Unità 9

Capire la logica del coding

Livello 3 - approfondimento

Temi trattati all'interno dell'Unità

- Pensiero Computazionale
- La mutazione proveniente dal Pensiero Computazionale.
- I concetti di riferimento del Pensiero Computazionale
- La logica deduttiva.

Sommario

"NON LIMITATEVI A GIOCARE CON IL VOSTRO TELEFONO CELLULARE, PROGRAMMATELO"

L'ex Presidente degli Stati Uniti Barack Obama lanciò questo messaggio durante la settimana dedicata alla "Computer science education" nel dicembre del 2013.

Dopo la nascita di internet e l'ampio utilizzo del web, la normale comunicazione è stata affiancata da sistemi di messaggistica sempre più raffinati e portabili. I social network raccolgono imponenti quantità d'informazioni sulla vita pubblica e privata degli individui e la stragrande maggioranza dei dispositivi elettrici, elettronici e meccanici vengono sempre più comandati a distanza con l'informatica.
L'informatica infatti non è più relegata nell'ambito delle attività professionali specifiche del mondo ICT ma è in contatto con noi ogni volta che utilizziamo un bancomat, acquistiamo on-line biglietti e pernottamenti, passiamo con il telepass sulle autostrade.
Il fenomeno è rapido e pervasivo e non può più essere ignorato. Non avere la consapevolezza di questo contesto comporta quasi una forma di rinuncia al controllo delle nostre azioni quotidiane.

L'informatica è presente e incorporata (**embedded**) nelle auto, nel riscaldamento, negli elettrodomestici e in ogni strumento di comunicazione digitale utilizzato. Comprendere il "linguaggio" dell'informatica diventa un bisogno, sia per l'utilizzo degli strumenti che usiamo e che ci circondano, sia per comprendere appieno come vengono trattate le informazioni che gestiamo in prima persona o che facciamo gestire.
Come proseguiva Obama: "l'abilità di scrivere codice informatico è una competenza importante, trasforma le persone da consumatori di tecnologie in creatori di tecnologie."
Conoscere il linguaggio dell'informatica permette di apprendere e praticare il pensiero applicato nelle soluzioni individuate e utilizzate. Conoscere questo pensiero permette di aggiungere una "lingua" ulteriore e un approccio

creativo alla soluzione dei problemi. “Pensare informatico” significa creare algoritmi o scomporre un problema in sottoproblemi, ovvero semplificare.

PENSIERO COMPUTAZIONALE

Il pensiero computazionale è un processo mentale per la **risoluzione di problemi** ed è costituito dalla combinazione di metodi e di strumenti intellettuali caratteristici.
Questo pensiero non è solo direttamente applicabile all’ICT, alle reti, ai sistemi informatici, ma fornisce strumenti e approcci concettuali per affrontare problemi in molti ambiti e discipline.

Anche dal punto di vista formativo, il lavoro stesso, qualsiasi lavoro, è pervaso dalle tecnologie informatiche e della comunicazione. Uno studente, nel terzo millennio, dovrà apprendere e comprendere i concetti di base di questo pensiero applicato nell’informatica così come nelle evoluzioni del passato apprendeva e comprendeva matematica, fisica, chimica, biologia.

Nel passato la stampa e la rivoluzione industriale hanno avuto un approccio significativo nei cambiamenti sociali, economici ed evolutivi delle persone. La stampa a caratteri mobili ha fatto esplodere la conoscenza rendendola accessibile a tutti senza vincoli di spazio e di tempo. La successiva rivoluzione industriale è intervenuta sugli aspetti materiali: macchine che replicano il lavoro manuale, produzioni più rapide, significativi cambiamenti nel rapporto e nelle condizioni economiche tra persone.

