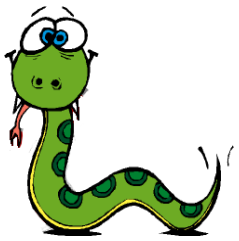
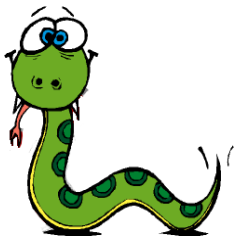


Il linguaggio Python

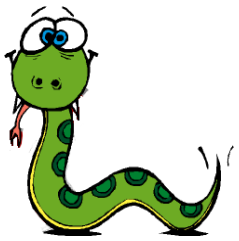
Capitolo 2
Variabili, Espressioni e Comandi

Prof. Mauro Gaspari: gaspari@cs.unibo.it



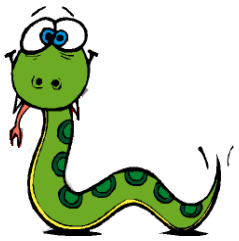
Il Linguaggio Python

- Python e' un linguaggio imperativo con alcune caratteristiche funzionali.
- Python e' basato su un interprete che e' disponibile su tutti i sistemi operativi ed anche sui palmari.
- Python e' uno “scripting language” linguaggio adatto a realizzare in modo molto veloce programmi, minimizzando i dettagli “notazionali” e quindi anche adatto alla prototipazione rapida.



L'interprete python

- Funzionamento interattivo.
  - interprete
  - esempio di comandi
- Invocazione di un programma scritto su file
  - Il file termina con .py
  - editare il file name.py
  - eseguire “python name.py”

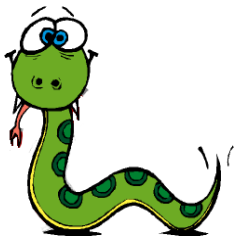


Un primo programma

- In genere il primo programma che si presenta quando si introduce un nuovo linguaggio e' chiamato “Hello, World!” perche' e' il programma che stampa questa frase.
- in python:

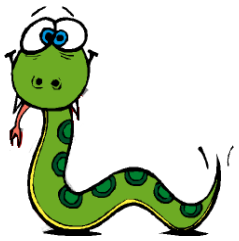
```
print “Hello, World!”
```

Valori e Tipi

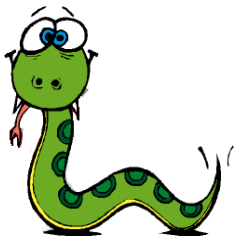


- Un ***valore*** (= ***value***) è una delle entità fondamentali che i programmi sono in grado di manipolare (ad esempio un valore può essere un numero o una lettera).
- Fino ad ora abbiamo visto 3 valori:
  - 1 il numero 1
  - 2 (il risultato di 1 + 1).
  - “Hello, World!”
- Ad ogni valore è associato un ***tipo*** (= ***type***) ad esempio:
  - 2 è un intero (tipo **int** in python)
  - “Hello, World!” è una “stringa” (tipo **string** in python)

Tipo **string**

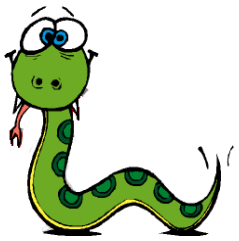


- **string** indica una sequenza di caratteri in inglese).
- Sia noi che l'interprete python possiamo riconoscere le stringhe perchè sono scritte tra virgolette.
- Interprete:
  - NB1. Il comando **print** funziona per gli interi e per gli altri tipi.
  - NB2. L'interprete e' in grado di stabilire il tipo di un valore con il comando **type: type**(VALORE)**.**
  - Tipo **float** per numeri con il punto decimale (perche' sono in virgola mobile - floating-point in inglese).



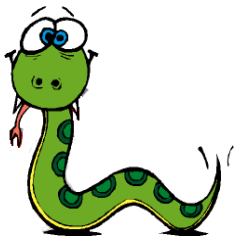
Stringhe e Numeri

- Interprete:
  - Numeri tra virgolette come “17”, “3.5” sono stringhe!
  - Numeri con la virgola 1,000,000 non sono proprio numeri!
    - NB. primo esempio di errore semantico: il codice viene eseguito senza errori ma l'argomento viene preso come una sequenza (lista) di numeri.



Variabili

- Una ***variabile*** (= ***variable***) e' un ***nome*** a cui viene associato (che riferisce a) un certo valore.
- NB. Le variabili sono comunemente utilizzate in tutti i linguaggi di programmazione:
  - A volte la loro definizione “tecnica” si scosta un po' da quella appena data per python.
  - Pero' il concetto intuitivo di legame tra nome e (possibile) valore e' in genere comune a tutti i linguaggi di programmazione ad alto livello.



Assegnamento

- Il comando di ***assegnamento*** (= ***assignment***) crea nuove variabili e associa ad esse un valore.
- Esempio nell'interprete:

```
>>> message = "What's up, Doc?"
>>> n = 17
>>> pi = 3.14159
```

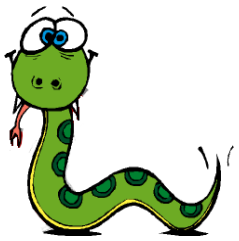
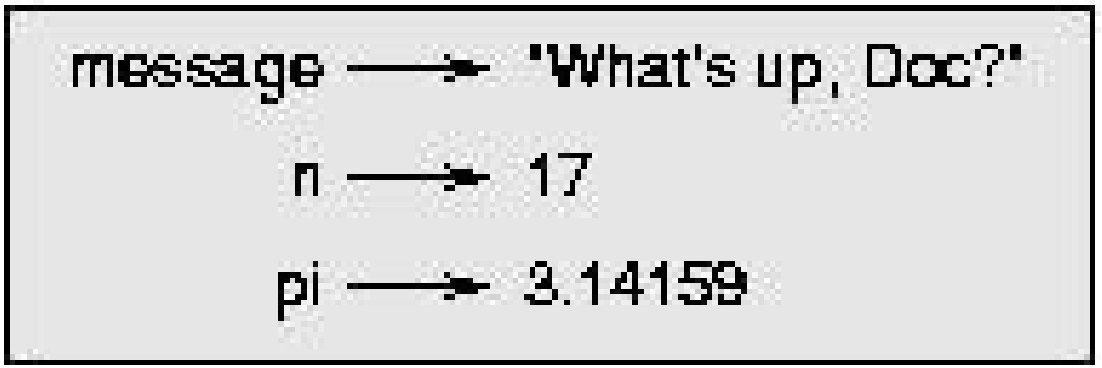
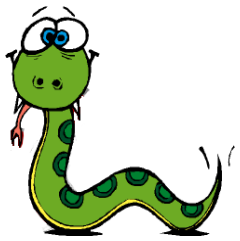


