Università di Trento

Laboratory of Computer Science Education

UNITÀ DI APPRENDIMENTO

“Strutture di controllo con libreria Turtle”

Fatto da:

Simone Degiacomi

Carlo Fanciulli

Rupert Gobber

Francesco Penasa

UDA

UNITÀ' DI APPRENDIMENTO	
Denominazione	Strutture di controllo con libreria Turtle
Compito significativo e prodotti	Lezioni didattiche dedicate all'insegnamento delle strutture di controllo con un approccio costruttivista, supportato da esercizi di error analysis, pair programming ed esercizi con la libreria grafica Turtle
Obiettivi Formativi	● Utilizzo cosciente delle condizioni booleane. ● Padronanza del costrutto If e delle sue varianti (ad esempio: Elif in serie). ● Conoscenza delle strutture di ciclo For e While. ● Utilizzo di cicli per descrivere l'esecuzione di azioni parametriche. ● Capacità di comprensione dell'evoluzione dell'esecuzione del programma. ● Sperimentazione di cambiamenti sul codice.
Utenti destinatari	Studenti della scuola secondaria di secondo grado
Prerequisiti	Conoscenza base della programmazione, tra cui: ● Parole chiave del linguaggio, operatori e separatori (parentesi,virgola). ● Variabili e operazioni matematiche (per le condizioni). ● Esecuzione di programmi in Python. ● Input/print da terminale e le istruzioni più comuni.
Struttura	L'unità didattica è divisa in 3 sottosezioni: ● sottosezione If + Turtle; ● sottosezione While + Turtle; ● sottosezione For + Turtle;
Tempi	12 ore (4 ore per ogni sottosezione) ● 3 settimane per gli istituti tecnici informatici. ● 6 settimane per i licei (in quanto le ore settimanali sono inferiori).
Metodologie	● Pair Programming ● Reinforcement ● Error Analysis
Strumenti fisici	● Laboratorio di informatica ● Computer ● Internet o cartella condivisa (per accedere agli esempi)
Strumenti didattici	● Flowchart ● Python ● Python Turtle
Valutazione	Non considerata

Indice

Metodologie e Strumenti Utilizzati 2

Python 4

Pair Programming 4

Reinforcement 5

Error Analysis 6

Libreria Turtle 6

Istruzione Condizionale: If 7

Introduzione 7

Funzionamento dell'If 7

Logica booleana 7

Error Analysis 8

Possibili difficoltà generali degli studenti 9

Esempi Turtle per l'istruzione condizionale If 10

Ciclo: While 11

Premessa: perché introduciamo il While prima del For? 11

A cosa ci serve un ciclo? 11

Flowchart: While e Do-while 11

Ciclo: For 14

Esercizi di conversione 14

For in Python 15

Note 16**5**

Bibliografia 17**6**

Metodologie e Strumenti Utilizzati

Abbiamo diviso l'unità didattica in tre sottosezioni, ognuna dedicata a una struttura di controllo diversa, e all'interno di esse abbiamo cercato di mantenere la stessa modalità di insegnamento. Questa modalità comprende una fase introduttiva basata sui Flowchart, seguita dal passaggio al codice Python e completata da esempi grafici con Turtle. All'interno di ogni sezione vengono proposti degli esercizi di Error Analysis, che abbiamo considerato come un'attività di scaffolding. Infine, suggeriamo di applicare il Pair programming nelle attività di laboratorio.

Le motivazioni che ci hanno portato a queste scelte vengono ora illustrate.

