

Il linguaggio Python

Liceo G.B. Brocchi

Classe 3AQSA – Compresenza Informatica - Arte
Bassano del Grappa, Dicembre 2022

Names e oggetti

- In Python, gli oggetti sono allocati in memoria quando il codice lo richiede, e deallocati automaticamente quando non servono più
- Un meccanismo interno a Python chiamato **garbage collection** provvede a liberare la memoria allocata dagli oggetti che non servono più
- Come fa Python a capire che un oggetto non serve più?
un oggetto non serve più quando non è più 'puntato' da alcun name

gli oggetti hanno un tipo e una identità, ossia l'indirizzo che identifica l'area di memoria in cui risiedono

Tipo e identità di un oggetto

```
ref = 30
```

```
print('type of ref is (__class__) ', ref.__class__)
```

```
print('type of ref is (type) ', type(ref))
```

```
print('memory address of ref is ', hex(id(ref)))
```

```
type of ref is (__class__) <class 'int'>
```

```
type of ref i (type) <class 'int'>
```

```
memory address of ref is 0x7f8878760490
```

Names e oggetti

var = 35



- *var* è un *name* (o *riferimento*, o *label*) che punta ad un'area di memoria contenente 35, di tipo *int*. Python è tipizzato dinamicamente: i tipi degli oggetti vengono stabiliti a *runtime*, ossia quando il programma è in esecuzione
- cosa succede se riassegniamo *var*? Che fine fa l'area di memoria che contiene il numero intero 35?

Names e oggetti

var = 67

var

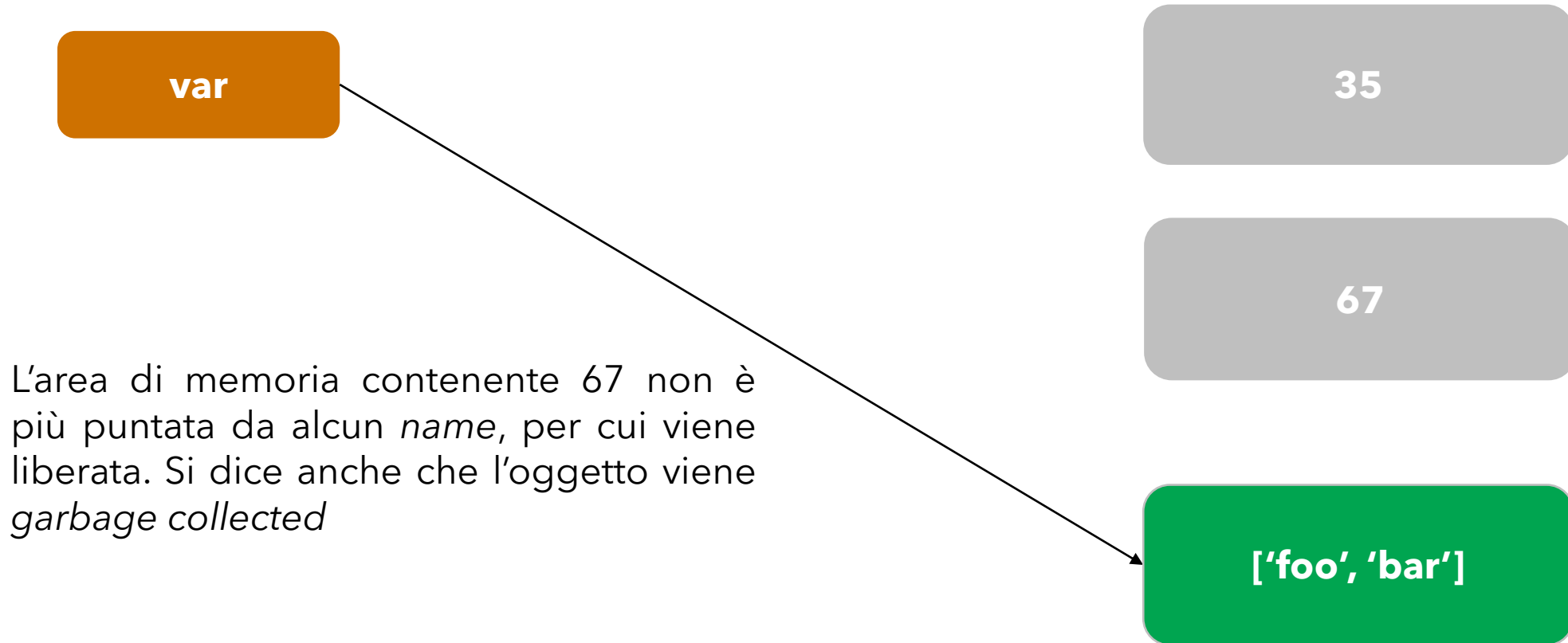
35

L'area di memoria contenente 35 non è più puntata da alcun *name*, per cui viene liberata. Si dice anche che l'oggetto viene *garbage collected*

