lezione05

October 9, 2023

1 Fondamenti di Programmazione

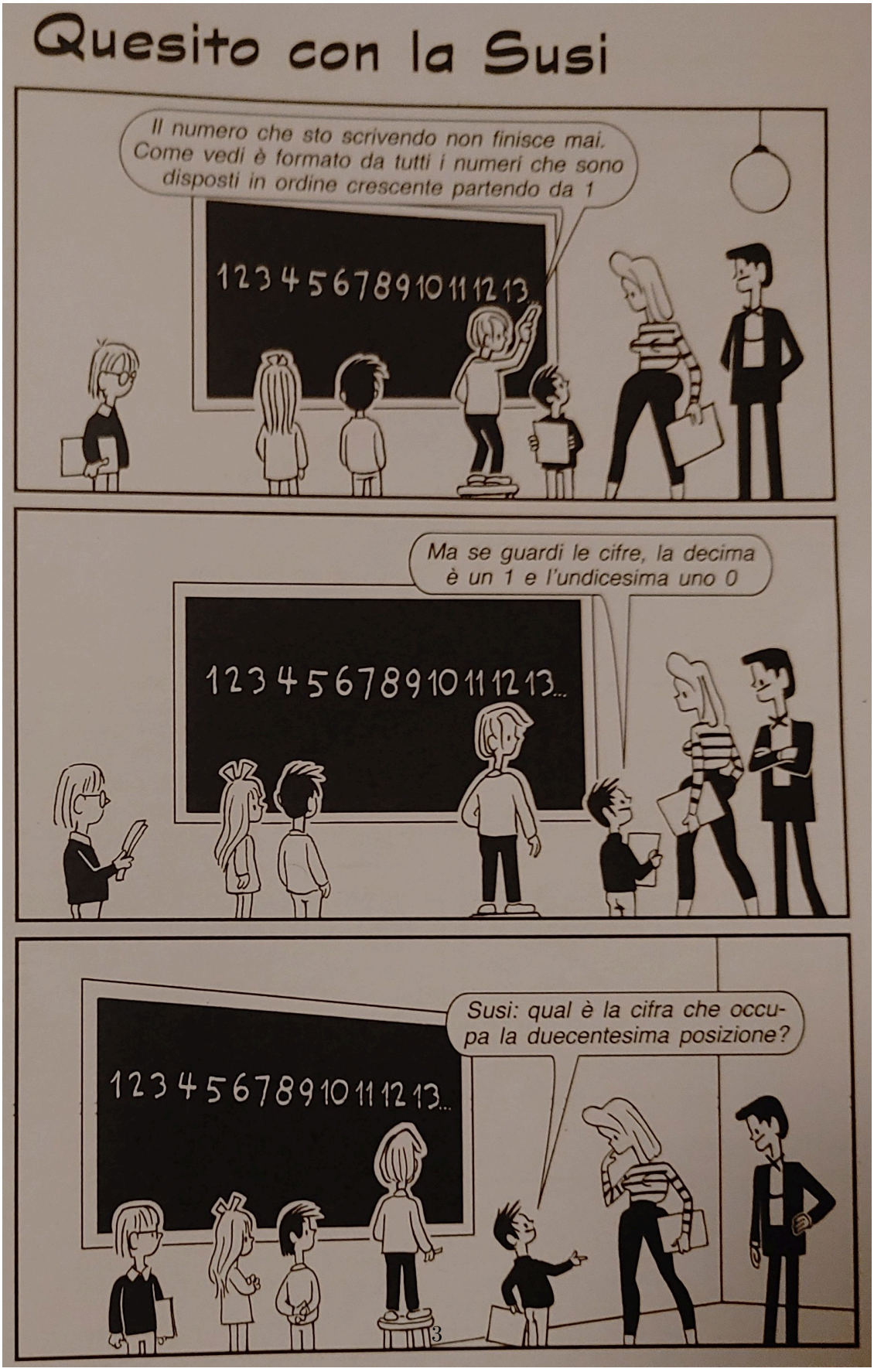
Andrea Sterbini

lezione 5 - 9 ottobre 2023

1.1 RECAP

- **if-elif-else**
- cicli **for**
- cicli **while**
- iteratore **range**
- **identificatori**
- funzioni e argomenti **obbligatori** o **opzionali**
- contenitori: **list, tuple, set, dict**

2 Quesito con la Susi 1 (ed ora un po' di problem-solving)



2.1 Un primo approccio ... costruisco la stringa e trovo il carattere!

- Per trovare la N-esima cifra
  - costruisco la stringa con almeno N cifre
  - ritorno il carattere in posizione N-1
- Per costruire la stringa con almeno N cifre
  - ripeto finchè non ho superato le 200 cifre
    - concateno alla stringa corrente il prossimo intero

Manca qualcosa? - l'inizializzazione delle variabili!