Python

Il primo fattore che abbiamo preso in considerazione per l'unità didattica è stato scegliere il linguaggio di programmazione da utilizzare.
Nel nostro caso la scelta è stata Python, un linguaggio user-friendly che viene incontro ai novizi essendo di alto livello e quindi più simile al linguaggio naturale.
Offrendo una sintassi più semplice rispetto ad altri linguaggi di programmazione di uso comune, gli studenti possono concentrarsi maggiormente sull'aspetto semantico del programma e produrre meno errori di sintassi (Mannila et. al., 2006).
Infatti, è molto facile imparare e capire la sintassi, è potente soprattutto in un contesto di formazione iniziale alla programmazione in cui i programmi sono di piccole dimensioni e fa sviluppare immediatamente l'uso di buone pratiche, come ad esempio l'indentazione. Infatti Python, a differenza di altri linguaggi, utilizza l'indentazione per gestire i blocchi di codice, ed è quindi obbligatoria per la scrittura di programmi.
A nostro parere ciò porta una serie di evidenti e significative soft skills come lo sviluppo di organizzazione del codice, di chiarezza e pulizia che ha dei vantaggi in un contesto di team working in cui è essenziale leggere e far leggere il codice prodotto.
Inoltre è evidente che sia diventato una dei linguaggi più utilizzati e importanti degli ultimi anni che vanta di un'immensa community con progetti, siti e tutorial disponibili in rete, che permettono anche ai più inesperti di approcciarsi immediatamente e in modo molto accessibile al linguaggio.

Pair Programming

Nel metodo dell'apprendimento costruttivista, ritenuto adatto per stimolare un'interazione creativa e relazionale tra gli studenti, la collaborazione, condivisione e cooperazione sono fondamentali per la costruzione della conoscenza, per la ricchezza degli scambi comunicativi e la creazione di legami sociali.

Inoltre, studenti diversi pongono la propria attenzione su aspetti diversi della programmazione, perdendone di vista altri. Similmente, quando si è bloccati su un problema computazionale, spiegare il problema ad un compagno aiuta a comprenderne la causa e porta ad una soluzione.

Per questi motivi riteniamo molto utile organizzare le attività di laboratorio in gruppi di due, massimo tre studenti. Tale modalità cooperativa risulta molto aperta e non prevede una separazione troppo formale del lavoro, questo per evitare confusione, frustrazione o conflitti all'interno del gruppo. Prevediamo una strategia collaborativa in cui uno studente del gruppo attivamente scrive il codice mentre l'altro, che funge da osservatore, supervisiona il lavoro svolto e ha più tempo per ragionare su di esso.

Infine, per evitare sia una formazione statica del gruppo sia la classica situazione in cui solo una persona svolge gran parte del lavoro, i ruoli dovranno essere ripetutamente invertiti nel corso delle sessioni di lavoro, scambiando anche i componenti tra i vari gruppi. Tale dinamismo e interazione tra i vari alunni permetterà di sviluppare aiuti reciproci, una più intensa capacità di concentrazione e focus verso l'obiettivo, ma soprattutto una elaborazione metacognitiva che permette di osservare diversi ruoli ed esperienze su se stesso e sui compagni favorendo sia la risoluzione dei problemi sia l'arricchimento personale.

Reinforcement

"La fiducia in se stessi non assicura il successo, ma la mancanza di fiducia origina sicuramente il fallimento" (Albert Bandura).

Per gestire correttamente la classe, il docente dovrebbe mirare alla conduzione dei rapporti interpersonali e di gruppo, volti a promuovere negli alunni un atteggiamento positivo verso la proposta educativa ed in generale verso l'apprendimento.
Skinner, noto psicologo statunitense di Harvard, rappresentante del comportamentismo ha identificato cinque ostacoli all'apprendimento: timore di fallire, compito troppo lungo e complicato, troppo generico e senza istruzioni e direzioni, istruzioni poco chiare o confuse, ambiente non adatto, poco o nessun rinforzo positivo.
Il rinforzo è costituito da quegli eventi o stimoli che per la loro piacevolezza aumentano la probabilità che il comportamento rinforzato si riproduca. Quindi è ovvio che un rinforzo positivo spinge il soggetto a ripetere un certo comportamento, quello negativo ad evitarlo. (legge dell'effetto).

Elogiare è molto più efficace che rimproverare: in una classe è molto meglio osservare, apprezzare e rinforzare i comportamenti virtuosi piuttosto che criticare e rimproverare i comportamenti disfunzionali. Quindi è importante creare un ambiente scolastico in cui tutti gli studenti si sentano sicuri, a proprio agio e accettati.

