Progetto “Interprete di espressioni”

Appello del 25 febbraio 2021

Descrizione del progetto

Lo scopo del progetto è quello di implementare i metodi necessari a valutare espressioni rappresentate come classi in Python. Per esempio, l'operazione si somma della variabile x con la costante 2 potrebbe venire rappresentata come:

```
espressione = Operazione(sum, [Variabile("x"), Costante(2)])
```

Questo ci fornirà un oggetto di tipo `Operazione` sul quale potremo chiamare il metodo `valuta` passando come argomento un dizionario Python (solitamente chiamato “ambiente”) avente come chiavi i nomi delle variabili e come valori associati i valori di ogni variabile. Per esempio, chiamando

```
espressione.valuta({"x": 4})
```

verrà valutata l'operazione di somma in cui alla variabile x è stato sostituito il valore 4. L'intera procedura di valutazione può essere descritta con tre differenti casi:

1. Se stiamo valutando una operazione dobbiamo prima valutare tutti i suoi argomenti e poi valutare l'operazione sul risultato della valutazione dei suoi argomenti (e.g., per valutare `sum` prima valutiamo tutti i suoi argomenti e poi li passiamo a `sum`).
2. Se stiamo valutando una costante il suo valore è già deciso (e.g., la costante 2 ha sempre valore 2).
3. Se stiamo valutando una variabile dobbiamo vedere quale è il suo valore nell'ambiente cercando il valore associato al nome della variabile (e.g., per la variabile x cerchiamo nell'ambiente il valore associato al nome x).

Si implementino tre classi `Operazione`, `Costante` e `Variabile`.

La classe `Operazione` dovrà contenere i seguenti metodi:

- `__init__(self, operazione, argomenti)`. Riceve come argomenti l'operazione da compiere (deve essere una funzione) e una lista di argomenti da passare all'operazione.
- `valuta(self, ambiente)`. Valuta l'operazione nell'ambiente (un dizionario Python) fornito come argomento.

La classe `Costante` dovrà contenere i seguenti metodi:

- `__init__(self, valore)`. Riceve come argomento il valore della costante.
- `valuta(self, ambiente)`. Valuta l'operazione nell'ambiente (un dizionario Python) fornito come argomento. Il risultato della valutazione sarà, ovviamente, indipendente dall'ambiente dato che è la valutazione di una costante.

La classe `Variabile` dovrà contenere i seguenti metodi:

- `__init__(self, nome)`. Riceve come argomento il *nome* della variabile (una stringa).
- `valuta(self, ambiente)`. Valuta l'operazione nell'ambiente (un dizionario Python) fornito come argomento.

Codice

Viene fornito uno scheletro del codice che deve essere implementato, con i metodi che devono essere scritti:

```
class Operazione:

    def __init__(self, operazione, argomenti):
        ...

    def valuta(self, ambiente):
        ...


class Costante:

    def __init__(self, valore):
        ...

    def valuta(self, ambiente):
        ...


class Variabile:

    def __init__(self, nome):
        ...

    def valuta(self, ambiente):
        ...
```

Nel progetto è consentito avere funzioni e classi aggiuntive per l'implementazione.

Si ricorda di commentare adeguatamente il codice, approfittandone per spiegare le scelte implementative effettuate.

Esempi d'uso

Il seguente frammento di codice chiama tutti i metodi la cui implementazione è richiesta dal progetto:

```
def test():
    espressione = Operazione(sum,
                             [Operazione(max, [Costante(2),
                                               Variabile("x")]),
                              Operazione(min, [Variabile("x"),
                                               Variabile("y")]),
                              Costante(3),
                              Variabile("y")])
    ambiente = {"x": 3, "y": -2}
    risultato = espressione.valuta(ambiente)
    print(risultato)
    ambiente = {"x": 1, "y": 3}
    risultato = espressione.valuta(ambiente)
    print(risultato)
```

Questo dovrebbe essere l'output prodotto da una implementazione funzionante:

```
2
9
```

Suggerimenti generali

1. Sebbene per risolvere lo stesso problema possano esistere altre strutture dati, per l'esame è da implementare la struttura dati descritta nel testo del progetto.
2. Rispettate i vincoli di complessità richiesti.
3. Fate più test possibile: assicuratevi che il codice implementato funzioni in tutti i casi richiesti.

Indicazioni

Il progetto deve essere svolto **individualmente**. La consegna dovrà avvenire entro le ore 23:59 del giorno 13/02/2021 secondo le seguenti modalità:

- Invio di una email a `lmanzoni@units.it` dal vostro account email istituzionale con oggetto *[informatica] consegna progetto appello del 25/02/2021*.
- L'email deve avere come allegato il progetto in un singolo file in codice sorgente Python versione 3, dal nome `Nome_Cognome_matricola.py`, quindi, per esempio Mario Rossi di matricola 12345 consegnerà un file dal nome `Mario_Rossi_12345.py`.
- Il file deve contenere sotto forma di commento le seguenti linee indicanti nome, cognome e numero di matricola:

  ```
  # Nome: Mario
  # Cognome: Rossi
  # Matricola: 12345
  ```