L’informatica è intervenuta in modo ancor più rivoluzionario abbinando l’impatto sia sul materiale che sull’immateriale. Gestione, trattamento e scambio dell’informazione, tecnologie evolutive e abilitanti su apparecchiature e macchinari.
Se le rivoluzioni precedenti sono avvenute in due secoli e mezzo, la rivoluzione informatica ha una rapidità maggiore nel continuo cambiamento operato sul sociale, sul piano della comunicazione della produzione e del commercio.
Interviene a livello globale, e ancor di più interviene sul piano cognitivo e relazionale delle persone. La conoscenza, l’applicazione della conoscenza gestita con le informazioni, in ogni aspetto lavorativo ed economico ha trasformato la società attuale in “società della conoscenza e società dei servizi”.

I benefici del “pensiero computazionale“ si estendono a tutte le professioni. Impiegati, dirigenti, medici, avvocati, architetti, ogni giorno affrontano problemi complessi. Diventa necessario individuare soluzioni che prevedono più fasi e la collaborazione con colleghi o collaboratori per immaginare una descrizione chiara di cosa deve essere fatto e quando farlo.

Il pensiero computazionale va ben oltre l'uso della tecnologia, ed è indipendente da essa (sebbene la sfrutti intensivamente): non si tratta di ridurre il pensiero umano, creativo e fantasioso, al mondo “ripetitivo e meccanico” di un calcolatore, bensì di far comprendere all'uomo quali sono le reali possibilità di estensione del proprio intelletto attraverso il calcolatore. Si tratta di risolvere problemi, progettare sistemi, comprendere il comportamento umano basandosi sui concetti fondamentali dell'informatica.

Riconosciuta la sua importanza, il pensiero computazionale può essere proposto come **quarta abilità di base oltre a leggere, scrivere e calcolare.**

CONCETTI DEL PENSIERO COMPUTAZIONALE

Il concetto di pensiero computazionale è stato fortemente approfondito nel 2006 da Jeannette Wing, accademica e manager americana, che ha mostrato come l'informatica abbia portato alla scienza non solo strumenti (computer e linguaggi di programmazione) ma anche innovazioni nel modo di pensare: “come leggere, scrivere e contare sono abilità che è importante imparare fin da bambini così il pensiero computazionale deve essere appreso ed esercitato fin dai primi anni di scuola”.

Un esempio illuminante di informatica applicata a discipline diverse è il sequenziamento del DNA umano, avvenuto nel 2003, in forte anticipo sulle previsioni, grazie all'adozione di un algoritmo di tipo shotgun (basato su metodi random) per ricombinare i frammenti di DNA analizzato.

Il **pensiero computazionale** è un **processo di problem-solving** che consiste nel:

- formulare problemi in una forma che ci permetta di usare un computer (nel senso più ampio del termine, ovvero una macchina, un essere umano, o una rete di umani e macchine) per risolverli;
- organizzare logicamente e analizzare dati;
- rappresentare i dati tramite astrazioni, modelli e simulazioni;
- automatizzare la risoluzione dei problemi tramite il pensiero algoritmico;
- identificare, analizzare, implementare e testare le possibili soluzioni con un'efficace ed efficiente combinazione di passi e risorse (avendo come obiettivo la ricerca della soluzione migliore secondo tali criteri);
- generalizzare il processo di problem-solving e trasferirlo ad un ampio spettro di altri problemi.

In breve, parliamo del **modo di esprimere le istruzioni naturali e usare strumenti di visualizzazione**

Non è facile strutturare l'insegnamento del pensiero computazionale: la pratica di usare la programmazione come contesto formativo resta a oggi il veicolo più efficace.
Mitchel Resnick responsabile del Lifelong Kindergarten del MIT MediaLab ha, con i suoi collaboratori, realizzato Scratch, un ambiente di programmazione visuale che consente ai ragazzi di creare in modo semplice e intuitivo storie animate, giochi e simulazioni: oggi Scratch vanta una community di giovani sviluppatori in tutto il mondo.
Il framework sviluppato dal Lifelong Kindergarten del MIT MediaLab identifica alcuni concetti base del pensiero computazionale:

- **Sequenza:**
  è un'attività espressa attraverso una serie consecutiva di singoli step o istruzioni.
- **Ciclo:**
  esecuzione in maniera iterativa della medesima sequenza.
- **Evento:**
  il verificarsi di un evento innesca un'azione successiva.
- **Parallelismo:**
  significa eseguire sequenze differenti allo stesso tempo.
- **Condizione:**
  permette di prendere decisioni sulla base del verificarsi di determinate situazioni.
- **Operatore:**
  permette la manipolazione di numeri e stringhe di caratteri.
- **Dati:**
  valori che possono essere salvati, recuperati e modificati durante l'esecuzione di un programma.

