**Che cosa sono i nomi e l’ambiente (environment)?**

Un nome è una sequenza di caratteri utilizzata per rappresentare un’entità, che può essere un dato (es. cella di memoria) oppure un dettaglio relativo al controllo (es. comandi).

Sono oggetti denotabili gli oggetti a cui viene attribuito un nome e l’associazione tra oggetto e nome può avvenire durante la creazione del linguaggio o all’esecuzione di un programma.

L’ambiente (referencing environment, namespace in Python) è l’associazione tra nomi e oggetti denotabili che esiste a tempo di esecuzione in un punto specifico del programma ed ad uno specifico momento dell’esecuzione.

**Che cosa sono le regole di visibilità?**

Le regole di visibilità specificano come e quando una dichiarazione può essere visibile: una dichiarazione locale ad un blocco è visibile in quel blocco ed in tutti quelli annidati, a meno che non vi sia una ridefinizione, che nasconde la dichiarazione precedente. Lo scope può essere statico, ossia basato sul testo del programma, oppure dinamico, dipendente quindi dall’esecuzione.

**Si spieghi la gestione della memoria dinamica tramite stack e record di attivazione.**

Nella gestione della memoria dinamica tramite stack i blocchi sono aperti e chiusi utilizzando uno schema LIFO, quindi quando si entra ad esempio in un blocco A e poi in un blocco B, prima di lasciare A si lascia B. Per gestire la memoria in questo modo si utilizza una pila (stack). Lo spazio di memoria allocato sullo stack e dedicato ad un blocco è chiamato record di attivazione o frame ed è associato ad una specifica attivazione della procedura e non alla dichiarazione della stessa.

**Si illustrino le operazioni che sono eseguite quando viene eseguita una procedura.**

Quando viene eseguita una procedura (chiamata) viene innanzitutto modificato il valore del contatore del programma, necessario per passare il controllo alla procedura chiamata, poi viene allocato spazio per il nuovo record e quindi il puntatore alla prima locazione di memoria libera deve essere aggiornato di conseguenza. Successivamente il puntatore punta al nuovo record di attivazione e vengono passati i parametri. Vengono poi salvati i valori per la gestione del controllo ed è eseguito il codice per l'inizializzazione.C

Per far ritornare il controllo al chiamante viene aggiornato il valore del contatore del programma, viene restituito il valore di ritorno e ripristinati i valori di registri precedentemente salvati. Infine viene eseguito il codice per la finalizzazione e viene deallocato lo spazio sullo stack.

**Si spieghino le possibili strategie per la gestione della memoria dinamica tramite heap.**

I metodi per la gestione dell'heap si dividono a seconda del fatto che i blocchi vengano considerati di dimensioni fisse o variabili.

Dimensioni fisse: l'heap è diviso in blocchi di dimensioni fisse collegati in una struttura a lista chiamata lista libera e, quando un'operazione richiede l'allocazione di un blocco di memoria nell'heap, il primo elemento della lista libera è rimosso dalla lista ed il puntatore alla lista libera è aggiornato per puntare all'elemento successivo, mentre quello all'elemento rimosso dalla lista viene restituito. Quando la memoria viene liberata, il blocco liberato è nuovamente collegato in testa alla lista libera.

Dimensioni variabili: unica lista libera o liste libere multiple. Nel primo caso quando viene chiesta l'allocazione di un blocco di n parole di memoria, le prime n parole vengono allocate ed il puntatore all'inizio dell'heap viene incrementato di n parole, mentre i blocchi deallocati vengono collegati in una lista libera. Quando termina lo spazio dedicato all'heap bisogna riutilizzare la memoria deallocata. Nel caso delle liste libere multiple, ne vengono usate con dimensioni diverse e quando viene richiesto viene utilizzato il blocco dalla lista che contiene blocchi di dimensione uguale o maggiore a n.