La classe si configura come un fondamentale spazio di crescita in cui lo studente sperimenta le proprie competenze e sviluppa la propria identità, in un continuo scambio con i pari e gli insegnanti, nel quale i processi emotivi e relazionali assumono un ruolo centrale.

Error Analysis

Prima di mettere gli studenti al lavoro su esercizi di coding, proponiamo degli esercizi di Error Analysis, dove i ragazzi vedono del codice sbagliato e devono trovare gli errori. In questi esercizi proponiamo diversi tipi di errori (sia di semantica che di concetto). Nell'inserire questo tipo di esercizi, vogliamo familiarizzare i ragazzi con il codice, senza fornire una soluzione corretta ad un esercizio, ma aiutandoli a porre l'attenzione sui dettagli del linguaggio e dell'algoritmo.

Libreria Turtle

Per rafforzare i concetti e le nozioni apprese in classe, approfittiamo dell'immediatezza della libreria Turtle. Questa libreria offre numerose possibilità per la creazione di rappresentazioni grafiche del codice. Nell'uso della shell interattiva di Python lo studente, sotto la guida dell'insegnante, può sperimentare i metodi di Turtle, modificarne la sequenza di questi e osservare i risultati dell'esecuzione di ogni riga di codice. L'interattività di questa operazione è ideata con lo scopo di rafforzare il concetto di esecuzione sequenziale, fondamentale nell'apprendimento della programmazione. L'immediatezza di Turtle nella rappresentazione grafica fornisce agli studenti uno strumento di debugging immediato, il codice risulta facilmente ispezionabile e seguirne i passaggi tramite la rappresentazione grafica è semplice e immediato, nonché divertente.

La libreria Turtle viene introdotta con degli esempi estremamente semplici (in allegato a questa relazione), tali esempi sono necessari al docente per introdurre le funzioni base della libreria come lo spostamento dell'oggetto Turtle sul piano cartesiano. Possono essere mostrati anche degli esempi più complessi, come quelli presenti nell'IDE di Python. Questa sorta di vetrina di esempi viene usata con l'intento di stimolare nello studente l'interesse per la disciplina mostrando esempi interessanti dal punto di vista visuale. Inoltre, lo spazio di movimenti della Turtle possono essere interpretati come un piano cartesiano dando un aspetto più interdisciplinare alla materia.

Gli esempi di codice allegati alla relazione sono stati pensati per essere visti, eseguiti e modificati in classe in modo interattivo con gli studenti, durante la presentazione degli esempi vengono presentate le misconception più comuni e gli spunti di discussione che nascono. Questa interazione ha lo scopo di ridurre le possibili misconception degli studenti, andando ad analizzare le parti più iconiche dell'esecuzione.

Istruzione Condizionale: If

Introduzione

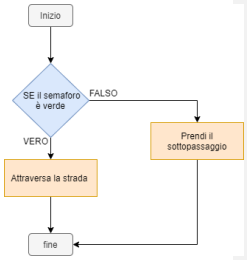
Come già sappiamo, i programmi sono un insieme di istruzioni che i computer eseguono per svolgere un compito e sono scritti in un certo linguaggio di programmazione.
La vera potenzialità della programmazione è l'avere alternative alla pura esecuzione sequenziale, poichè, valutando certe condizioni, possiamo decidere di far eseguire alcune istruzioni invece che altre, ripeterle o evitarle. Questo costrutto, usato universalmente, è chiamato **if-else** e consente di effettuare una selezione **a 2 vie.**

Commentato [1]: rivolto agli studenti

Dopo una breve introduzione segue una spiegazione riguardante l'uso dei diagrammi di flusso, che accompagneranno tutte le strutture di controllo, spiegando come una rappresentazione grafica di un nostro ragionamento permetta una facile comprensione del funzionamento del nostro percorso logico, evitando distrazione dovute alla sintassi del linguaggio di programmazione.

