Introduzione a Python e al data model

UNIVERSITÀ DI PISA

Linguaggi di programmazione

UNIVERSITÀ DI PISA

Un linguaggio di programmazione ci permette di esprimere un nostro algoritmo in un linguaggio comprensibile anche da una macchina, e agisce da linguaggio intermedio tra il linguaggio umano e quello macchina.

Noi tratteremo Python, versione `3.11`.

Installazione Python in `Tools, guida all'installazione`

Perché Python?

- Sintassi relativamente semplice e flessibile
- Uno tra i linguaggi *de facto* più utilizzati, specialmente in biologia e affini
- Fornisce tanti programmi e funzioni di bioinformatica, genetica, analisi dati, etc.
- Un linguaggio che permette sia di scrivere programmi... che di interagirci!

Class, meet Python

Un programma Python definisce una **lista di istruzioni** che seguono una specifica e rigorosa *sintassi* e *semantica*.

- Sintassi: regole di scrittura
- Semantica: regole di funzionamento

Come in ogni linguaggio, abbiamo delle componenti fondanti su cui costruiamo l'equivalente di frasi, paragrafi, etc.

Anatomy of a (small) Python program

Università di Pisa

```python
from Bio import SeqIO
from Bio.Blast.NCBIWWW import qblast

# legge il dataset salvato in data/flora.gbk
flora = list(SeqIO.parse("data/flora.gbk", "genbank"))
# filtra: seleziona solo dati relativi alle orchidee
orchids = filter(lambda flower: "Orchyd" in flower.name, flora)
# estrae le sequenze genetiche
sequences = map(lambda orchid: orchid.dna_sequencing)

# ricerca sequenze simili
similar_sequences = [Blast.qblast("blastn", "nt", sequence) for sequence in sequences]
```

Carica un piccolo dataset di flora, filtra le orchidee, e calcola la similarita' tra il loro genoma e una banca dati.

Il mondo in silico di Python

Università di Pisa

Python, come altri linguaggi, ci fornisce un insieme minimale di oggetti:

- **Valori**: oggetti componibili in espressioni che valutano a un valore (`3`, `4.1`, "ACGTTTAC...", `True`, `False`, ecc.)
- **Tipi**: definiscono i possibili valori, e le operazioni cui possono essere sottoposti, e.g., numero intero, numero con la virgola, stringa, ecc. Ogni valore ha un tipo!
- **Operatori**: combinano valori, creandone di nuovi, e.g., `+, -, *, /, in`
- **Costrutti**: permettono di modificare che istruzioni eseguire, e quando, e.g., non eseguire alcune istruzioni, o ripeterne altre
- **Funzioni**: raggruppano e parametrizzano istruzioni, e.g., `mean`, `integral`, `cosine`, `synthesize_protein` ecc.

Il mondo in silico di Python: tipi

Python offre un insieme di tipi "primitivi" o "built-in", i.e., tipi nativi al linguaggio e che **non** possiamo modificare:

- `int` numeri interi
- `float` numeri reali
- `str` stringhe: testo che inseriamo tra `"` o `'`
- `bool` valori di verità: vero o falso
- `NoneType` assenza di valore

Il mondo in silico di Python: tipi

Università di Pisa

```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7  # sum(x) somma i valori in x
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 tipi `int`, i.e., numeri interi
- Trova tipi `float`, i.e., numeri reali
- Trova tipi `str`, i.e., testo

Il mondo in silico di Python: tipi collezione

- `set` Definisce un insieme eterogeneo (anche vuoto) di valori: `{1.3, "ACGT"}`
- `list` Definisce una lista di valori che può' variare in lunghezza: `[1.3, "ACGT", 1.3]`
- `tuple` Definisce una lista (a dimensione **fissa!**) di valori: `(1.3, "ACGT")`
- `dict` Dizionario: mappa dei valori ad altri valori: `{"A": "Adenosine", "T": "Thymine"}`

Il mondo in silico di Python: tipi collezione


```
numbers = [10, 2, 3, 4, 0, 8, 12]
average = sum(numbers) / 7  # sum(x) somma i valori in x
deviations = [number - average for number in numbers]
standard_deviation = sum(deviations) / 7
statistics = {
          "mean": average,
          "std": standard_deviation,
          "variance": standard_deviation ** 2
}
```

Trova le collezioni.

Il mondo in silico di Python: espressioni

Come nelle espressioni matematiche, i tipi possono essere composti in espressioni:

- `a + b` somma
- `a - b` sottrazione
- `a * b` moltiplicazione
- `a / b` divisione, e `a // b` divisione intera
- `a ** b` elevamento a potenza
- `!= e ==` disuguaglianza e uguaglianza
- `and`, `or`, `not` operatori booleani (di verità)

Possiamo usare solo le parentesi tonde per imporre ordine di valutazione!

Il mondo in silico di Python: espressioni

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 valori
- Trova le espressioni

Il mondo in silico di Python: funzioni

Raggruppano codice, in modo da poterlo riusare in futuro senza riscriverlo, e con parametri diversi. Sono *invocate* con il loro nome, e dei parametri tra parentesi, e.g., `trova_minimo([1, 3, -1, 4, 0])`.

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.

Da valori a variabili

Il valore è transiente, una volta usato, viene "consumato", e non lo possiamo più usare in futuro. La variabile ci permette di salvare valori in memoria. Salviamo valori in una variabile definendo un *nome*, un *tipo* (opzionale, ma consigliato), e assegnando un valore con `=`, che, come suggerisce il nome, potremo poi cambiare in futuro.

```
genome_with_type: str = "ACGTC"
genome_without_type = "ACGTC"
```

Memorizzare: variabili

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 variabili

Memorizzare: variabili

Università di Pisa

Le variabili vengono memorizzate nelle tabelle dei simboli

```
genome_with_type: str = "ACGTC"
```

Produce una tabella

Nome	Dimensione	Indirizzo	Tipo	Valore
`genome_with_type`	`...`	`...`	`str`	`ACGTC`

Memorizzare: variabili

Ogni volta che assegniamo nuovi valori, aggiorniamo la variabile, anche di tipi diversi.

```
genome_with_type = "ACGTC"
genome_with_type = "TTTTT"
genome_with_type = 3
```

Memorizzare: variabili

Le variabili vengono valutate, pertanto **dove possiamo usare un valore, possiamo usare una variabile!**

```
conversion_rate = 5


in_celsius = 23
in_fahrenheit = in_celsius * conversion_rate
```