Diagramma di stato

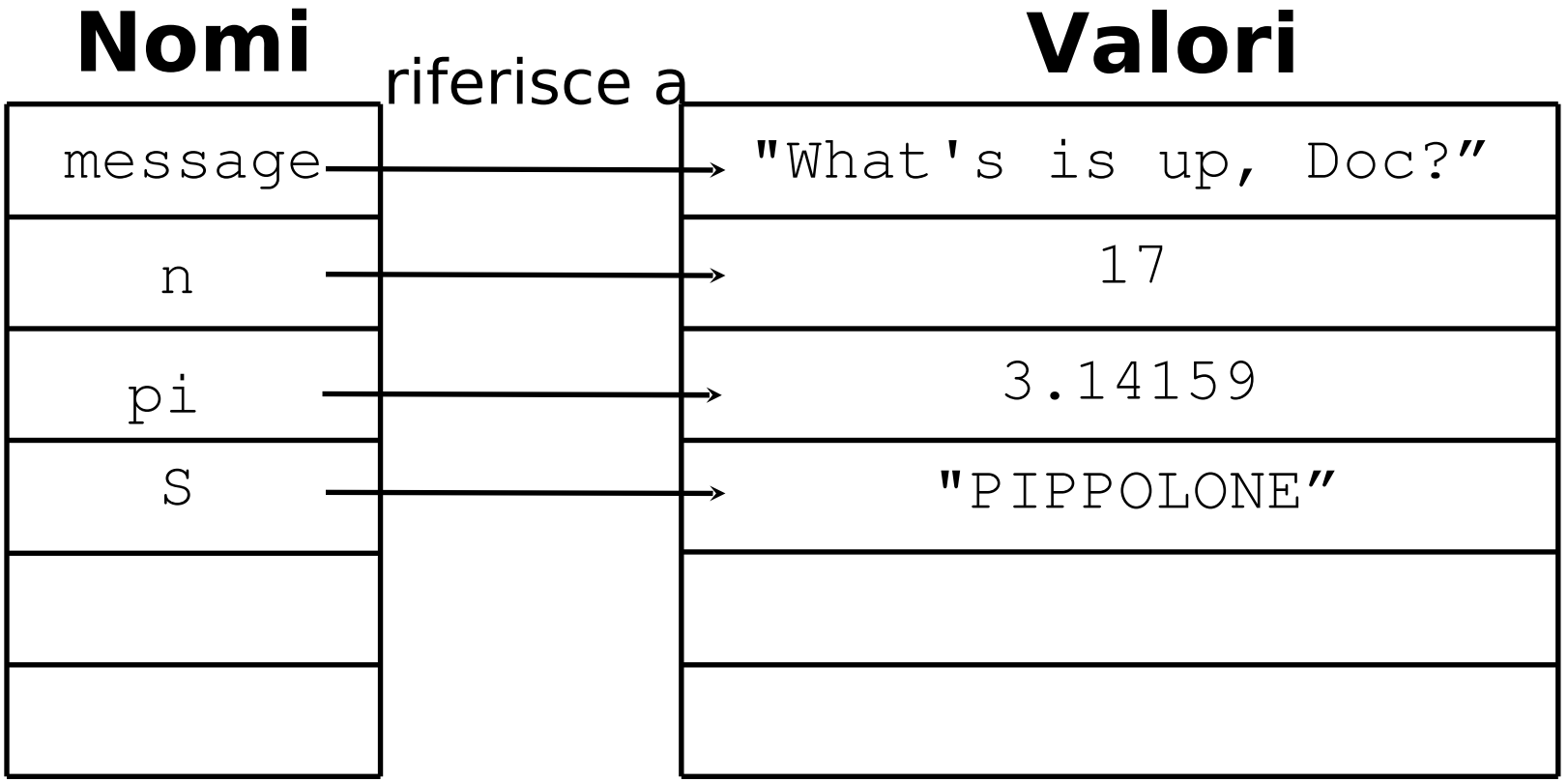
- Si può usare un diagramma di per indicare i valori associati alle singole variabili in un certo momento (***diagramma di stato***).
- Spesso utile in fase di debugging.