Funzionamento dell'If

Un modo per iniziare il processo di apprendimento potrebbe essere l'uso di esempi pratici, come ragionare la scrittura di algoritmi di ricette (Hromkovič, 2006) includendo controlli del flusso di esecuzione. Il processo dovrebbe portare gli studenti a scrivere prima programmi semplici e poi combinare insieme le soluzioni semplici per ottenere la soluzione a problemi più complicati.



Un possibile esempio è la considerazione di una scena di vita quotidiana, come l'attraversamento pedonale: ***se*** il semaforo è verde attraversi la strada, ***altrimenti*** prendi il sottopassaggio.

Logica booleana

Il passo successivo è introdurre i concetti principali alla base delle condizioni, come il valore booleano, l'espressione booleana e la costruzione di condizioni più complesse attraverso gli operatori logici.
La semantica dei linguaggi formali che si usano in matematica ed informatica non ha molto a che fare con quella del linguaggio naturale e per questo è importante dedicare molto tempo alla logica, utilizzando anche esempi iniziali intuitivi per far prendere familiarità.

È rilevante sottolineare che la condizione di un'istruzione If può essere formata da:

- Una singola variabile che esprime un valore booleano – `if a`
- Un'espressione condizionale semplice con operatori di confronto – `if a > b`
- Un'espressione complessa formata con operatori logici – `if a > b and c > a`
- Chiamata ad una funzione – `if quadrato(a) > 10`

Dopo aver dedicato del tempo alla logica, è necessario introdurre le condizioni in serie e gli Elif. Entrambi permettono di avere l'annidamento di più istruzioni If e scegliere tra più di due alternative. Anche in questo caso risulta costruttivo portare degli esempi esplicativi dei costrutti presi in considerazione facendo capire la differenza nel modo di utilizzare entrambi.

```
if x < y:
    print x, "e' minore di", y
elif x > y:
    print x, "e' maggiore di", y
else:
    print x, "e", y, "sono uguali"
```

```
if x == y:
    print x, " e ", y, "sono uguali"
    else:
        if x < y:
          print x, " e' minore di ", y
        else:
          print x, " e' maggiore di ", y
```

Prima di assegnare esercizi, abbiamo considerato l'idea di dedicare del tempo agli studenti per organizzare mentalmente le nozioni insegnate lasciandogli inventare, in gruppo, degli esempi per esprimere cosa hanno capito riferendosi a situazioni della propria vita (usando espressioni logiche, If annidati etc..). Esempio semplice di questo genere: "se è Lunedì o .. Martedì.. o Venerdì, metti la sveglia alle 7, altrimenti disattivala perchè non hai scuola".

Error Analysis

Riteniamo interessante provare ad approcciare gli studenti con degli esercizi di Error Analysis, in cui proponiamo ai ragazzi spezzoni di codici semplici e mirati al far uscire fuori problemi e misconceptions comuni, in modo da far ragionare e riflettere sui concetti.

Alcuni esempi riguardo il costrutto If..Elif sono i seguenti:

1) Realizzare il programma "FizzBuzz": se il numero inserito in input è multiplo di 3 stampa "Fizz" mentre se è multiplo di 5 stampa "Buzz". Nel caso in cui il numero è multiplo sia di 3 che di 5 stampa "FizzBuzz".
   Nel codice riportato qui sotto vengono messi in evidenza tre errori di sintassi e di indentazione base.

versione con errori **versione corretta**

```
numero = int(input("Inserisci un numero: "))

if numero %3 == 0 and num %5 == 0:
    print ("Fizzbuzz")
elseif numero %5 == 0:
    print ("Buzz")
elif numero %3 == 0:
    print ("Fizz")
    else:
        print ("Il numero e'", numero)
```

```
numero = int(input("Inserisci un numero: "))

if numero %3 == 0 and numero %5 == 0:
    print ("Fizzbuzz")
elif numero %5 == 0:
    print ("Buzz")
elif numero %3 == 0:
    print ("Fizz")
else:
    print ("il numero e'", numero)
```