67

Names e oggetti

```
var = ['foo', 'bar']
```



I tipi *built-in* «sequenza»: le stringhe

- Le stringhe sono contenitori ordinati (*sequenze*) di caratteri
- Le stringhe (e le sequenze in generale) sono ordinate perché ad ogni elemento (carattere) è associato un indice crescente, partendo dall'elemento più a sinistra con indice 0

```
pres1 = 'george bush'
pres2 = "barack obama"
pres3 = '''donald trump'''
pres4_multiple_lines = 'donald \
trump' #a string on two lines
```

```
pres5 = "joe \
biden" #a string on two lines
```

I metodi del tipo str

```
Python 3.10.6 (main, Nov  2 2022, 18:53:38) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> str.
```

str.capitalize()	str.format()	str.isidentifier()	str.ljust()	str.replace()
str.splitlines()				
str.casefold()	str.format_map()	str.islower()	str.lower()	str.rfind()
str.startswith()				
str.center()	str.index()	str.isnumeric()	str.lstrip()	str.rindex()
str.strip()				
str.count()	str.isalnum()	str.isprintable()	str.maketrans()	str.rjust()
str.swapcase()				
str.encode()	str.isalpha()	str.isspace()	str.mro()	str.rpartition()
str.title()				
str.endswith()	str.isascii()	str.istitle()	str.partition()	str.rsplitleft()
str.translate()				
str.expandtabs()	str.isdecimal()	str.isupper()	str.removeprefix()	str.rstrip()
str.upper()				
str.find()	str.isdigit()	str.join()	str.removesuffix()	str.split()
str.zfill()				

Documentazione del tipo str: `help(str)`

Help on class str in module builtins:

```
class str(object)
|   str(object='') -> str
|   str(bytes_or_buffer[, encoding[, errors]]) -> str
|
|   Create a new string object from the given object. If encoding or
|   errors is specified, then the object must expose a data buffer
|   that will be decoded using the given encoding and error handler.
|   Otherwise, returns the result of object.__str__() (if defined)
|   or repr(object).
|   encoding defaults to sys.getdefaultencoding().
|   errors defaults to 'strict'.
|
|   Methods defined here:
|
|   __add__(self, value, /)
|       Return self+value.
|
|   __contains__(self, key, /)
|       Return key in self.
|
|   __eq__(self, value, /)
|       Return self==value.
|
|   __format__(self, format_spec, /)
|       Return a formatted version of the string as described by format_spec.
|
|   __ge__(self, value, /)
```

I metodi del tipo str

```
pres1 = 'george bush'
```

```
print(pres1.index('e')) # 'e' occurs at index 1 for the first time
```

```
print(pres1[5]) # prints 'e'
```

```
pres1 = 'george bush'
```

```
print(pres1[15])
```

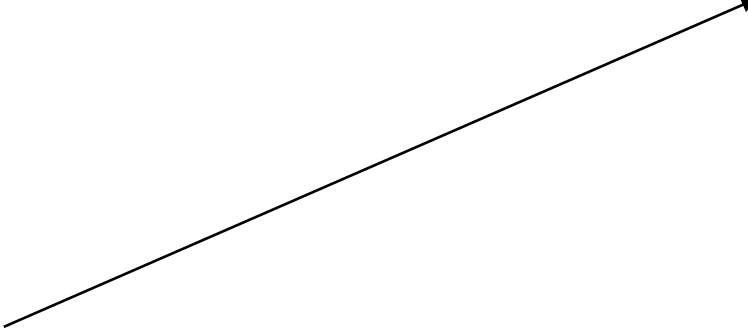
Traceback (most recent call last):

File "/home/cyofanni/Desktop/high-school-cs-class/python/examples.py", line 2, in <module>

```
    print(pres1[15])
```

IndexError: string index out of range

i linguaggi a livello molto alto, a differenza di C e C++ effettuano controlli sui limiti dei contenitori



I metodi del tipo str

text = '''Monty Python (also collectively known as the Pythons) were a \ British comedy troupe who created the sketch comedy television show Monty Python\'s Flying Circus, \ which first aired on the BBC in 1969 Forty-five episodes were made over four series.'''