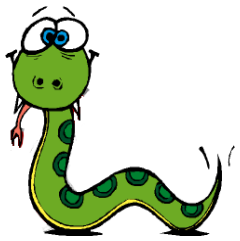




Ancora assegnamento

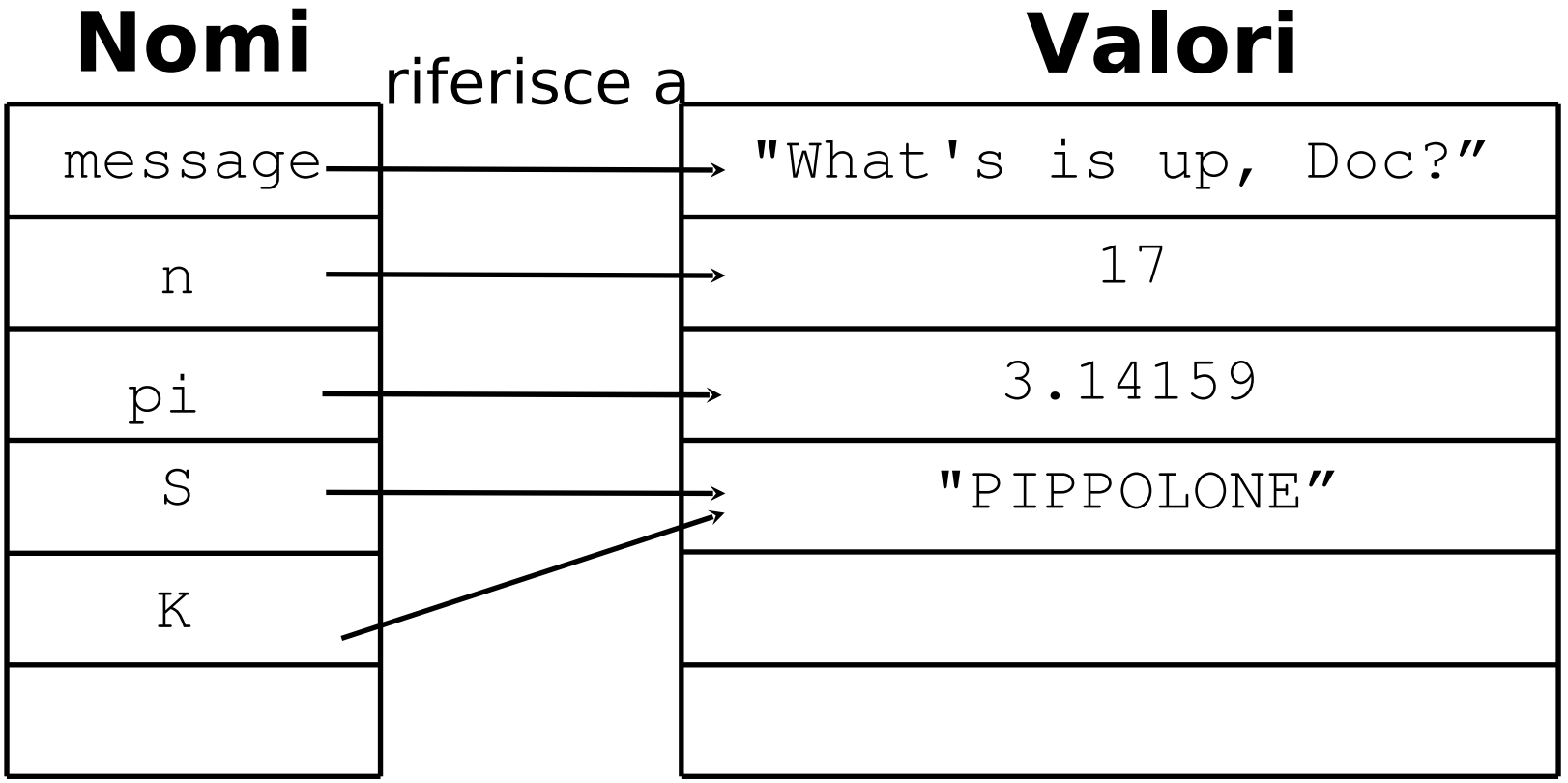
```
>>> s = "PIPPOLONE"
>>>
```

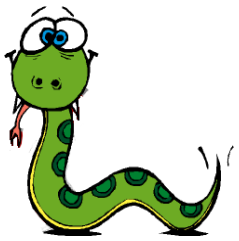




Ancora Assegnamento

```
>>> K = S
>>>
```

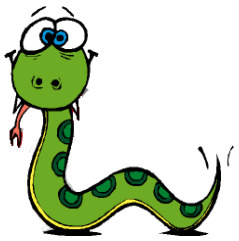




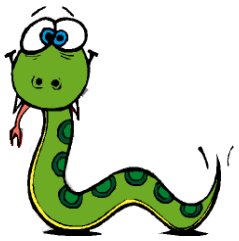
Tipo delle variabili

- Anche le variabili hanno associato un tipo.
- Il ***tipo di una variabile*** e' il tipo del valore a cui questa riferisce.
- Interprete
  - Il comando **print** funziona anche sulle variabili!
  - Il comando **type** funziona anche sulle veriabili!

Nomi di variabili



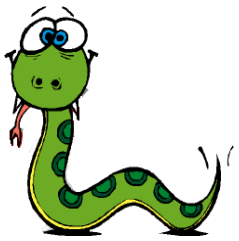
- Un buon programmatore in genere sceglie i nomi delle variabili per ricordare per cosa sono utilizzate queste variabili.
- I nomi delle variabili possono essere di lunghezza arbitraria (in genere questo vale per quasi tutti i linguaggi ad alto livello moderni).
- I nomi delle variabili devono iniziare con una lettera e possono contenere sia lettere che numeri.
- Il carattere *underscore* **(_)** puo' apparire nel nome di una variabile**,** ad esempio il_mio_nome **e' un nome legale di variabile (**utile per nomi con piu' parole).
- Nomi scoretti generano errori sintattici.



Esempi di nomi scorretti

```
>>> 76trombones = "big parade"
SyntaxError: invalid syntax
>>> more$ = 1000000
SyntaxError: invalid syntax
>>> class = "Computer Science 101"
SyntaxError: invalid syntax
```

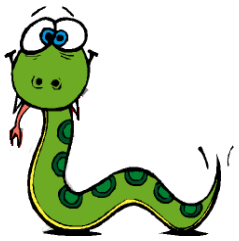
- La prima inizia con un numero.
- La seconda ha il carattere illegale $.
- **Quale e' il problema dell'ultima?**



Parole chiave

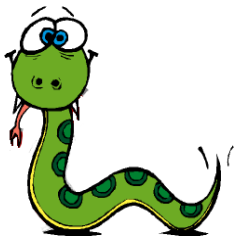
- Il problema e' che class e' una ***parola chiave*** (= ***keyword***) di python.
- Python ha 28 parole chiave che non possono essere utilizzate come nomi di variabili.
- Le parole chiave sono utilizzate per le istruzioni del linguaggio.

`and`	`continue`	`else`	`for`	`import`	`not`	`raise`
`assert`	`def`	`except`	`from`	`in`	`or`	`return`
`break`	`del`	`exec`	`global`	`is`	`pass`	`try`
`class`	`elif`	`finally`	`if`	`lambda`	`print`	`while`



Comandi

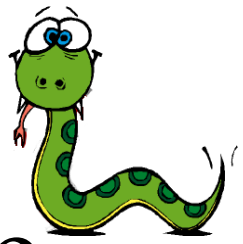
- Un ***comando*** (= ***statement***) e' un istruzione che puo' essere eseguita dall'interprete python.
- Quando si scrive un comando sulla riga di comando dell'interprete, python lo esegue e stampa il risultato.
- NB. ci sono comandi come l'assegnamento che non producono risultati.
- Uno script di solito contiene una sequenza di comandi.
- Interpete: esempi di comandi e script.