E applica pratiche quali:

- **Essere incrementali e iterativi:**
  progettare e realizzare è un processo adattativo che procede per iterazioni e incrementi.
- **Testare e debuggare:**
  verificare e quindi individuare ed analizzare eventuali errori per poterli correggere.
- **Riusare:**
  riconoscere le componenti di una soluzione riusabili nella stessa realizzazione o riapplicabili a problemi simili.
- **Remixare:**
  è possibile prendere spunto da altre idee per costruire soluzioni più complesse di quelle che si potevano realizzare per conto proprio, dando di conseguenza un'ulteriore spinta alla propria creatività (*copiare per migliorare*).
- **Astrarre:**
  processo di riduzione della complessità, per lavorare solo sull'idea principale mantenendo solo alcuni aspetti e tralasciandone altri.
- **Modulare:**
  scomporre un problema complesso in problemi più semplici, in modo tale che, risolvendo i problemi più semplici, si arrivi anche a risolvere il problema complesso (*scomporre*).

Sviluppa inoltre attitudini come:

- **Esprimere sé stessi:**
  una persona che applica il pensiero computazionale vede nella tecnologia uno strumento per esprimere sé stesso, la propria creatività e dire qualcosa di sé ad altri.
- **Essere connessi:**
  è necessario saper comunicare e lavorare con gli altri per raggiungere un obiettivo o una soluzione condivisa.
- **Porre domande:**
  avere una mente vigile per porsi sempre la domanda di come possa funzionare un oggetto incontrato nel mondo reale.

La logica è l'anatomia del pensiero (John Locke)

Per risolvere un problema (computazionale), bisogna coltivare e sviluppare la logica attraverso la comprensione iniziale della natura del problema, seguita poi dalla progettazione di un'appropriata procedura di risoluzione algoritmica (se possibile!), ovvero una sequenza di istruzioni formali che possono anche essere tradotte in un linguaggio intellegibile al computer (programma), il tutto deve concludersi da una verifica della correttezza del risultato
La logica è necessaria e utile per un pensiero corretto e razionale, e la logica del linguaggio naturale si esprime anche simbolicamente.
La logica può essere applicata per risolvere una varietà di problemi del mondo reale.
La vita è fatta di decisioni quotidiane che, nel tempo, determinano il percorso della nostra storia personale. Molte decisioni implicano la realizzazione di scelte importanti: chi sposare, quale percorso di carriera intraprendere, cosa mangiare.

Le decisioni possono basarsi su convinzioni credenze o istinto, qui vedremo però la logica come la base adatta alla gestione del pensiero e delle scelte razionali. Questo modello di pensiero è espresso in modo tale da essere compreso anche da un computer. Ma a parte questo, ci renderemo conto che questi modelli sono pervasivi e utili per risolvere problemi reali.

Lo studio della logica può essere suddiviso in due categorie: logica induttiva e logica deduttiva
La logica induttiva è un tipo di ragionamento che inizia con una serie di osservazioni o esperienze da cui si possono derivare conclusioni.
Come esempio di ragionamento induttivo, consideriamo uno scenario in cui avete mangiato i cavoli di Bruxelles solo due volte nella vita. Siete stati male dopo averli mangiati. Concludete che siete allergici ai cavoli di Bruxelles.
La conclusione può essere ragionevole ma non certa. Forse eravate esposti a un virus prima di mangiarli e questo virus era la vera causa.
Nella **logica deduttiva**, al contrario, si assume che certe cose sono vere e da queste discendono altri fatti che sono veri. Le conclusioni che si possono raggiungere sotto questa logica sono vere se le ipotesi sono vere.
Classico esempio:
"Tutti gli uomini sono mortali e Aristotele è un uomo".
Da queste due ipotesi si può dedurre con assoluta certezza che Aristotele è mortale".