2) Questo esempio, leggermente più complicato, riguarda gli If annidati. In questo caso, potrebbe tornare utile allo studente disegnare un flowchart per comprendere l'organizzazione gerarchica dei controlli.

versione con errori

```
tempo =input("Sta piovendo? \n")
if tempo == "si":
    vento = input("E' molto ventoso? \n")
if vento =="si":
        print ("E' troppo ventoso per l'ombrello")
    else:
        print ("Prendi un ombrello")
    else:
    print ("Buona giornata")
```

versione corretta

```
tempo =input("Sta piovendo? \n")
if tempo == "si":
    vento = input("E' molto ventoso? \n")
    if vento =="si":
        print ("E' troppo ventoso per l'ombrello")
    else:
        print ("Prendi un ombrello")
else:
    print ("Buona giornata")
```

3) Esempio esplicativo riguardo le espressioni booleane. Questo esempio potrebbe essere una buona occasione per spiegare le differenze tra un If con molti Elif rispetto a molti If annidati.

versione con errori

```
eta =int(input("Quanti anni hai? \n"))
if eta >= 3:
    print("Sei un neonato")
elif eta = 4:
    print("Fra poco inizi la scuola!")
elif eta != 16:
    print("Allora stai andando a scuola")
else:
    print("Adesso puoi andare a lavorare")
```

versione corretta

```
eta =int(input("Quanti anni hai? \n"))
if eta <= 3:
    print("Sei un neonato")
elif eta == 4:
    print("Fra poco inizi la scuola!")
elif eta <=16:
    print("Allora stai andando a scuola")
else:
    print("Adesso puoi andare a lavorare")
```

Possibili difficoltà generali degli studenti

- Per molti studenti questo sarà il primo approccio con la logica. È quindi necessario far ragionare se le affermazioni sono vere o false, e fare espressioni complesse usando And e Or. Se gli studenti avessero difficoltà con la logica, incoraggiarli a riflettere attentamente su ogni espressione condizionale, ad esempio scrivendo su un foglio di carta e ragionando.

- Gli studenti potrebbero mettere in sequenza i loro If/Elif in un flusso che non si traduce in ciò che vogliono. A causa della mancanza di esperienza e di una Notional Machine confusa lo studente potrebbe cercare una logica che nella pratica è sbagliata e non si traduce in una corretta implementazione. Ancora una volta, è necessario incoraggiarli a rallentare e a ragionare su come vogliono che il loro programma funzioni. Inoltre, farli riflettere su tutti i possibili valori che potrebbero fluire attraverso la logica. Insistere sul verificare bene i vari casi valutando casi che è facile dimenticarsi (come gli Edge Cases).

Esempi Turtle per l'istruzione condizionale If

In questo paragrafo analizzeremo **alcuni** degli esempi forniti in Turtle per quanto riguarda l'istruzione condizionale **If.**

L'esempio *if_quadrato_1.py* può essere usato per sperimentare le espressioni booleane che possono essere valutate a True o False in un costrutto **If**. Con gli studenti è possibile modificare una riga del codice per modificare la condizione, ipotizzando con gli studenti il risultato dell'esecuzione e verificare l'ipotesi eseguendo il file.

L'esempio *pari.py* è un codice che funziona solo in semplici casi, in questo caso l'argomento principale di discussione dovrebbe essere "Come far funzionare il programma con qualsiasi numero?" chiedendoci quindi come fare in caso di input non numerico, come riconosciamo qualsiasi numero pari e come implementiamo questi casi. A questo punto diventa banale modificare il codice per capire se un numero è dispari o meno e questo può essere un esercizio da proporre per far capire al docente se la discussione ha avuto esito positivo.

Ciclo: While

Premessa: perché introduciamo il While prima del For?

Durante la creazione dell'unità didattica, abbiamo visto il ciclo For come un caso specifico del ciclo While, adatto a risolvere dei problemi specifici molto comuni, ad esempio l'iterazione su una lista. Tuttavia, i destinatari dell'unità sono programmatori novizi, e non hanno ancora visto le liste.