Valutazione di espressioni

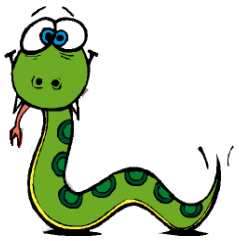
- Un ***espressione*** (= ***expression)*** e' una combinazione di valori, variabili e operatori.
- L'interprete **valuta** le espressioni e se puo' restituisce un valore.
- Esempi con l'interprete
- NB1. le espressioni più semplici sono costituite solo da valori.

```
>>> 17
17
>>> x
2
```



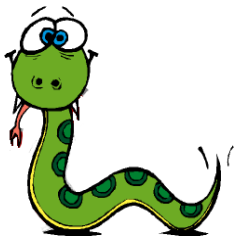
Differenza tra stampa e valutazione

- NB. Anche se in alcuni casi l'interprete sembra restituire le stesse cose i concetti di stampa e valutazione sono molto diversi tra loro.
- Quando l'interprete restituisce un valore lo restituisce cosi come è. Quindi questo può essere riutilizzato ad esempio assegnato ad una variabile.
- La stampa invece scrive sul video (vedremo anche da altre parti per esempio su un file) il valore dell'espressione. NB. Questa espressione non puo' essere riutilizzata dall'interprete, ad esempio assegnata ad una variabile.



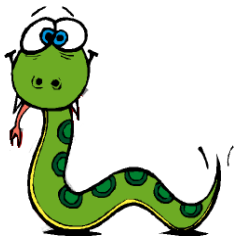
Esempio interprete

```
>>> message = "What's up, Doc?"
>>> message
"What's up, Doc?"
>>> print message
What's up, Doc?
```



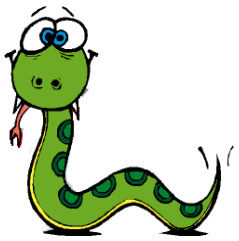
Assegnamento e Espressioni

- Le espressioni possono essere utilizzate nella parte destra di un assegnamento.
- NB. la parte sinistra di un assegnamento deve essere sempre un nome di variabile.



Esempi

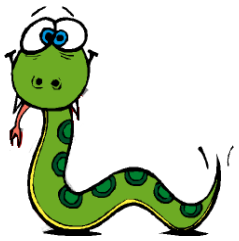
```
>>> print 1 + 1
2
>>> t = 1 + 1
>>> t
2
>>> t = print 1 + 1
  File "<stdin>", line 1
    t = print 1 + 1
              ^
SyntaxError: invalid
syntax
>>> print 1 + 1
2
```



Espressioni negli script

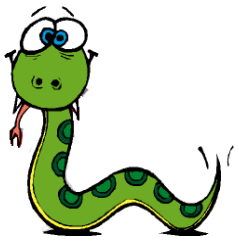
- Un espressione e' un comando legale in uno script ma non producono risultati.
- Ad esempio questo script non restituisce risultati.
- Come fare per stampare qualcosa?

```
17
3.2
"Hello, World!"
1 + 1
```



Operatori e Operandi

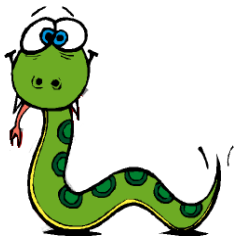
- Le espressioni in python si costruiscono con operatori.
- Gli ***operatori*** (= ***operators***) sono simboli speciali che rappresentano operazioni (ad esempio funzioni matematiche).
- I valori utilizzati dagli operatori si chiamano ***operandi***.
- Operatori matematici:
  - +, -, /, * (moltiplicazione), ** (esponente).



Espressioni con operatori

```
20+32          hour-1     hour*60+minute
minute/60      5**2       (5+9)*(15-7)
```

- Se un espressione con un operatore contiene una variabile anche questa viene valutata e quindi sostituita con il suo valore.
- Esempi interprete.

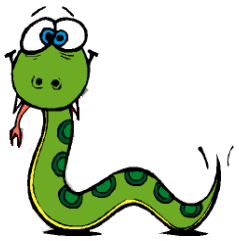


Esempio espressioni matematiche

```
>>> minute = 59
>>> minute/60
0
```

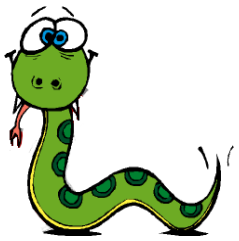
- Come mai?
  - nell'aritmetica convenzionale 59/60=0.98333
  - ma python sta eseguendo la divisione intera perche' i due operandi sono interi.
  - di solito la divisione intera arrotonda con il troncamento: 0.98333 --> 0~~.98333~~

Precedenza degli operatori



- La ***precedenza degli operatori*** stabilisce delle regole precise riguardo la valutazione delle espressioni con più operatori.
- Regole generali (PEMDAS):
  - P: le parentesi hanno la precedenza maggiore.
  - E: esponenziale. 2**1+1 --> (2**1)+1 da risultato 3
  - MD: moltiplicazione e divisione con stessa precedenza.
  - AS: addizione e sottrazione con stessa precedenza.
    - 2*3-1 --> (2*3)-1 da risultato 5
    - 2/3-1 --> (2/3)-1 da risultato -1 (NB 2/3 = 0 e' la div intera)

Operatori su stringhe

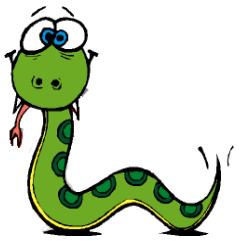


- In genere non è possibile effettuare operazioni matematiche su stringhe anche se le stringhe contengono numeri.
- Le seguenti operazioni sono illegali:
  - `message-1  "Hello"/123  message*"Hello"  "15"+2`
- Notare che in realta' l'operatore **+** funziona sulle stringhe: infatti realizza l'operazione di **concatenazione** tra stringhe:

```
>>>“pippo”+”pluto”
“pippopluto”
```

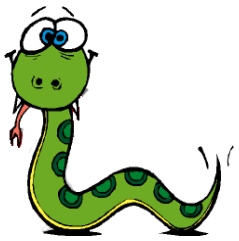
- Anche l'operatore * funziona sulle stringhe e realizza l'operazione di **ripetizione**: ha come argomenti una stringa e un intero:

```
>>>“pippo”*3
“pippopippopippo”
```



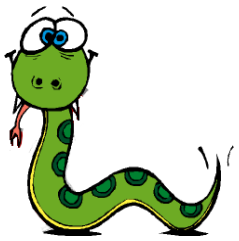
Esempi operatori su stringhe

```
>>>fruit = "banana"
>>>bakedGood = " nut bread"
>>>print fruit + bakedGood
banana nut bread
```



Commenti

- Quando i programmi diventano piu' lunghi e strutturati a volte diventa difficile, guardando un pezzo di codice, capire cosa questo fa'.
- Utile in fase di debugging o di aggiornamento del codice.
- E' una prassi comune aggiungere al codice commenti (in genere tutti i linguaggi di programmazione ad alto livello lo permettono).
- In python i commenti iniziano con il simbolo #.
- Tutto quello che segue il simbolo # viene ignorato fino alla fine della riga dove si trova.

