Funzioni

UNIVERSITÀ DI PISA

Informatica per le biotecnologie

Throwback: tabella dei simboli

Università di Pisa

Struttura che indica i simboli disponibili in memoria.

Nome	Dimensione	Indirizzo	Tipo	Valore
`genoma_1`	`2GB`	`0x345321`	`str`	`ACGG...`
`genoma_2`	`2.2GB`	`0x798991`	`str`	`TTCG...`
`genoma_3`	`1.9GB`	`0x110982`	`str`	`CCGT...`
`genoma_4`	`2.8GB`	`0x110984`	`str`	`CCCA...`

Il mondo in silico di Python

Università di Pisa

```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7
deviations = [number - average for number in numbers]
standard_deviation = sum(deviations) / 7

print(f"La media e' {average}, la standard deviation e' {standard_deviation}")
```

Costruiamo la tabella dei simboli, istruzione per istruzione.

```
numbers = [10, 2, 3, 4, 0, 8, 12]
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`numbers`	`...`	`...`	`list`	`[10, 2, ...]`

```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`numbers`	`...`	`...`	`list`	`[10, 2, ...]`
`average`	`...`	`...`	`float`	`5.57...`

```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7
deviations = [number - average for number in numbers]
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`numbers`	`...`	`...`	`list`	`[10, 2, ...]`
`average`	`...`	`...`	`float`	`5.57...`
`deviations`	`...`	`...`	`list`	`[4.43..., -3.57...]`

Funzioni

Una funzione definisce un comportamento o una funzionalità, e implementa una funzione da uno o più input (possibilmente vuoto), a un output, e.g.,

Input	Funzione	Output
`[1, 2, 3]`	massimo	`3`
`"ATCCGTATTC"`	conta timine	`4`
`["ATC...", "CTG..."], "ATC..."`	genoma più simile a quello dato	`"ATC..."`

Funzione: codice (parametrico) cui assegniamo un nome.

Le funzioni e l'invocazione

Le funzioni sono il componente principale che ci permette di riciclare il codice che scriviamo: una volta **definita**, una funzione può essere *invocata*, i.e., calcolata, quante volte vogliamo!

- Definisco una volta una funzione di conteggio basi azotate, la posso riutilizzare su ogni genoma
- Definisco una volta una funzione di calcolo similarità genomi, la posso riutilizzare su ogni coppia di genomi

Fun fact: posso invocare anche funzioni che non ho definito io.

Le funzioni e l'invocazione

UNIVERSITÀ DI PISA

Input	Funzione	Output	Invocazione
`[1, 2, 3]`	massimo	`3`	`massimo([1, 2, 3])`
`"ATCCGTATTC"`	conta timine	`4`	`conta_timine("ATCCGTATTC")`
`["ATC...", "CTG..."], "ATC..."`	genoma piu' simile a quello dato	`"ATC..."`	`trova_piu_simile( ["ATC...", "CTG..." ] , "ATC...")`
`"Hello, world!"`	stampa a schermo	`None`	`print("Hello, world")`

L'invocazione risulta in un valore! Posso comporre invocazioni con altri valori, e.g. `5 + media([1, 2, 3])`.

Il mondo in silico di Python: funzioni

Università di Pisa

```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7
deviations = [number - average for number in numbers]
standard_deviation = sum(deviations) / 7

print(f"La media e' {average}, la standard deviation e' {standard_deviation}")
```

Trova le funzioni.

Funzioni e parametri

Una funzione definisce:

- un insieme (potenzialmente vuoto) di *parametri*, che definiscono l'input alla funzione, e
- un *valore di ritorno*, che definisce l'output della funzione, e a cui la funzione valuta quando invocata. Ricorda: anche `None` è un valore!

Parametri	Funzione	Valore di ritorno
`[1, 2, 3]`	massimo	`3`
`"ATCCGTATTC"`	conta timine	`4`
`["ATC...", "CTG..."], "ATC..."`	genoma piu' simile a quello dato	`"ATC..."`
`"Hello, world!"`	stampa a schermo	`None`

Il mondo in silico di Python: funzioni

Università di Pisa

```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7
deviations = [number - average for number in numbers]
standard_deviation = sum(deviations) / 7

print(f"La media e' {average}, la standard deviation e' {standard_deviation}")
```

Trova i parametri.

Funzioni e parametri

Università di Pisa

La funzione stessa è del codice Python! I parametri sono variabili e il valore di ritorno è un valore.

Parametri	Funzione	Invocazione	Valore di ritorno
`numbers = [1, 2, 3]`	massimo	`massimo([1, 2, 3])`	`3`
`string = "ATCCGTATTC"`	conta timine	`conta_timine("ATCCGTATTC")`	`4`
`database = ["ATC...", "CTG..."], genoma = "ATC..."`	genoma piu' simile a quello dato	`trova_piu_simile( [ "ATC...", "CTG..." ] , "ATC...")`	`"ATC..."`
`"Hello, world!"`	stampa a schermo	`print("Hello, world")`	`None`

Definire funzioni

Università di Pisa