[1]:
```
# Per calcolare la 200° cifra:
    # costruisco una stringa con le cifre (fin dove? 200?)
    # estraggo il 200° carattere (indice = 199)
def cerca_carattere(N):
    testo = ''                    # si parte con una stringa vuota
    i = 1                         # e dal numero 1
    while len(testo) < N:         # e si continua se il numero di cifre è␣
 ↪insufficiente
        testo += str(i)           # a concatenare le cifre del numero corrente
        i += 1                    # ed a incrementarlo
    return testo[N-1]             # e alla fine torniamo il carattere giusto

# Semplice no?
cerca_carattere(20000)
```

[1]:
```
'7'
```

2.1.1 E' una buona soluzione?

non tanto ... produce un mucchio di stringhe!

2.2 Seconda idea che non crea stringhe inutili

- Per calcolare la 200° cifra:
  - trovo in quale numero sta la 200° cifra
  - e quante cifre sono rimaste da scandire
  - la tiro fuori dal numero trovato

[2]:
```
def cerca_senza_stringhe(N):
    numero, mancanti = cerca_numero_e_rimanenti(N)  # ritorno una coppia
    testo = str(numero)
    return testo[mancanti-1]
```

- Per trovare in che numero sta la N-esima cifra
  - scandisco i numeri da 1 in poi e per ciascuno
    - trovo quante cifre ha
    - se sono di meno delle cifre ancora da contare
      - sottraggo le cifre da quelle ancora da contare

    * altrimenti
        · lo ritorno assieme al numero di cifre mancanti

[3]:
```
def cerca_numero_e_rimanenti(N):
    for i in range(1,N):         # scandisco al più N-1 numeri
        cifre = num_cifre(i)     # calcolo le cifre
        if cifre < N:            # se sono poche
            N -= cifre           # le sottraggo
        else:                    # altrimenti ci siamo
            return i, N          # e torno il numero corrente e quante cifre␣
 ↪mancano
```

Ma come trovo quante sono le cifre? - opzione 1: lo trasformo in stringa e conto i caratteri (BLEAH!!!) - opzione 2: ne calcolo il logaritmo in base 10 e sommo 1 - opzione 3: genero le potenze di 10 finchè lo supero

[4]:
```
def opzione_1(X):
    'calcolo il numero di cifre di un numero in base 10 trasformandolo in␣
 ↪stringa'
    return len(str(X))       # genero un sacco di stringhette!!!

opzione_1(100000), opzione_1(99999)
```

[4]:
```
(6, 5)
```

[5]:
```
from math import log10       # uso la funzione log10 del modulo math

def opzione_2(X):
    'calcolo il numero di cifre di un numero in base 10 con i logaritmi'
    # ne calcolo il logaritmo in base 10
    # lo tronco ad intero e gli sommo 1
    return int(log10(X))+1

opzione_2(100000), opzione_2(99999)
```

[5]:
```
(6, 5)
```

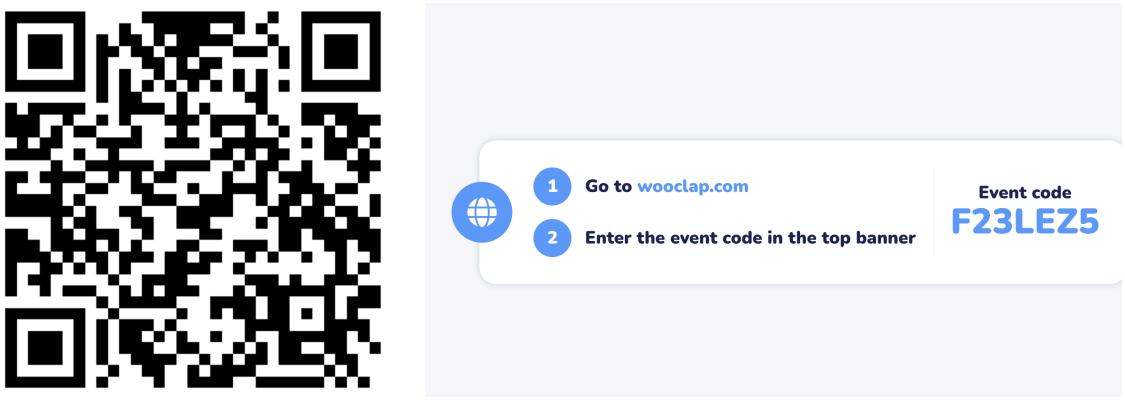
[6]:
```
def opzione_3(X):
    'calcolo il numero di cifre di un numero in base 10 generando le potenze'
    numcifre  = 1
    potenza10 = 10
    while potenza10 <= X:
        potenza10 *= 10
        numcifre  += 1
    return numcifre

opzione_3(100000), opzione_3(99999)
```