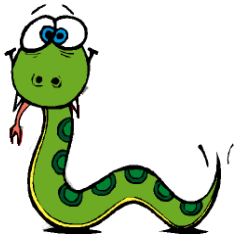


Esempio commenti

- Tipologie di commenti:
  - documentare il significato delle operazioni;
  - segnalare possibili fonti di errori;
  - segnalare modifiche al codice.

# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60

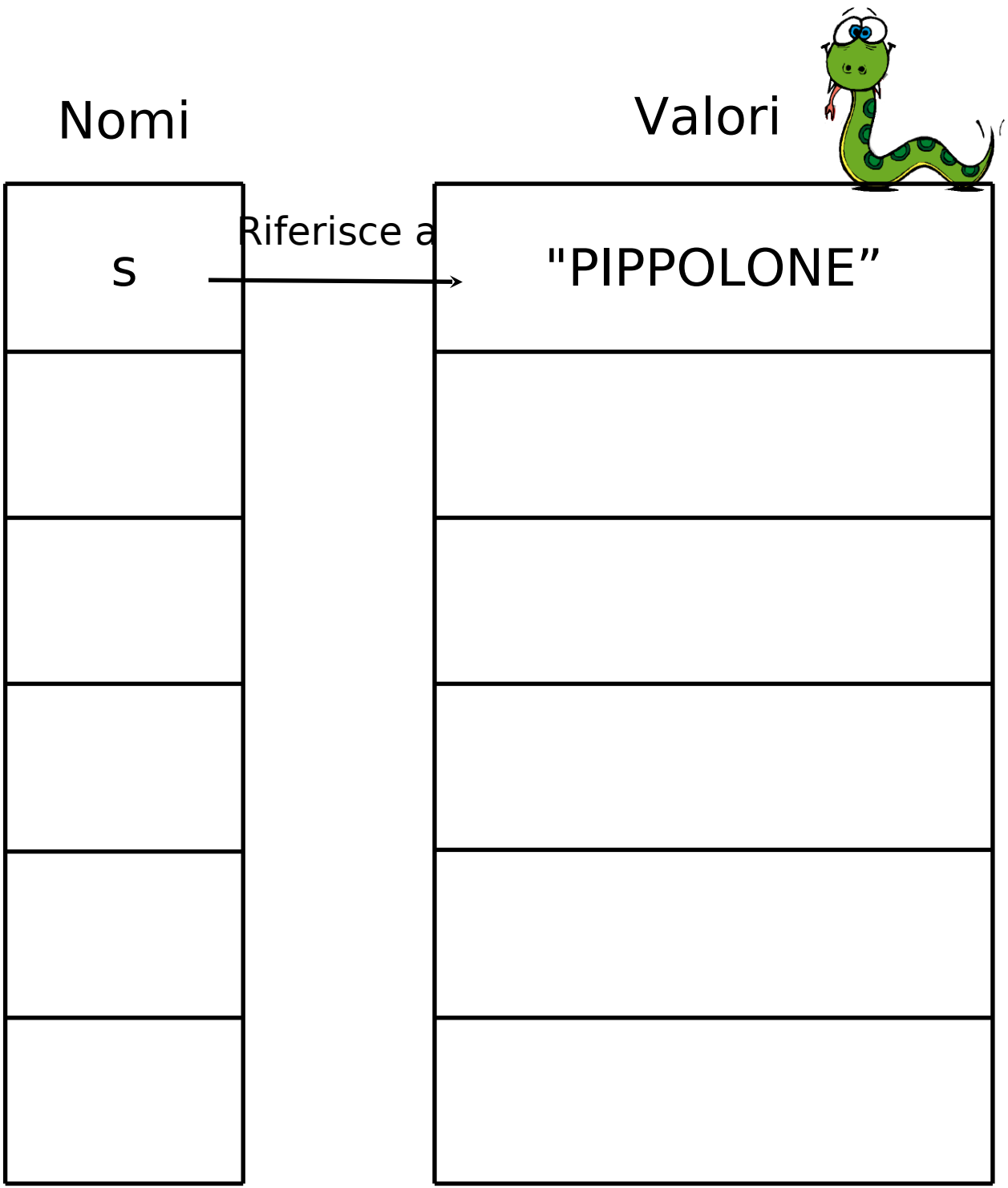
```
percentage = (minute * 100) / 60   # att. divisione int
```