NON SONO PIÙ INTELLIGENTE, ANALIZZO I PROBLEMI PIÙ A LUNGO

Come ci suggerisce questa frase di Albert Einstein, gli analisti informatici non sono più intelligenti, semplicemente applicano delle strategie di base per analizzare e trovare soluzioni ai problemi:

1. Definizione del Problema
2. Logica e Scomposizione
3. Astrazione

Definizione del Problema

Definire bene il problema significa capire e definire attentamente il bisogno. Senza questa attenzione non si possono né comprendere le attività che possono soddisfare il bisogno né capire se il bisogno è risolto, qualsiasi sia l'intervento (attività umane o applicazioni digitali).
In questa fase è essenziale il coinvolgimento di tutte le persone, sia chi analizza sia, e soprattutto, chi richiede la soluzione.
È in questa fase che si parla di **Analisi** quando si progettano sistemi informatici. E quando si fa Analisi si identificano i **Requisiti**, ovvero gli aspetti relativi alla soluzione richiesta.
I Requisiti sono di due tipi: **funzionali** e **non funzionali**. Quelli funzionali riguardano le attività richieste per realizzare il bisogno, quelli non Funzionali riguardano le caratteristiche e i vincoli sulla soluzione; ad esempio possono riguardare la sicurezza, la velocità o la facilità di esecuzione.

Logica e scomposizione

I Requisiti devono essere:

Chiari: descritti e compresi da tutti, soprattutto da chi richiede la soluzione
Coerenti: non devono esserci contrapposizioni con un altro requisito
Completi: vanno analizzati tutti gli scenari possibili della soluzione.

Il ragionamento logico si applica analizzando i passi e applicando il concetto di **causa-effetto**, ovvero SE "accade questo" SUCCESSIVAMENTE "si fa questo" ALTRIMENTI "si fa altro".
Tutti i passi possono essere pensati e descritti in questo modo per capire come si agisce e come si arriva a una soluzione.
Applicare la logica e analizzare relazioni causa-effetto significa comprendere anche la complessità delle attività. Spesso, la semplificazione avviene con il concetto di **scomposizione** ovvero il "*divide and conquer"* ossia separa (dividi) e risolvi (conquista).
Questo permette di affrontare la soluzione risolvendo passi intermedi coerenti, autonomi e necessari al quadro di insieme.
Scomporre significa semplificare e affrontare la soluzione passo dopo passo.
Pensate al fare una pizza. E scomponete l'azione in questo modo:

(1) preparo la pasta
(2) cuocio e verso la salsa
(3) spargo il formaggio
(4) aggiungo condimenti secondo richiesta
(5) metto in forno e aspetto
(6) tolgo servo e taglio

Astrazione
L'astrazione è tutto ciò che permette di concentrarci sulle caratteristiche importanti tralasciando le meno importanti, distraenti, o che non hanno importanza nel contesto analizzato.
Facciamo astrazione se diciamo: "ho visto una Ferrari rossa".
Ovvero per noi è più importante il modello e il colore piuttosto che le caratteristiche tecniche.
Astrazione è comunicare, significa utilizzare i termini essenziali per soddisfare la comunicazione che vogliamo.
L'astrazione viene utilizzata nell'analisi. Permette di concentrarci sui termini giusti e adatti al problema da risolvere.
Può essere fatta una **Astrazione sui dati e le informazioni** e una **Astrazione funzionale** sui passi analizzati.
Astrazione Funzionale sulla pizza significa semplificare fasi già scomposte:

(1) preparo la pasta
(2) **condisco secondo richiesta** (comprende: verso la salsa, spargo il formaggio, aggiungo dei condimenti)
(3) metto in forno e aspetto
(4) tolgo servo e taglio

Ho astratto "il condimento" per evitare distrazioni sui passi generali e importanti di base: Pasta, Condimento, Cucino..
Astrazione sui dati, significa capire come un oggetto contribuisce in termini di informazioni possedute (**cos'è**) e fornite (**cosa fa**).