[6]: (6, 5)

[7]:
```
N = 20000
num_cifre = opzione_1                       # uso la prima opzione
print(cerca_senza_stringhe(N), end=' ')
num_cifre = opzione_2                       # uso la seconda opzione
print(cerca_senza_stringhe(N), end=' ')
num_cifre = opzione_3                       # uso la terza opzione
print(cerca_senza_stringhe(N), end=' ')
```

```
7 7 7
```

2.3 Momento Wooclap: qual'è la 200 esima cifra?



[8]:
```
# SOLUZIONE
cerca_carattere(200), cerca_senza_stringhe(200), cerca_numero_e_rimanenti(200)
```

[8]: ('0', '0', (103, 2))

2.4 Moduli e import

Per caricare nel namespace GLOBALE le funzionalità realizzate in altri file si usa **`import`**

```
import modulo          # solo 'modulo' viene inserito nel namespace
# chiamare la funzione
modulo.funzione(argomenti)
```

```
from modulo import funzione  # solo 'funzione' -> namespace
# chiamare la funzione
funzione(argomenti)
```

```
from modulo import *       # BAD CODER!!! NON USATELO!!!
# chiamare la funzione
funzione(argomenti)
```

NOTA - A loro volta gli altri moduli possono importare altre librerie, nel proprio namespace di modulo - Se un modulo è stato già importato viene riusato senza rileggerlo di nuovo dal file

3 metodi dei Contenitori

3.0.1 Operazioni sulle liste (list)

- **`elemento in L`** True se l'elemento è presente in L
- **`L1 + L2`** nuova lista concatenazione di L1 e L2
- **`L1 * N`** replicazione N volte e concatenazione
- **`L.append(elemento)`** aggiunta di un elemento alla fine
- **`L[i]`** elemento all'indice **i** (lettura o assegnamento)
- **`L.pop()`** estrazione distruttiva dell'ultimo elemento (ERR se vuota)
- **`L.pop(i)`** estrazione distruttiva dell'elemento all'indice **i** (ERR se **i** non esiste)
- **`L.insert(i, elemento)`** inserisce l'elemento all'indice i (o in cima o in fondo)

[9]:
```
# Esempi di operazioni sulle liste
L1 = [ 1, 2, 3, 4, 5, ]
L2 = [ 11, 22, 33, ]
L1 + L2
```

[9]:
```
[1, 2, 3, 4, 5, 11, 22, 33]
```

[10]:
```
print(L1.pop(2))   # elimino quello a indice 2
L1.insert(78, 55)  # inserisco ad indice 78 il valore 55
L1
```

```
3
```

[10]:
```
[1, 2, 4, 5, 55]
```

[11]:
```
L2[-2:] = []       # elimino gli ultimi due elementi
L2
```

[11]:
```
[11]
```

3.0.2 Altre operazioni sulle liste (list)

- **`L.remove(elemento)`** elimina la **prima occorrenza** dell'elemento (ERR se non presente)
- **`L.index(elemento)`** trova il primo indice in cui c'è l'elemento (ERR se non presente)
- **`L.count(elemento)`** conta l'elemento
- **`L.reverse()`** rovesciamento distruttivo della lista
- **`L.sort()`** ordinamento distruttivo della lista

[12]:
```
# Altri esempi di operazioni sulle liste
L = [10, 43, 92, 1, -43, 90, -200, int('1') ]
L.count(3)
```