The Python phenomenon developed from the television series into something larger in scope and influence, \ including touring stage shows, films, albums, books and musicals.'''

```
print(text)
```

```
print(text.upper()) #uppercase the string
```

```
print(text.startswith('Monty')) #does it start with the substring 'Monty'? --> True
```

```
print(text.startswith('monty')) #does it start with the substring 'Monty'? --> False
```

```
words = text.split(' ')
```

```
print(words.__class__) #the split method returns a list
```

Le stringhe sono oggetti immutabili

```
>>> s = 'montypython'
>>> s[0]
'm'
>>> s[0] = 'M'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

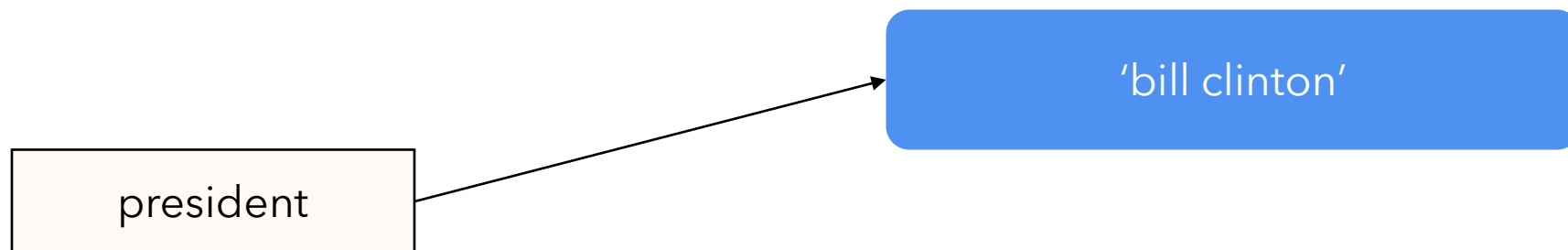
**il messaggio di errore di 'str' object does not support item assignment
significa che un oggetto di tipo str non è modificabile.
Si dice che le stringhe in Python sono immutabili**

**Questo non significa che non si possa riassegnare una riferimento ad una stringa
Un riferimento può sempre essere riassegnato (vedi la slide successiva)!**

Le stringhe sono oggetti immutabili

```
>>> president = 'bill clinton'  
>>> hex(id(president))  
'0x7f13b2c73730'  
>>> president = 'john kennedy'  
>>> hex(id(president))  
'0x7f13b2c737f0'  
>>>
```

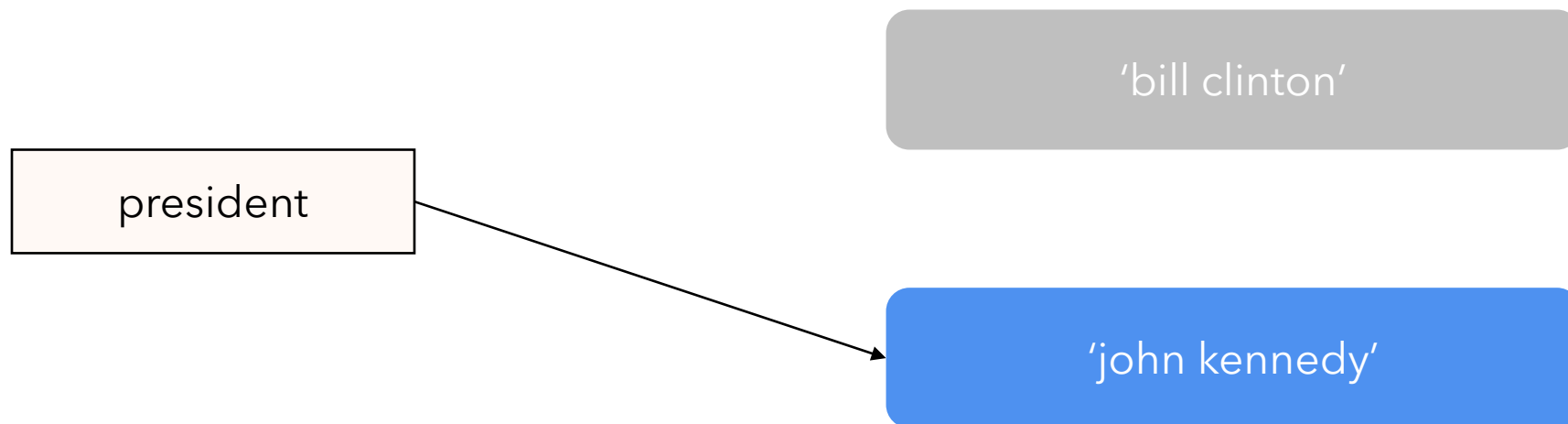
Qui è avvenuta una riassegnazione del riferimento `president`. Leggendo gli indirizzi si capisce che le stringhe `'bill clinton'` e `'john kennedy'` sono in due aree di memoria diverse. La stringa `'bill clinton'` viene deallocata in quanto non ha più riferimenti che la puntano



Le stringhe sono oggetti immutabili