Inoltre, i cicli For in Python sono molto diversi dai While: al contrario del C (e simili), in cui possiamo specificare la condizione che preferiamo nel For, in Python siamo limitati a iterare su liste. Dato che, come abbiamo fatto per l'If, introdurremo i cicli con i flowchart, potremmo creare delle false aspettative se non esistesse una corrispondenza tra flowchart e codice.

Per questi motivi abbiamo preferito introdurre inizialmente un concetto di ciclo generico, che permetta di specificare qualsiasi tipo di condizione, per poi arrivare al ciclo For.

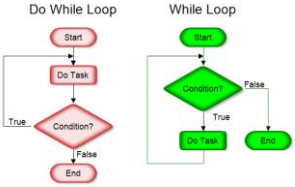
A cosa ci serve un ciclo?

Commentato [2]: Spiegazione rivolta ai ragazzi

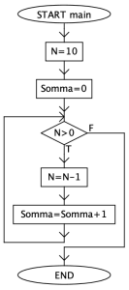
Come abbiamo visto, Il costrutto If-Elif permette di scegliere se eseguire o meno una sequenza di istruzioni sulla base di una condizione. Man mano che la complessità dei programmi aumenta, risulterà necessario ripetere più volte delle operazioni fintanto che una condizione rimane verificata. Un esempio che chiarisce la necessità è: "scrivi un programma che somma i numeri da 0 a *x*". Risulta intuitivo che per mappare tutti i possibili numeri *x* con le rispettive somme utilizzando gli If-Elif richiederebbe un codice infinito.

Flowchart: While e Do-while

Esistono due tipologie di ciclo While. La prima, in rosso, presenta la condizione al termine del blocco di istruzioni chiamato "task", che quindi verrà eseguito al minimo una volta, anche se la condizione risultasse falsa all'istante iniziale. La seconda tipologia, in verde, presenta la condizione prima del blocco "task". In questo caso, le istruzioni verranno eseguite fintanto che la condizione è e resta vera.



Un metodo che abbiamo considerato per chiarire queste sottili differenze consiste nell'introdurre nella spiegazioni delle vignette umoristiche per evidenziare i concetti che spieghiamo attraverso metafore e/o similitudini. L'immagine a sinistra mostra un esempio.



Successivamente alla parte iniziale di ragionamento, si può passare ad analizzare concettualmente come il ciclo While si comporta e viene utilizzato. Come fatto per l'If, iniziamo con la realizzazione di soluzioni in flowchart.

Iniziamo dal seguente esempio: "incrementare un numero 10 volte".
La prima cosa da fare è definire una variabile `N=10`, che rappresenta il numero di incrementi che vogliamo fare. Qui si presenta al professore l'opportunità di spiegare il concetto di variabile contatore.
Serve inoltre una variabile *somma*, che tiene il conteggio dei vari incrementi. Quindi si può ragionare sulla condizione e al comportamento del While. Il ragionamento sarebbe: "Finché ho ancora incrementi a disposizione, incrementa somma di 1".

Una volta realizzati diversi algoritmi in flowchart si può passare al codice in Python.

Istruzioni break e continue

Nel linguaggio Python esistono due istruzioni non strettamente necessarie, ma per completezza riteniamo utile quantomeno menzionarle. L'istruzione Break consente di terminare l'esecuzione di un ciclo in qualsiasi istante. L'istruzione Continue permette di annullare una iterazione del ciclo per passare alla successiva. A differenza di Break, quest'ultima non causa l'uscita forzata dal ciclo ma soltanto l'interruzione anticipata della corrente iterazione.