```
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg
```

- `def` introduce la funzione
- `( )` delimita i parametri
- Il codice (*corpo*) della funzione e' indentato
- `return` termina la funzione e restituisce l'espressione data

Quando invocata, Python **assegna i parametri**, e esegue il corpo.

Funzioni e tabelle dei simboli

Università di Pisa

```python
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`

Funzioni e tabelle dei simboli

Università di Pisa

```python
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`
`elements`	`...`	`...`	`list`	`[1, 2, 3]`

Funzioni e tabelle dei simboli

Università di Pisa

```python
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`
`elements`	`...`	`...`	`list`	`[1, 2, 3]`
`total`	`...`	`...`	`int`	`6`

Funzioni e tabelle dei simboli

Università di Pisa

```python
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`
`elements`	`...`	`...`	`list`	`[1, 2, 3]`
`total`	`...`	`...`	`int`	`6`
`avg`	`...`	`...`	`float`	`2.0`

Funzioni e tabelle dei simboli usa e getta


```
def average(elements: list) -> float:
  total = sum(elements)
  avg = total / len(elements)

  return avg

average([1, 2, 3])
...
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`

Una volta terminata la funzione (finite le istruzioni nel corpo, o dopo un `return`), tutti i simboli introdotti vengono eliminati, e si svuota la tabella dei simboli.

Funzioni e tabelle dei simboli

Università di Pisa

```python
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

total = 7
average([1, 2, 3])
```

All'inizio dell'invocazione di `average` abbiamo la tabella seguente.

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`
`total`	`...`	`X`	`int`	`7`
`elements`	`...`	`...`	`list`	`[1, 2, 3]`

Funzioni e tabelle dei simboli

Università di Pisa

```python
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

total = 7
average([1, 2, 3])
```

Nome	Dimensione	Indirizzo	Tipo	Valore
`average`	`...`	`...`	`function`	`...`
`total`	`...`	`X`	`int`	`7`
`elements`	`...`	`...`	`list`	`[1, 2, 3]`
`total`	`...`	`X`	`int`	`6`

Due `total` nella stessa tabella? A quale facciamo riferimento?

La pila (stack)

Quando invocata, ogni funzione ha una propria tabella dei simboli, che si aggiunge a quelle precedenti, formando una pila.
Quando termina l'invocazione, la tabella viene eliminata, liberando la pila.

Nella pila, si fa riferimento all'ultima occorrenza di un simbolo, i.e., **si leggono le tabelle dei simboli dall'alto verso il basso nella pila**!

Stack
Funzione n
...
Funzione 2 Invoca Funzione 3
Funzione 1 Invoca Funzione 2
Funzione 0 Invoca Funzione 1

La pila (stack)

Università di Pisa

```
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
...
```

Stack

Funzione 0

Invoca `average`

La pila (stack)

Università di Pisa

```
def average(elements: list) -> float:
  total = sum(elements)
  avg = total / len(elements)

  return avg

average([1, 2, 3])
...
```

Stack
sum
average Invoca `sum`
Funzione 0 Invoca `average`

La pila (stack)

Università di Pisa

```
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
...
```

Stack
len
average Invoca `len`
Funzione 0 Invoca `average`

La pila (stack)

Università di Pisa

```
def average(elements: list) -> float:
    total = sum(elements)
    avg = total / len(elements)

    return avg

average([1, 2, 3])
...
```

Stack

Funzione 0

Invoca `average`

Conflitto di simboli e parametri

Se non troviamo un simbolo nella tabella corrente, scendiamo alla tabella successiva. Questo ci permette di utilizzare simboli senza doverli passare come parametri, e di riciclare nomi senza creare conflitti!

```
def average(elements: list) -> float:
  total = sum(elements)
  avg = total / len(elements)

  return avg

total = 0
average([1, 2, 3])
...
```

Stack
average Dichiara un altro `total`, diverso da quello di Funzione 0
Funzione 0 Invoca `average`

Conflitto di simboli e parametri

Se non troviamo un simbolo nella tabella corrente, scendiamo alla tabella successiva. **Se agiamo su un simbolo definito in tabelle precedenti... lo modifichiamo!**

```
def replace(elements: list, value: int, position: int) -> None:
  elements[position] = value

elements = [1, 2, 3]
replace(elements, 0, 0)
...
```

Stack
replace Non trova `elements` nella propria tabella, cerca in basso, e considera quello di `Funzione 0` !
Funzione 0 Invoca `average` , contiene `elements`

Conflitto di simboli e parametri

Se non troviamo un simbolo nella tabella corrente, scendiamo alla tabella successiva. **Se agiamo su un simbolo definito in tabelle precedenti... lo modifichiamo!**

```
def replace(elements: list, value: int, position: int) -> None:
  elements[position] = value

elements = [1, 2, 3]
replace(elements, 0, 0)
# elements qui e' [0, 2, 3]
...
```

Side effect

Un *side effect* è una modifica dettata da una funzione che agisce al di fuori della propria tabella dei simboli. Generalmente da evitare, specialmente in programmi complessi, poiché rende il codice più complesso e meno prevedibile.