per 10 impiegati definendo una funzione che calcola e restituisca al chiamante il salario per un singolo impiegato e utilizzando questa funzione nel programma.
- Estendi il programma in modo che possa leggere da file le informazioni sulle ore di lavoro e sulla paga oraria di ciascun impiegato.

Caesar's code

- Write a program that reads your contacts from a file
- Write a program implementing Caesar's code

https://cryptii.com/pipes/caesar-cipher



SimpleScrabble (1/2)

- *SimpleScrabble* è una versione del gioco *Scrabble* (https://it.wikipedia.org/wiki/Scrabble) dove il valore delle parole dipende solo dalle lettere che le compongono
- Ogni lettera ha un punteggio (da 1 a 10) ed è disponibile in una certa quantità come in tabella
- Una parola è valida se utilizza una quantità di lettere inferiore a quella disponibile, per esempio, la parola quaquaraqua non è valida perchè abbiamo a disposizione una sola q

Lettere in Scrabble		
Lettera	Quantità	Punti
A	14	1
B	3	5
C	6	2
D	3	5
E	11	1
F	3	5
G	2	8
H	2	8
I	12	1
L	5	3
M	5	3
N	5	3
O	15	1
P	3	5
Q	1	10
R	6	2
S	6	2
T	6	2
U	5	3
V	3	5
Z	2	8

SimpleScrabble (2/2)

- Scrivere le seguenti funzioni
  - **valida(...)**, che restituisce 1 se la parola è valida, 0 altrimenti.
  - **punteggio(…)** che calcola e restituisce il valore della parola ricevuta come parametro, assegnando 0 alle parole *non valide*.
- Scrivere uno programma che utilizzi le due funzioni per consentire a due giocatori di alternarsi nella formazione di parole
- Per ogni turno di gioco, il programma chiede al giocatore la parola, controlla che sia valida e ne calcola il punteggio sommandone il valore al punteggio totale di quel giocatore
- Vince il giocatore che arriva per primo ad accumulare un punteggio massimo che viene stabilito all'inizio di ogni partita

Parole totalmente diverse

- Definiamo come *totalmente diverse (TD)* due parole che non hanno lettere in comune. Ad esempio "fuoco" e "aria" sono TD, mentre "fuoco" e "acqua" no (entrambe hanno 'c', anche se in posizioni diverse).
- Si scriva una funzione **td(…)** che riceve come parametri due stringhe e restituisce **1** se le stringhe rappresentano due parole TD, **0** altrimenti
- Facendo uso di td, si scriva un programma che acquisisce dall'utente una sequenza di parole e, al termine della sequenza, stampi il numero di parole della sequenza, e il numero di coppie di parole TD **consecutive**.
- La sequenza può essere terminata da una parola speciale che si chiede all'utente di inserire, per esempio, la parola "fine"

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

Adaptation and extensions Elisabetta Di Nitto, Politecnico di Milano

Other slides have been adapted from material by my colleagues Prof. Sam Guinea, Prof. Alessandro Campi, Prof. Danilo Ardagna Politecnico di Milano