[12]:
```
0
```

[13]:
```
L.index(-43)
```

[13]:
```
4
```

[14]:
```
L.sort()    # riordinamento DISTRUTTIVO (modifica la lista)
L
```

[14]:
```
[-200, -43, 1, 1, 10, 43, 90, 92]
```

[15]:
```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(…)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
```

```
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
|      Return a reverse iterator over the list.
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  __setitem__(self, key, value, /)
|      Set self[key] to value.
|
|  __sizeof__(self, /)
|      Return the size of the list in memory, in bytes.
|
|  append(self, object, /)
|      Append object to the end of the list.
|
|  clear(self, /)
|      Remove all items from list.
|
|  copy(self, /)
|      Return a shallow copy of the list.
|
|  count(self, value, /)
|      Return number of occurrences of value.
```

```
 |
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  insert(self, index, object, /)
 |      Insert object before index.
 |
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |
 |  sort(self, /, *, key=None, reverse=False)
 |      Sort the list in ascending order and return None.
 |
 |      The sort is in-place (i.e. the list itself is modified) and stable (i.e.
the
 |      order of two equal elements is maintained).
 |
 |      If a key function is given, apply it once to each list item and sort
them,
 |      ascending or descending, according to their function values.
 |
 |      The reverse flag can be set to sort in descending order.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(…) from builtins.type
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
```

```
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

3.0.3 Operazioni sulle tuple

- **`elemento in T`** True se l'elemento è presente in T
- **`T1 + T2`** nuova tupla concatenazione di T1 e T2
- **`T * N`** replicazione N volte e concatenazione
- **`T[i]`** accesso all'elemento all'indice i (SOLO lettura)
- **`L.index(elemento)`** trova il primo indice in cui c'è l'elemento (ERR se non presente)
- **`L.count(elemento)`** conta l'elemento

[58]:
```
# Esempi di operazioni sulle tuple
T1 = 1, 2, 3, 4
T2 = 24, 52, 67, 31
T1 + T2
T1[2] = 55
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 5
      3 T2 = 24, 52, 67, 31
      4 T1 + T2
----> 5 T1[2] = 55

TypeError: 'tuple' object does not support item assignment
```

[17]:
```
T1[1:3]
```

[17]:
```
(2, 3)
```

[18]:
```
help(tuple)
```

```
Help on class tuple in module builtins:

class tuple(object)
 |  tuple(iterable=(), /)
 |
 |  Built-in immutable sequence.
 |
 |  If no argument is given, the constructor returns an empty tuple.
 |  If iterable is specified the tuple is initialized from iterable's items.
 |
```

```
|  If the argument is a tuple, the return value is the same object.
|
|  Built-in subclasses:
|      asyncgen_hooks
|      UnraisableHookArgs
|
|  Methods defined here:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __mul__(self, value, /)
|      Return self*value.
```

```
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __rmul__(self, value, /)
|      Return value*self.
|
|  count(self, value, /)
|      Return number of occurrences of value.
|
|  index(self, value, start=0, stop=9223372036854775807, /)
|      Return first index of value.
|
|      Raises ValueError if the value is not present.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(…) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
```

3.0.4 Operazioni sugli insiemi (set)

- **`elemento in S`** True se elemento è presente in S
- **`S1 | S2`** oppure **`S1.union(S2)`** unione (or)
- **`S1 & S2`** oppure **`S1.intersection(S2)`** intersezione (and, elementi in comune)
- **`S1 - S2`** oppure **`S1.difference(S2)`** insieme degli el. di S1 che non sono in S2
- **`S1 ^ S2`** oppure **`S1.symmetric_difference(S2)`** insieme degli elementi NON in comune (xor)
- **`S.pop()`** estrazione distruttiva di un el. (ERR if empty)
- **`S.add(elemento)`** aggiunta di un elemento
- **`S.remove(elemento)`** rimozione di un el. (ERR if missing)
- **`S1.update(S2)`** come **`S1 |= S2`** (distruttiva)

[19]:
```
# Esempi di operazioni sugli insiemi
S1 = set(range(3,101,3))          # insieme dei multipli di 3 in [1..100]
S2 = set(range(1,101)) - S1       #         NON multipli di 3
print('S1 =',S1)
print('S2 =',S2)
```

```
S1 = {3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,
60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99}
S2 = {1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 22, 23, 25, 26, 28, 29,
31, 32, 34, 35, 37, 38, 40, 41, 43, 44, 46, 47, 49, 50, 52, 53, 55, 56, 58, 59,
61, 62, 64, 65, 67, 68, 70, 71, 73, 74, 76, 77, 79, 80, 82, 83, 85, 86, 88, 89,
91, 92, 94, 95, 97, 98, 100}
```

[20]:
```
help(set)
```

```
Help on class set in module builtins:

class set(object)
 |  set() -> new empty set object
 |  set(iterable) -> new set object
 |
 |  Build an unordered collection of unique elements.
 |
 |  Methods defined here:
 |
 |  __and__(self, value, /)
 |      Return self&value.
 |
 |  __contains__(…)
 |      x.__contains__(y) <==> y in x.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iand__(self, value, /)
 |      Return self&=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __ior__(self, value, /)
 |      Return self|=value.
 |
 |  __isub__(self, value, /)
 |      Return self-=value.
```

```
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __ixor__(self, value, /)
|      Return self^=value.
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __rand__(self, value, /)
|      Return value&self.
|
|  __reduce__(…)
|      Return state information for pickling.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __ror__(self, value, /)
|      Return value|self.
|
|  __rsub__(self, value, /)
|      Return value-self.
|
|  __rxor__(self, value, /)
|      Return value^self.
|
|  __sizeof__(…)
|      S.__sizeof__() -> size of S in memory, in bytes
|
|  __sub__(self, value, /)
|      Return self-value.
|
|  __xor__(self, value, /)
|      Return self^value.
```