**Si illustrino la differenza tra linguaggi imperativi e dichiarativi.**

Nei linguaggi imperativi sia l'ambiente che lo stato possono variare, mentre in quelli dichiarativi solo l’ambiente può farlo. I primi sono ispirati alla struttura fisica del computer e molti dei linguaggi di programmazione solitamente utilizzati lo sono. I secondi (dichiarativi) hanno lo scopo di astrarre dalla macchina fisica e non possiedono comandi per modificare lo stato ( = la memoria); i programmi non sono altro che un insieme di dichiarazioni di funzioni.

I linguaggi dichiarativi possono essere funzionali o logici: nei primi si valutano le funzioni utilizzando regole matematiche, mentre in quelli logici la computazione è basata sulla deduzione della logica del prim’ordine.

**Si parli del goto.**

Il comando goto, ispirato alle istruzioni jump dei linguaggi assembly, trasferisce il controllo al punto del programma in cui è definita la label A. Non è essenziale in un linguaggio, anzi è stato dimostrato che può essere sempre sostituito con altre istruzioni. In generale, il goto viene poco utilizzato in quanto rende la lettura del codice più difficile e a tratti incomprensibile (spaghetti code). Per questo motivo, nei linguaggi più moderni di alto livello questa istruzione sta sparendo (es. Java è il primo ad averlo completamente rimosso)

**Si parli dei comandi condizionali.**

I comandi condizionali o di selezione esprimono un’alternativa tra due possibili continuazioni della computazione a seconda del valore di un’espressione logica. I gruppi di comandi condizionali sono due, if e case.

Utilizzo if: if Boolexpr then C1 else C2 dove Boolexpr è un’espressione booleana, mentre C1 e C2 sono comandi

Utilizzo case: specializzazione del comando if, ma con più rami.

Case Exp of

Label1: C1;

Label2: C2;

…

Labeln: Cn;

Else Cn+1

Dove Exp è l’espressione il cui valore è di un tipo compatibile con quello delle etichette label, mentre C1, …, Cn+1 sono comandi

**Si parli dei comandi iterativi.**

I comandi iterativi permettono la ripetizione di un gruppo di istruzioni e sono fondamentali per ottenere un programma Turing-complete. Nei linguaggi ad alto livello esistono due meccanismi fondamentali, quello di iterazione strutturata e quello di ricorsione.

L’iterazione può essere determinata ed indeterminata: nel primo caso, la ripetizione avviene attraverso un costrutto che permette l’iterazione un numero prefissato di volte, mentre nel secondo la ripetizione avviene finché la condizione non diventa vera.

Nella ricorsione (comune nei linguaggi dichiarativi), la funzione richiama se stessa e si ripete un numero arbitrario di volte.

La forma più comune di comando iterativo indeterminato è quella del while, mentre per quelli determinati si utilizza il for.

L’iterazione indeterminata da solo rende un programma Turing-complete, mentre quella determinata no.

**Che cos’è la programmazione strutturata?**

La programmazione strutturata nasce negli anni ‘70 e consiste in una serie di prescrizioni con l’obiettivo di permettere lo sviluppo di software strutturato sia nel codice che nel flusso di controllo. Alcune di queste sono la progettazione top-down (il programma è sviluppato attraverso raffinamenti successivi in cui vengono aggiunti nuovi dettagli), la modularizzazione del codice (raggruppamento dei comandi che rappresentano determinate funzioni nell’algoritmo), l’utilizzo di nomi significativi, di commenti, di tipi di dato strutturati e di costrutti strutturati per il controllo.

**Che cos’è la tail recursion?**

La tail recursion è una ricorsione in cui non devono essere svolti altri calcoli nelle chiamate intermedie al ritorno della ricorsione. Serve a risparmiare spazio di memoria usato per memorizzare i valori intermedi di ogni chiamata: questo è possibile perchè i valori parziali sono memorizzati in uno spazio accumulatore che ad ogni chiamata ricorsiva viene modificato, nello stesso spazio.

**Che differenza c’è tra ricorsione e iterazione?**

Ricorsione e iterazione raggiungono la stessa potenza espressiva, ma per l’elaborazione di dati utilizzando strutture rigide è più semplice usare costrutti iterativi, mentre per strutture di natura simbolica è più naturale l’utilizzo della ricorsione. Detto questo, la ricorsione viene considerata meno efficiente rispetto all’iterazione, quindi i linguaggi dichiarativi sono ritenuti meno efficienti rispetto a quelli imperativi.