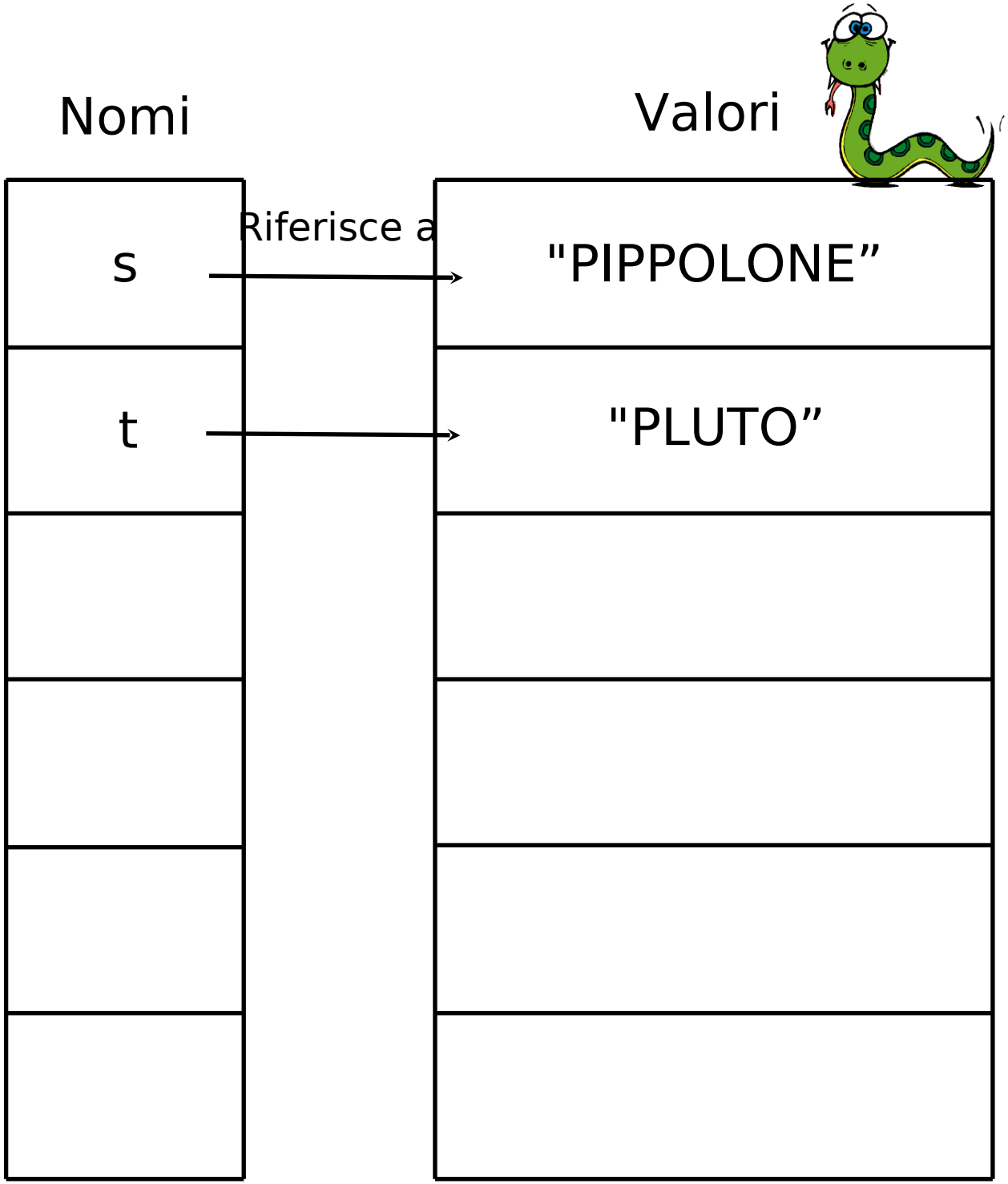
Assegnamento e riferimenti

Cosa accade in realtà,
un esempio ....