```
|
|  add(…)
|      Add an element to a set.
|
|      This has no effect if the element is already present.
|
|  clear(…)
|      Remove all elements from this set.
|
|  copy(…)
|      Return a shallow copy of a set.
|
|  difference(…)
|      Return the difference of two or more sets as a new set.
|
|      (i.e. all elements that are in this set but not the others.)
|
|  difference_update(…)
|      Remove all elements of another set from this set.
|
|  discard(…)
|      Remove an element from a set if it is a member.
|
|      If the element is not a member, do nothing.
|
|  intersection(…)
|      Return the intersection of two sets as a new set.
|
|      (i.e. all elements that are in both sets.)
|
|  intersection_update(…)
|      Update a set with the intersection of itself and another.
|
|  isdisjoint(…)
|      Return True if two sets have a null intersection.
|
|  issubset(…)
|      Report whether another set contains this set.
|
|  issuperset(…)
|      Report whether this set contains another set.
|
|  pop(…)
|      Remove and return an arbitrary set element.
|      Raises KeyError if the set is empty.
|
|  remove(…)
|      Remove an element from a set; it must be a member.
```

```
|
|      If the element is not a member, raise a KeyError.
|
|  symmetric_difference(…)
|      Return the symmetric difference of two sets as a new set.
|
|      (i.e. all elements that are in exactly one of the sets.)
|
|  symmetric_difference_update(…)
|      Update a set with the symmetric difference of itself and another.
|
|  union(…)
|      Return the union of sets as a new set.
|
|      (i.e. all elements that are in either set.)
|
|  update(…)
|      Update a set with the union of itself and others.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  __class_getitem__(…) from builtins.type
|      See PEP 585
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __hash__ = None
```

3.0.5 Operazioni sui dizionari (dict)

- **`key in D`** True se la chiave è presente
- **`D[key]`** accesso al valore con chiave key (R/W) (ERR se non presente)
- **`D.keys()`** elenco delle chiavi (che sono uniche)
- **`D.values()`** elenco dei valori (con duplicati)
- **`D.items()`** elenco delle coppie (chiave, valore)
- **`D.popitem()`** torna l'ultima coppia (k,v) e la rimuove
- **`D1 | D2`** nuovo dizionario con tutte le coppie di D1 **e** di D2 (prese nell'ordine)
- **`D1.update(D2)`** modifica D1 aggiungendo le coppie (K,V) di D2 nell'ordine

[21]:
```
# Esempi di operazioni sui dizionari
D1 = { 1: 'uno', 2: 'due', 3:'tre' }
D2 = { 11: 'undici', 2: 'ventidue', 33:'trentatre' }
D = D1  | D2    # creo un dizionario con tutte le voci di D1 e D2
D
```

[21]:
```
{1: 'uno', 2: 'ventidue', 3: 'tre', 11: 'undici', 33: 'trentatre'}
```

3.0.6 Altre operazioni sui dizionari

- **D.get(key, default)** se la chiave è presente ne torna il valore, altrimenti torna il valore di default
- **D.setdefault(key, default)** se la chiave è presente ne torna il valore altrimenti la inserisce col valore di default (e lo torna)
- **D.pop(key, default)** se la chiave è presente ne torna il valore e la rimuove, altrimenti torna il valore di default (o ERRORE se no default)
- **D.fromkeys(keys, value)** costruisce un dizionario con le chiavi fornite, tutte associate allo stesso valore indicato

[22]:
```
# Esempi di operazioni sui dizionari
D.get(66, 'non trovato')
```

[22]:
```
'non trovato'
```

[23]:
```
dict.fromkeys('ABCADEF', 0)
```

[23]:
```
{'A': 0, 'B': 0, 'C': 0, 'D': 0, 'E': 0, 'F': 0}
```

[24]:
```
help(dict)
```

```
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
 |
 |  Built-in subclasses:
 |      StgDict
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
```

```
|      True if the dictionary has the specified key, else False.
|
|  __delitem__(self, key, /)
|      Delete self[key].
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(…)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __ior__(self, value, /)
|      Return self|=value.
|
|  __iter__(self, /)
|      Implement iter(self).
|
|  __le__(self, value, /)
|      Return self<=value.
|
|  __len__(self, /)
|      Return len(self).
|
|  __lt__(self, value, /)
|      Return self<value.
|
|  __ne__(self, value, /)
|      Return self!=value.
|
|  __or__(self, value, /)
|      Return self|value.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __reversed__(self, /)
```