**Quando un linguaggio ha first-class functions?**

Un linguaggio ha first-class functions quando supporta il passaggio di funzioni come argomenti ad altre funzioni, restituirle come valori da altre funzioni e salvarle a variabili o strutture. Sono necessarie per la programmazione funzionale.

**Che cosa fa il garbage collector e perché è utile?**

Il garbage collector reclama automaticamente la memoria allocata nell'heap e non più utilizzata: è utile perchè permette di liberare in automatico memoria riservata inutilizzata.

**Si illustrino le due tecniche principali per la garbage collection.**

Reference counting: quando viene creato un oggetto nell'head, allo stesso tempo viene allocato un contatore di riferimento inaccessibile al programmatore. Questo contatore contiene il numero di puntatori ancora attivi verso l'oggetto e quando avviene un assegnamento tra puntatori p = q, il contatore dell'oggetto puntato da q viene incrementato di 1 mentre quello di p decrementato di 1. Quando si esce dall'ambiente locale, i contatori di tutti gli oggetti puntati da un puntatore locale a quell'ambiente vengono decrementati di 1. Quando il contatore raggiunge 0, l'oggetto può essere deallocato. Il reference counting è abbastanza inefficiente.

Mark and sweep: nella fase "mark" vengono analizzati tutti gli oggetti nell'heap e vengono marcati come "non in uso", poi si attraversano ricorsivamente tutte le strutture presenti nell'heap e ogni oggetto incontrato viene marcato come "in uso". Nella fase "sweep" tutti i blocchi marcati come "non in uso" vengono restituti alla lista libera. Questo collector viene invocato quando la memoria dell'heap è qausi terminata.

**Come sono implementati gli oggetti?**

Un'operazione è eseguita inviando un messaggio all'oggetto che consiste nel nome del metodo da eseguire insieme ad eventuali parametri. Invocare un metodo può coinvolgere la selezione dinamica del metodo da eseguire ed i dati dell'oggetto possono essere più o meno accessibili dall'esterno, da alcuni oggetti oppure completamente private e visibili solo all'interno dell'oggetto stesso. Molti oggetti in un programma si differenziano spesso e volentieri solo per il valore dei dati, non per la struttura nè per i metodi. Il principio di organizzazione più noto per gli oggetti è quello delle classi.

**Come si svolge il dynamic method lookup (selezione dinamica dei metodi)?**

Il dynamic method lookup si svolge ad un'invocazione o.m(parametri), che è una funzione del tipo dell'oggetto che riceve il messaggio, non del tipo del riferimento.

**Come sono rappresentate le classi?**

Le classi sono rappresentate utilizzando una lista collegata in cui ogni elemento rappresenta una classe e contiene l'implementazione dei metodi che sono definiti o ridefiniti nella classe.

**Come si invocano i metodi nel caso di ereditarietà singola?**

I metodi nel caso dell'ereditarietà singola si invocano con un riferimento che appartiene staticamente ad una superclasse.

**Che cos’è la programmazione basata su eventi in JavaFX?**

JavaFX permette la realizzazione di applicazione con GUI (Graphical User Interface) ed attraverso gli eventi è possibile avviare determinate azioni attraverso un click del mouse, la digitazione di un input, ecc.

**Che cosa sono lo stage, la scena, il root node e i nodes in JavaFX?**

Lo stage è il container top-level che contiene la scena, mentre la scena è il contenitore per tutti i contenuti JavaFX e racchiude il root node, ossia il nodo che contiene tutti gli altri nodi. Questi ultimi non sono altro che oggetti visionabili o addirittura interagibili.

**Che cosa sono le lambda expressions in Java e come si usano in JavaFX?**

Le lambda expressions sono dei blocchi che prendono parametri e restituiscono un valore: in JavaFX rappresentano in modo chiaro e conciso un metodo.