```
>>> president = 'bill clinton'  
>>> hex(id(president))  
'0x7f13b2c73730'  
>>> president = 'john kennedy'  
>>> hex(id(president))  
'0x7f13b2c737f0'  
>>>
```

Qui è avvenuta una riassegnazione del riferimento `president`. Leggendo gli indirizzi si capisce che le stringhe `'bill clinton'` e `'john kennedy'` sono in due aree di memoria diverse. La stringa `'bill clinton'` viene deallocata in quanto non ha più riferimenti che la puntano



I tipi *built-in* «sequenza»: le liste

- Le liste sono contenitori ordinati (*sequenze*) di oggetti
- Le liste (e le sequenze in generale) sono ordinate perché ad ogni elemento è associato un indice crescente, partendo dall'elemento più a sinistra con indice 0
- Le liste sono oggetti **mutabili**, quindi è possibile riassegnare gli item di una lista

```
composers = ['bach', 'mozart', 'beethoven', 'brahms']
```

```
print('a list of', len(composers), 'composers')  
print('the first composer is', composers[0].capitalize())  
composers[3] = 'shostakovich'  
print(composers)
```

riassegnazione di un
item della lista
composers

I tipi *built-in* «sequenza»: le liste

- Inizializzare una lista vuota
- Riempirla con gli interi da 1 a 64 (estremi inclusi) utilizzando un ciclo while e il metodo append
- Stampare l'indirizzo della lista in esadecimale, prima di entrare nel ciclo e dopo l'istruzione append
- Stampare la lista risultante

I tipi *built-in* «sequenza»: le liste

```
ls = []  
print(hex(id(ls)))  
i = 1  
while i <= 2**6:  
    ls.append(i)  
    print(hex(id(ls)))  
    i += 1  
  
print(ls)
```

Il costrutto for di Python

```
lst = [1, 2, 3, 4]
print('lst before for loop: ', lst)
for item in lst:
    item = item * 2
print('lst after for loop: ', lst)
```

lst viene modificata nel ciclo for?

Il costrutto for di Python

```
lst = [1, 2, 3, 4]
print('lst before for loop: ', lst)
for item in lst:
    item = item * 2
print('lst after for loop: ', lst)
```

```
lst before for loop: [1, 2, 3, 4]
lst after for loop:  [1, 2, 3, 4]
```

lst non è stata modificata. Perché? Perché item è una copia dell'elemento corrente della lista, e non l'elemento stesso

Un for più familiare e vecchio stampo

```
for ind in range(len(lst)):
    lst[ind] = lst[ind] * 2
print('lst after old-style loop: ', lst)
```

lst after old-style loop: [2, 4, 6, 8]

Un altro tipo «sequenza»: le tuple

- Le tuple vengono utilizzate per tenere insieme più oggetti, anche di tipo diverso
- A differenza delle liste, **le tuple sono immutabili**
- È meglio delimitare gli elementi di una tupla con le parentesi tonde. Comunque non è necessario.
- L'accesso agli elementi avviene sempre tramite l'operatore di indexing []

```
dante_comedy_books_1 = 'inferno', 'purgatorio', 'paradiso'  
dante_comedy_books_1 = ('inferno', 'purgatorio', 'paradiso')
```

Un altro tipo «sequenza»: le tuple

- Ha senso utilizzare le tuple al posto delle liste quando si è certi che il contenuto della collezione di dati non cambierà durante l'esecuzione del programma
- Ex: creare una tupla contenente i libri della Divina Commedia;
- Ex: creare una tupla contenente i vostri libri preferiti, più la tupla definita sopra;
- Ex: accedere agli elementi della tupla interna alla tupla dei vostri libri preferiti;

Un altro tipo «sequenza»: le tuple

```
dante_comedy_books_1 = ('inferno', 'purgatorio', 'paradiso')  
best_italian_books = (dante_comedy_books_1, 'decameron',  
                      'canzoniere')
```

```
print('my favourite italian work is Dante\'s', best_italian_books[0][0])
```

#tuple concatenation

```
t1 = (1, 3, 5, 7)
```

```
t2 = (10, 20, 30, 40)
```

```
t3 = t1 + t2
```

I dizionari

- Un dizionario in Python è una struttura dati formata da coppie {key:value}, assimilabile ad una rubrica telefonica

#keys must be immutable objects

#pairs are not sorted

```
authors_books = {'dante': ['vita nova', 'comedy', 'de monarchia'],  
                 'galileo': ['dialogue', 'sidereus nuncius'],  
                 'homer': ['iliad', 'odyssey']}
```

```
print('Homer\'s works are: ')
```

```
print('\t', end = '')
```

```
for work in authors_books['homer']:
```

```
    print(work, end = ' ')
```

```
print()
```

```
print('Galileo\'s works are: ')
```

```
print('\t', end = '')
```

```
for work in authors_books['galileo']:
```

```
    print(work, end = ' ')
```

```
print()
```