```
 |      Return a reverse iterator over the dict keys.
 |
 |  __ror__(self, value, /)
 |      Return value|self.
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __sizeof__(…)
 |      D.__sizeof__() -> size of D in memory, in bytes
 |
 |  clear(…)
 |      D.clear() -> None.  Remove all items from D.
 |
 |  copy(…)
 |      D.copy() -> a shallow copy of D
 |
 |  get(self, key, default=None, /)
 |      Return the value for key if key is in the dictionary, else default.
 |
 |  items(…)
 |      D.items() -> a set-like object providing a view on D's items
 |
 |  keys(…)
 |      D.keys() -> a set-like object providing a view on D's keys
 |
 |  pop(…)
 |      D.pop(k[,d]) -> v, remove specified key and return the corresponding
value.
 |
 |      If the key is not found, return the default if given; otherwise,
 |      raise a KeyError.
 |
 |  popitem(self, /)
 |      Remove and return a (key, value) pair as a 2-tuple.
 |
 |      Pairs are returned in LIFO (last-in, first-out) order.
 |      Raises KeyError if the dict is empty.
 |
 |  setdefault(self, key, default=None, /)
 |      Insert key with a value of default if key is not in the dictionary.
 |
 |      Return the value for key if key is in the dictionary, else default.
 |
 |  update(…)
 |      D.update([E, ]**F) -> None.  Update D from dict/iterable E and F.
 |      If E is present and has a .keys() method, then does:  for k in E: D[k] =
E[k]
```

```
 |      If E is present and lacks a .keys() method, then does:  for k, v in E:
D[k] = v
 |      In either case, this is followed by: for k in F:  D[k] = F[k]
 |
 |  values(…)
 |      D.values() -> an object providing a view on D's values
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(…) from builtins.type
 |      See PEP 585
 |
 |  fromkeys(iterable, value=None, /) from builtins.type
 |      Create a new dictionary with keys from iterable and values set to value.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

4 Quesito con la Susi 2

- un amico della Susi ha fatto un esame a crocette True/False con 100 domande
- le risposte giuste **True** erano tutte quelle **multiple di 3** e il resto **False**
- ma lui ha marcato **False** tutte le domande **multiple di 4** ed il resto **True**

Quante ne ha azzeccate?

4.1 ancora problem-solving

- per contare quelle azzeccate
  - scandisco i numeri da 1 a 100 e per ciascuno
    - calcolo la risposta corretta
    - calcolo la risposta data
    - se sono uguali incremento il conteggio
  - torno il conteggio

(cosa manca? l'inizializzazione delle variabili!)

[25]:
```
# per contare le risposte giuste
def conta_azzeccate():
    # inizializzo una variabile per contare quelle azzeccate a 0
    azzeccate = 0
    # scandisco i numeri da 1 a 100
    for i in range(1,101):
        # calcoliamo la risposta corretta
        corretta = risposta_corretta(i)
        # calcoliamo la risposta data
        data     = risposta_data(i)
        # se la riposta data è uguale alla risposta corretta
```

```
        if corretta == data:
            # incremento il conteggio
            azzeccate += 1
    # alla fine torno il conteggio
    return azzeccate
```

- per calcolare la risposta corretta X-esima
  - torno True se **X è multiplo di 3**

[26]:
```
# per calcolare la risposta corretta
def risposta_corretta(X):
    # se il numero è divisibile per 3
        # la risposta corretta è True
    # altrimenti
        # la risposta corretta è False
    return X%3 == 0
```

- per calcolare la risposta data X-esima
  - torno True se **X NON è multiplo di 4**

[27]:
```
# per calcolare la risposta data
def risposta_data(X):
    # se il numero è divisibile per 4
        # lui ha risposto False
    # altrimenti
        # lui ha risposto True
    return X % 4 != 0

# Noooo Non vi do subito il risultato 3-)
```

4.2 Un modo completamente diverso: con gli insiemi

- costruisco i due insiemi delle risposte **corrette_True** e **corrette_False**
- costruisco i due insiemi delle risposte **date_True** e **date_False**
- le risposte azzeccate sono:
  - le **date_True** che sono **corrette_True**
  - e le **date_False** che sono **corrette_False**

[28]:
```
# un altro modo di rispondere: con gli insiemi
# raccolgo le risposte corrette
#   in due insiemi corrette_True e corrette_False
domande       = set(range(1,101))        # i numeri da 1 a 100
corrette_True = set(range(3,101,3))      # i multipli di 3
corrette_False= domande - corrette_True  # i non multipli di 3

# raccolgo le risposte date     True
#   in due insiemi date_True e date_False
date_False = set(range(4,101,4))
```

```
date_True = domande - date_False
```