Error Analysis

Come già per il costrutto If, anche nella sezione del While intendiamo proporre esercizi di Error Analysis. I seguenti codici possono essere utilizzati per evidenziare gli errori più comuni, ad esempio ciclo infinito, sintassi sbagliata o condizione sbagliata.

```
i = 0
while i < 10:
    print(i)
```

```
i = 0
while i < 10:
    print i
    i += 1
```

```
i = 0
while i < 0:
    print(i)
    i += 1
```

Esempi Turtle per il ciclo While

L'esempio *quadrato_while.py* è una semplicissima trasformazione in ciclo di uno degli esempi affrontati inizialmente (*quadrato.py*), dove viene implementato il ciclo introducendo il ruolo della variabile contatore. In questo esempio è possibile affrontare diversi argomenti che possono trasformarsi in misconception, come il numero di cicli necessari, l'inizializzazione del contatore prima del ciclo e quando incrementare il contatore.

L'esempio *pari_2.py* implementa il concetto di ciclo infinito (`while(true)`) ed uno dei suoi scopi più diffusi, l'interazione tra utente e macchina. Questo esempio viene implementato sulla base di un codice già visto in precedenza come quello di *pari.py* per alleggerire il carico cognitivo, consentendo allo studente di focalizzarsi sull'uso corretto del costrutto piuttosto che su aspetti meno importanti come la sintassi, la struttura del If interno o altro.

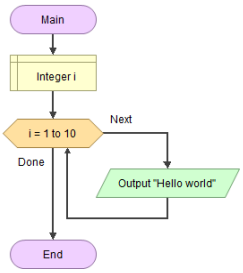
Ciclo: For

Perchè un nuovo ciclo?

La nostra unità didattica introduce la lezione dei cicli For spiegando ai ragazzi come mai parliamo di un nuovo tipo di ciclo. Per fare ciò, facciamo notare agli studenti che, nei programmi che hanno realizzato, hanno programmato diverse volte una struttura molto simile. Questo ci permette di introdurre il concetto di un pattern che si ripete, e possiamo spiegare che i programmatori cercano spesso di creare dei costrutti che permettano di isolare un pattern, allo scopo di semplificare la programmazione e di ridurre la duplicazione di codice.

Struttura dei For nei diagrammi di flusso

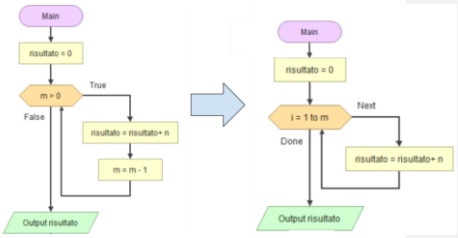
A questo punto, mostriamo ai ragazzi un piccolo diagramma di flusso che utilizza il For, e lo analizziamo insieme. Vogliamo porre la loro attenzione sulla scelta del nome “i” della variabile contatore: spieghiamo che “i” non è una lettera scelta “a caso”, ma è il nome convenzionale. In questo modo, un programmatore che legge il nome “i” capisce immediatamente che si sta parlando di un contatore.



Tuttavia, dobbiamo far notare che non è obbligatorio chiamare la variabile “i”, ma è possibile scegliere un altro nome se più adatto. Dare un nome “a caso” alla variabile può essere comunque utile a scopo didattico, con il fine di rendere chiaro che il computer “non vede” il significato del nome.

Esercizi di conversione

Una volta vista la struttura del for, proponiamo un esercizio alla lavagna in cui i ragazzi convertono algoritmi già visti che utilizza il While nei corrispondenti con il For.

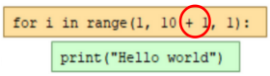


Riportiamo ora un esempio dello svolgimento di questo tipo di esercizio, considerando un programma che calcola la moltiplicazione di *n* per *m* utilizzando il ciclo While. Il fine di questo esempio e di far notare che, a prima vista, pare che il For richieda una variabile in più. Questa è un’occasione per spiegare che prima usavamo la variabile *m* implicitamente come un contatore, mentre ora *m* non viene più modificata.

For in Python

È giunto il momento di tornare a Python, e quindi mostriamo la conversione da flowchart a codice. I ragazzi sono già familiari con il concetto del For, quindi riteniamo sia sufficiente spiegare solamente come convertire la sintassi da diagramma di flusso a codice.