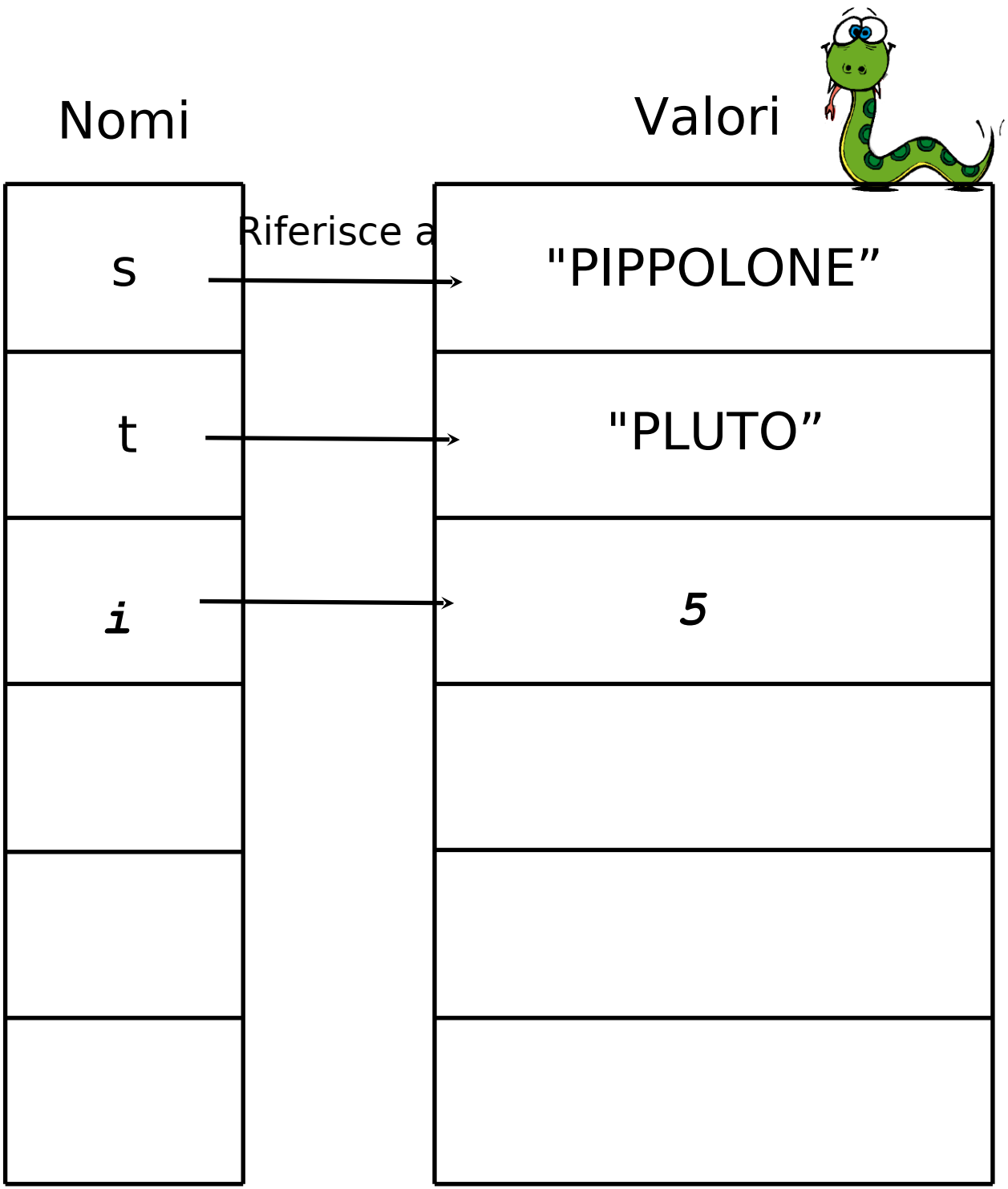
```
>>> s = "PIPPOLONE"
>>>
```



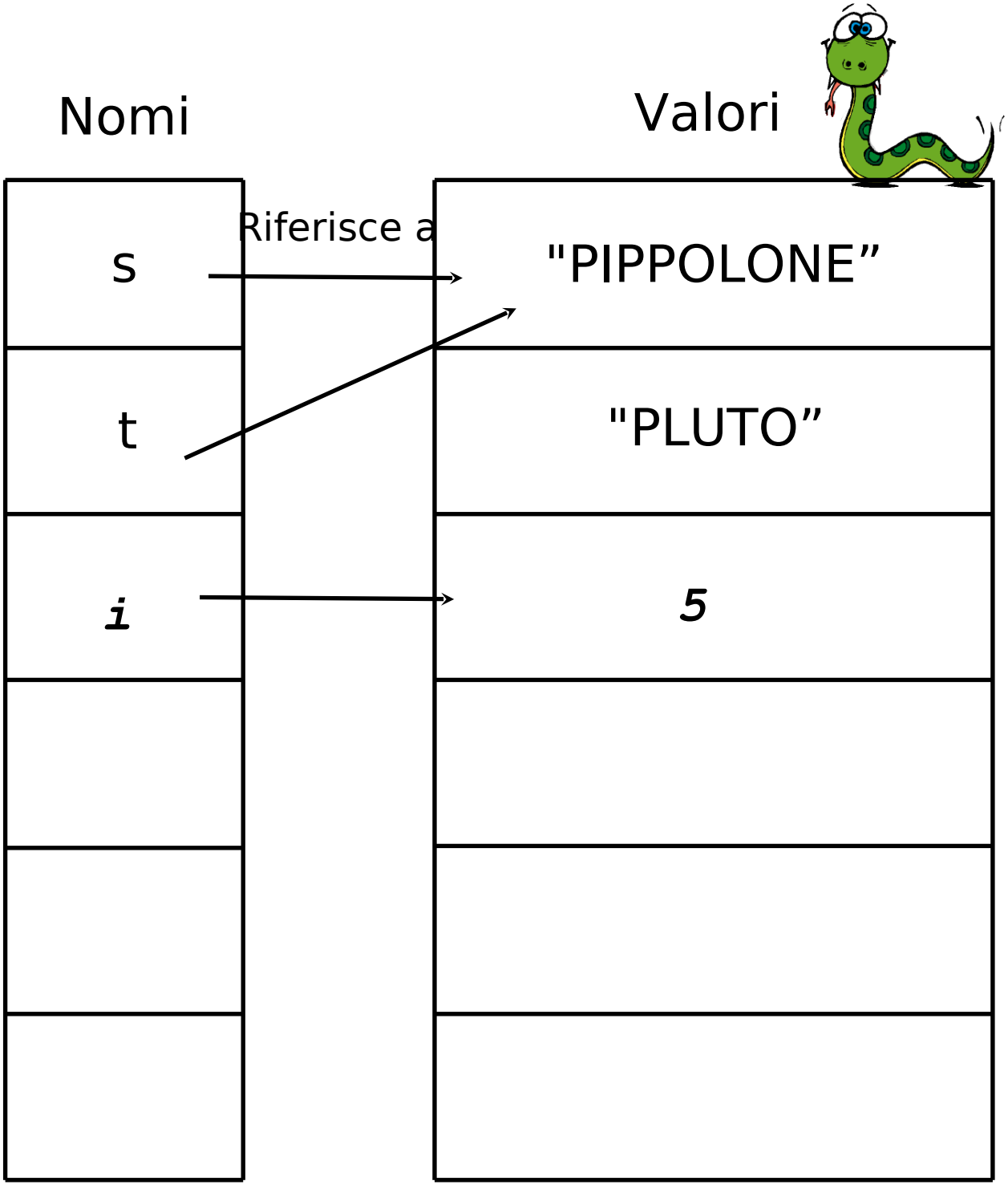
```
>>> s = "PIPPOLONE"
>>> t = "pluto"
>>>
```



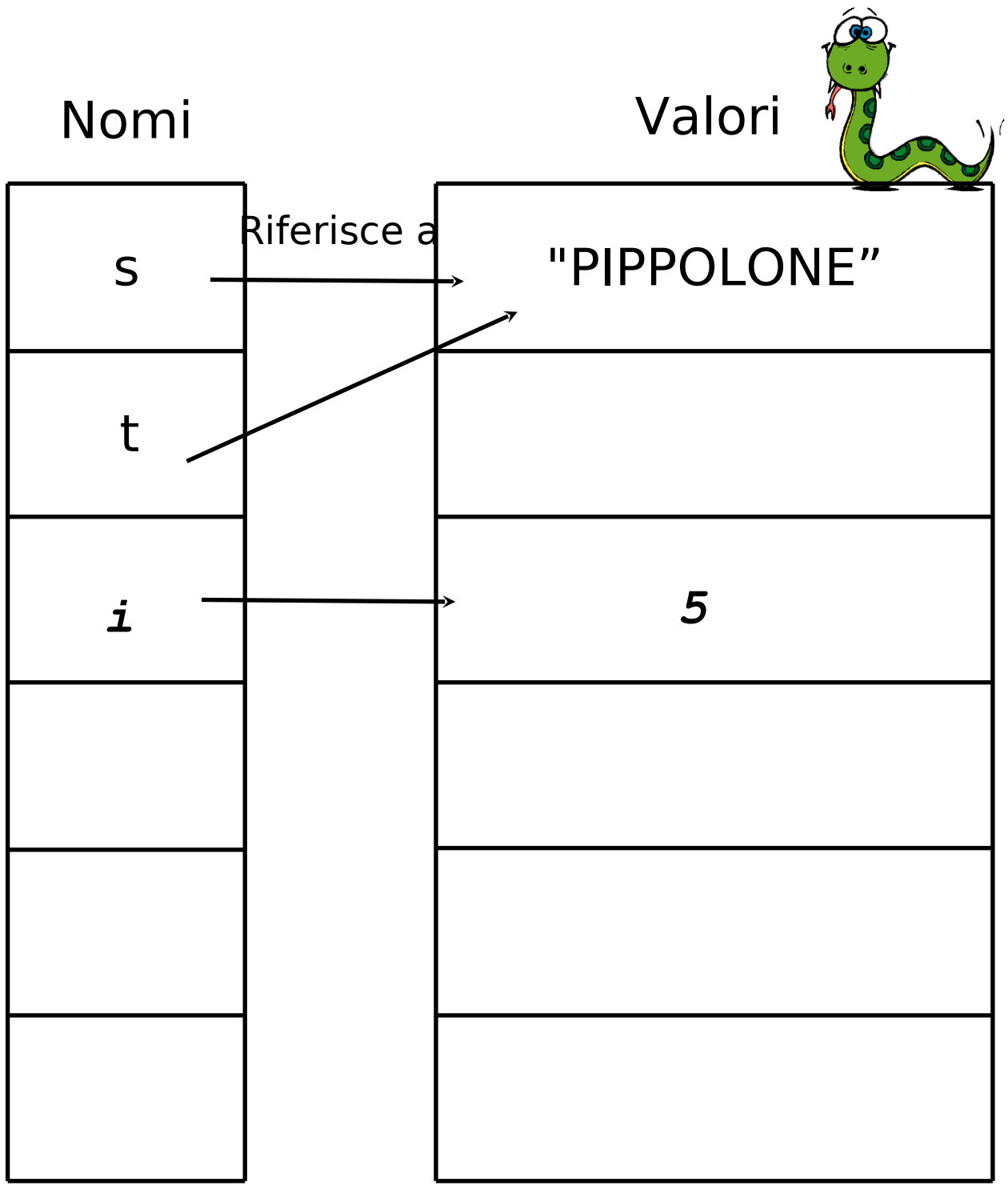
```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>>
```