[29]:
```
# Le risposte giuste sono:
  # l'intersezione delle risposte date_False con le risposte corrette_False
  # assieme a
  # l'intersezione delle risposte date_True con le risposte corrette_True
azzeccate = (corrette_True & date_True) | (corrette_False & date_False)

# il risultato cercato è il numero di elementi dell'insieme

# Noooo ... non ve lo dico, prima dovete risolverlo voi
```

4.2.1 Ma se invece degli insiemi usassimo dei dizionari domanda->risposta ?

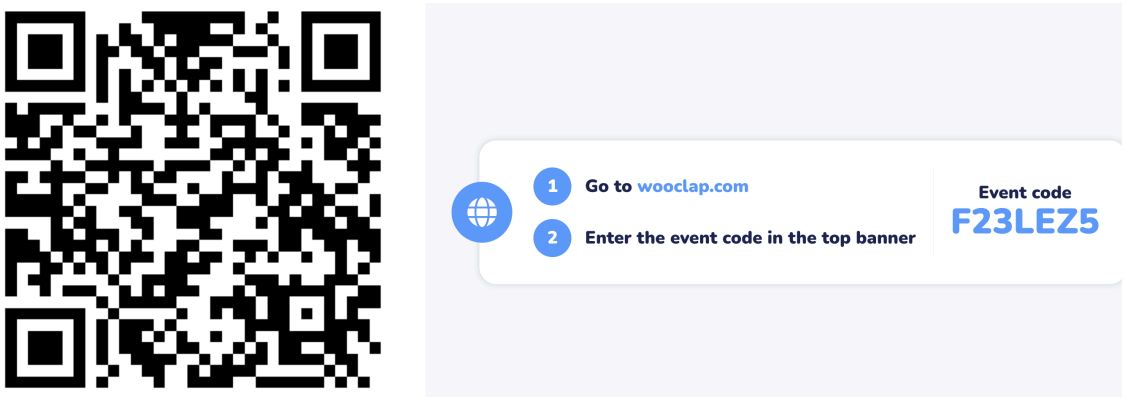
- costruisco il dizionario **corrette** che contiene **domanda->risposta corretta**
- costruisco il dizionario **date** che contiene **domanda->risposta data**
- confronto i dizionari

[30]:
```
## Un terzo modo usando i dizionari
# costruisco i dizionari
#     numero -> risposta giusta
#     numero -> risposta data
dizionario_risposte = {}
for i in range(1, 101):
    dizionario_risposte[i] = i % 3 == 0
dizionario_date = {}
for i in range(1, 101):
    dizionario_date[i] = i % 4 != 0
```

[31]:
```
# e poi calcolo l'intersezione dei due insiemi di items (domanda, risposta)
azzeccate_diz = set(dizionario_risposte.items()) & set(dizionario_date.items())
# La risposta dopo il break
print(azzeccate)
print(azzeccate_diz)
```

```
{3, 4, 6, 8, 9, 15, 16, 18, 20, 21, 27, 28, 30, 32, 33, 39, 40, 42, 44, 45, 51,
52, 54, 56, 57, 63, 64, 66, 68, 69, 75, 76, 78, 80, 81, 87, 88, 90, 92, 93, 99,
100}
{(69, True), (4, False), (45, True), (92, False), (27, True), (3, True), (28,
False), (93, True), (51, True), (8, False), (52, False), (32, False), (76,
False), (18, True), (56, False), (100, False), (80, False), (9, True), (99,
True), (75, True), (42, True), (57, True), (33, True), (66, True), (90, True),
(16, False), (81, True), (6, True), (39, True), (15, True), (40, False), (20,
False), (64, False), (21, True), (30, True), (44, False), (88, False), (68,
False), (87, True), (63, True), (78, True), (54, True)}
```

4.2.2 Wooclap: secondo voi quante risposte ha azzeccato ?



[32]:
```
# vediamo finalmente quante ne ha azzeccate
print('azzeccate', conta_azzeccate())
len(azzeccate)
```

```
azzeccate 42
```

[32]: 42

5 Ancora argomenti formali delle definizioni delle funzioni

Avrete notato che la funzione **print** accetta un numero indefinito di argomenti

Ma come fa? E' speciale?