Vari modi di utilizzare la funzione range



Successivamente, proponiamo delle discussioni sul funzionamento degli indici di Range. Per esempio, è possibile prendere in considerazione il "+1" che Flowgorithm aggiunge in automatico quando genera del codice Python. Per spiegare il concetto di indici inclusi/esclusi, i ragazzi possono sperimentare con diverse stampe del contatore.

Terminata la sperimentazione con gli indici, proponiamo di continuare la discussione su Range per diminuire il numero di argomenti. Ancora una volta si presenta per il professore la possibilità di spiegare i pattern di programmazione. Per esempio, possiamo parlare del fatto che è comune incrementare il contatore di un'unità, e che quindi possiamo risparmiare un parametro. Inoltre, possiamo introdurre il concetto di "contare da 0", arrivando alla versione più semplice di Range, dove specifichiamo un unico parametro.

Esercizi di error analysis

Per concludere la spiegazione dei cicli For, mostriamo alcuni programmi che includono degli errori. Un possibile programma potrebbe essere il seguente:

versione con errori

```
n = int(input("Quanti valori vuoi inserire? "))
somma = 0

for i in range(n):
    valore = int(input("Inserisci il prossimo valore: "))
    somma = somma + valore
    i = i + 1

media = somma / n
print("La media è: {}".format(media))
```

versione corretta

```
n = int(input("Quanti valori vuoi inserire? "))
somma = 0

for i in range(n):
    valore = int(input("Inserisci il prossimo valore: "))
    somma = somma + valore

media = somma / n
print("La media è: {}".format(media))
```

Questo programma calcola la media di *n* numeri inseriti dall'utente, ma contiene una riga che non dovrebbe esserci: l'incremento "manuale" della variabile contatore *i*. Da notare è il fatto che anche la versione del programma errata produce il risultato corretto, ma indica una misconception.

Esempi Turtle per il ciclo For

L'esempio *quadrato_for.py* dimostra come un programma scritto in precedenza come *quadrato_while.py* possa essere riscritto usando il costrutto For, per questo esempio gli spunti di discussione sono molto simili a quelli affrontati in *quadrato_while.py.*

L’esempio *da_while_a_for.py* è usato come dimostrazione delle differenze nella sintassi e nell’esecuzione dei costrutti While e For. Inoltre, è presente anche la misconception dell’incremento manuale del contatore.

L’esempio *for_annidati.py* implementa il concetto di cicli annidati. I valori presenti in questi cicli possono essere modificati per visualizzarne le conseguenze.

Con *turtle_race.py* si vuole stimolare la curiosità e l’entusiasmo dello studente fornendo questo videogioco estremamente semplice che contiene tutti i concetti affrontati nell’unità.

Note

Funzione range

In questa unità didattica stiamo “barando” sul modo in cui spieghiamo il ciclo For: Range è in realtà una funzione che ritorna un iteratore su una lista che contiene i valori del contatore. Nonostante ciò, abbiamo preferito spiegare il concetto “classico” del For (come per esempio quello utilizzato nel linguaggio C). Questo “imbroglio” può essere svelato una volta introdotte le liste.

Step decimali

Man mano che i ragazzi eseguono esercizi di programmazione, potrebbero essere tentati di utilizzare valori decimali come step del ciclo For. È importante far notare che il computer non può rappresentare i numeri decimali in modo esatto, ma li può solo approssimare, e che quindi è meglio evitare questo approccio.

Bibliografia

Mannila, L., Peltomaki, M., Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. Computer Science Education, 16, 3, 211–227.

Hromkovic, J. (2006). Contributing to general education by teaching informatics. In: Mittermeir, R.T. (Ed.), ˇ ISSEP 2006, LNCS, 4226, 25–37.

Mara Saeli, Jacob Perrenet, Wim M. G. Jochems, Bert Zwaneveld. Teaching programming in secondary school: A pedagogical content knowledge perspective.

Stuart Wray, Royal School of Signals, How Pair Programming Really Works