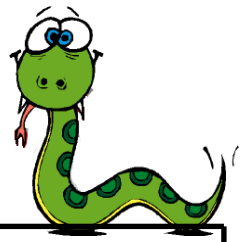


```
>>> s = "PIPPOLONE"
>>> t = "PLUTO"
>>> i = 5
>>> print i
5
>>> t = s
```

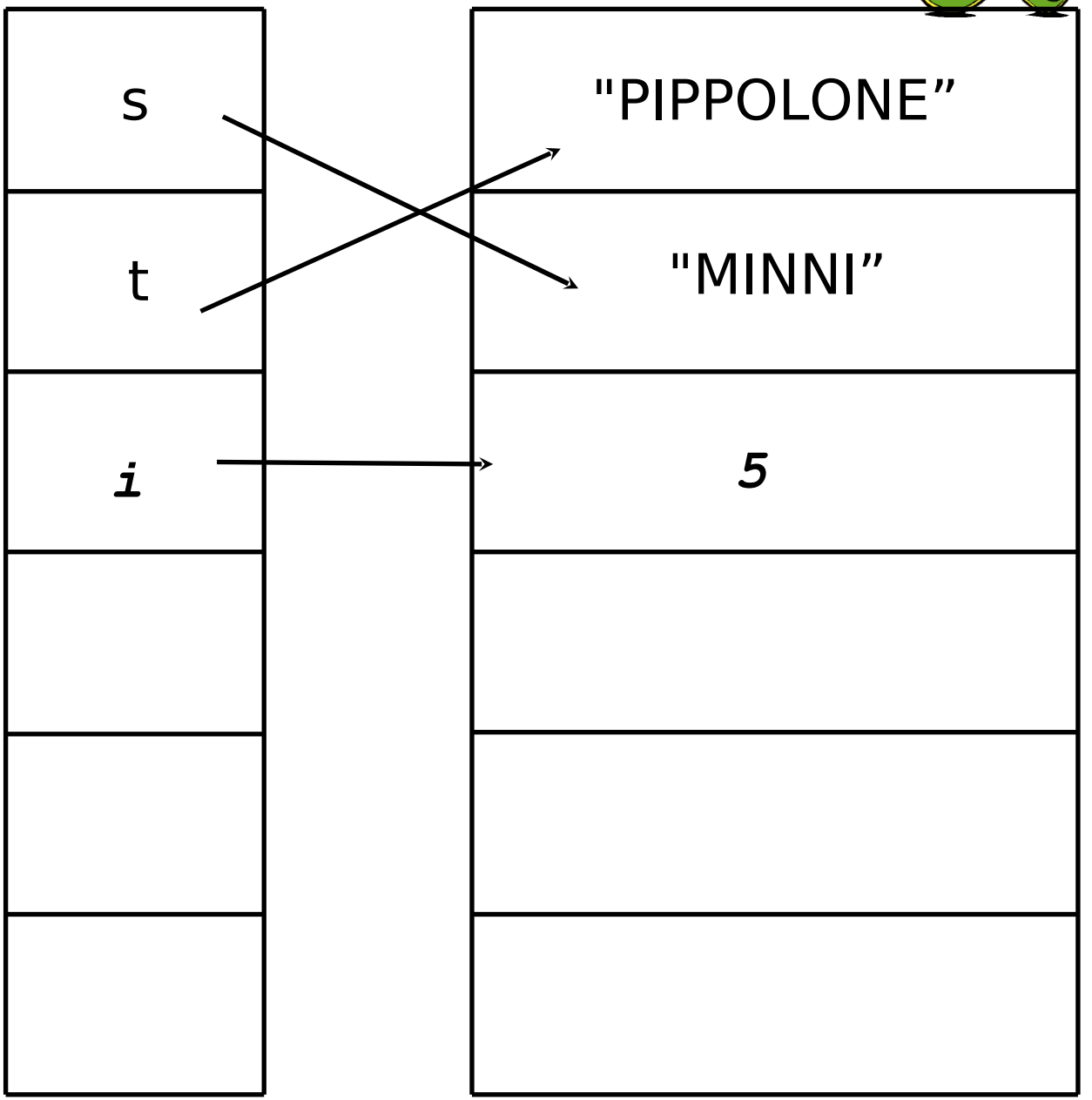


```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
>>> print a
Traceback (most.....
  File "<stdin>", line.
NameError: name 'a' is
not defined
>>>
```





```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
>>> print a
Traceback (most.....
  File "<stdin>", ....
NameError: name 'a' is
not defined
>>> S = “MINNI”
>>> print t
PIPPOLONE
```



```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
>>> print a
Traceback (most.....
  File "<stdin>", ....
NameError: name 'a' is
not defined
>>> S = “MINNI”
>>> print t
PIPPOLONE
>>> del t
>>> print t
Traceback (most.....
  File "<stdin>", ....
NameError: name 't' is
not defined
```