No, approfitta della sintassi degli **assegnamenti multipli** e del **packing/unpacking**

5.0.1 Assegnamento multiplo? (WTF?)

elenco di variabili = contenitore con stesso numero di elementi

Ciascuna variabile viene assegnata, **nell'ordine**, con i valori elencati a destra dell' **=**

- **PRIMA** viene calcolato tutto il contenitore a destra
- **POI** viene fatto l'assegnamento

[33]:
```
A, B = 1, 2
A, B = B, A             # fa uno SCAMBIO!
print(A, B)
```

```
2 1
```

5.0.2 Ci permette di gestire facilmente gruppi di dati coerenti (struct o record in altri linguaggi)

[34]:
```
# p.es. se abbiamo altezza (H), larghezza (L) e profondità (P)
# di una scatola in quest'ordine in una lista,
dimensioni_scatola = [ 10, 20, 30, ]
# per stamparli dovremmo prima estrarli
H = dimensioni_scatola[0]
L = dimensioni_scatola[1]
P = dimensioni_scatola[2]
# poi stamparli
print('altezza',H)
print('larghezza',L)
print('profondità',P)
# ma che strazio!!!! Bisogna ricordarsi gli indici! (0, 1, 2)
```

```
altezza 10
larghezza 20
profondità 30
```

[35]:
```
# MOOOLTO meglio!
H, L, P = dimensioni_scatola # li estraggo in ordine e li stampo
print('altezza',H)
print('larghezza',L)
print('profondità',P)
# NOTA: NON dobbiano ricordarci la posizione,
#       basta mettere le variabili nell'ordine giusto
```

```
altezza 10
larghezza 20
profondità 30
```

5.1 "packing" ed "unpacking"

In un assegnamento possiamo raccogliere in una sola variabile tutti i valori rimanenti di una lista (**packing**)

Il nome della variabile deve essere preceduto da asterisco *****

NOTA il packing avviene nelle variabili a SINISTRA di un assegnamento

[36]:
```
# PACKING: raccolgo in C tutti i valori
# tranne i primi due che vanno in A e B
# mettendo un asterisco subito prima del nome della variabile C
A, B, *C = [ 1, 2, 3, 4, 5, 6]
print('A:',A)
print('B:',B)
print('C:',C)
```

```
A: 1
```

```
B: 2
C: [3, 4, 5, 6]
```

[37]:
```
# Esempio di packing che raccoglie *lista* vuota
A, B, *C = (1,2)
print('C ora è', C)
```

```
C ora è []
```

[38]:
```
# ancora meglio, posso mettere la variabile "packed" all'inizio!!!
*A, B, C = [ 1, 2, 3, 4, 5, 6]
print('A:',A)
print('B:',B)
print('C:',C)
```

```
A: [1, 2, 3, 4]
B: 5
C: 6
```

[39]:
```
# oppure in mezzo!!!
A, *B, C = [ 1, 2, 3, 4, 5, 6]
print('A:',A)
print('B:',B)
print('C:',C)
```

```
A: 1
B: [2, 3, 4, 5]
C: 6
```

Basta che ci sia una sola variabile "packed" in modo che Python sappia cosa mettere nelle altre, così metterà il resto in quella con asterisco

[40]:
```
## Quindi la definizione di print dev'essere simile a

def my_print(*roba_da_stampare, end='\n', sep=' '):
    # roba_da_stampare è una *tupla* con tutti i valori passati
    # NOTA: print prima converte tutto in stringhe
    stringhe = []
    for valore in roba_da_stampare:
        stringhe.append(str(valore))
    # e poi stampa (qui uso print per semplicità)
    print(sep.join(stringhe), end=end)

my_print(1, 1.3, [1, 2, 3])
```

```
1 1.3 [1, 2, 3]
```

5.1.1 E l' "unpacking" cos'è?

Sempre con l'asterisco possiamo "spacchettare" una variabile che contiene più valori all'interno di una altra **espressione**

```
X = sequenza
[ ..., *X, ... ]
diventa
[ ..., elementi di X, ... ]
```

NOTA l'unpacking avviene quando si **calcola** il valore della variabile

[41]:
```
# esempio di unpacking di un *set*
X = {11, 22, 33, 44}
D = 1, 2, *X, 4       # nella tupla appaiono gli elementi di X
print('D ora è', D)
```

```
D ora è (1, 2, 33, 11, 44, 22, 4)
```

[42]:
```
# Esempio di unpacking di una sequenza di caratteri
X = 'ABCDE'
[ 1, 2, 3, 4, *X, 5, 6, 7, 8, ]    # la espando in mezzo all'altra sequenza
```

[42]:
```
[1, 2, 3, 4, 'A', 'B', 'C', 'D', 'E', 5, 6, 7, 8]
```

5.1.2 RIASSUMENDO: se un asterisco precede il nome di una variabile:

- in un **assegnamento** fa il **packing** di zero o più valori in una **lista** (**tupla** se sono gli argomenti di una funzione)
- in una **espressione** fa l'**unpacking** dei valori contenuti nella variabile

5.1.3 Wooclap: quanto vi è chiaro ?