Ogni giorno somiglia sempre a una sequenza di problemi da affrontare e provare a risolvere.
Spesso non sappiamo come e dove cominciare.
Qui abbiamo visto due proposte per **applicare la logica nella soluzione dei problemi**: comincia con una chiara definizione del problema e poi astrai e individua le caratteristiche importanti ed essenziali.
Il "*divide and conquer*", separa e risolvi, è applicabile sia al digitale che al giorno per giorno.

ALGORITMI

Molte attività, anche di vita normale, comportano sempre una sequenza di passi e azioni per risolvere problemi o giungere a una conclusione.
Cuocere una pizza, diagnosticare una malattia, progettare un prodotto sono esempi in cui è possibile applicare il pensiero algoritmico con i suoi concetti di base.
Un Algoritmo è una sequenza di azioni che porta a un obiettivo o a una soluzione di un problema.
Azioni, sequenza, condizioni e ripetizioni sono i concetti presenti nel pensiero algoritmico.
Anche senza seguirne precisamente ogni passaggio logico, nelle nostre normali attività spesso noi applichiamo e affrontiamo questi passaggi.

L'algoritmo può prevedere dei valori input e dei valori di output forniti alla fine della sequenza delle azioni.
Spesso gli output sono trasformazioni effettuate sui valori di ingresso.
Ogni singola azione dell'algoritmo deve avere un significato e uno scopo. Altra proprietà importante è la corretta sequenza delle azioni. Inoltre una proprietà, scontata ma da ribadire, è che l'algoritmo deve terminare sempre con un risultato, qualsiasi esso sia.

Flusso e Azioni

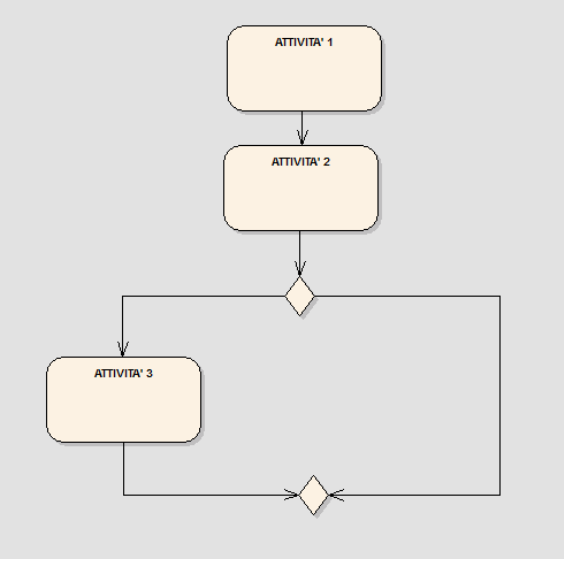
Il flusso delle azioni e il controllo del flusso (*Control flow*) è un concetto importante che fornisce flessibilità all'algoritmo che definiamo nel risolvere un problema.

Alcune azioni infatti possono essere differenti e necessarie a input differenti.

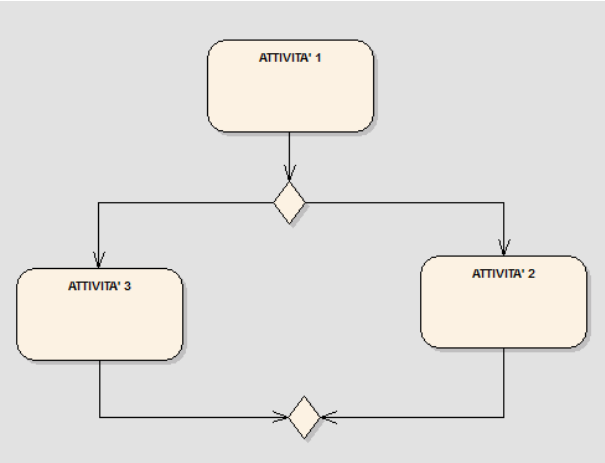
Per gestire questa possibilità l'algoritmo presenta il concetto di decisione, ovvero "una losanga" che differenzia i percorsi secondo certi assunti analizzati.

Ci sono passaggi in cui decidiamo se:

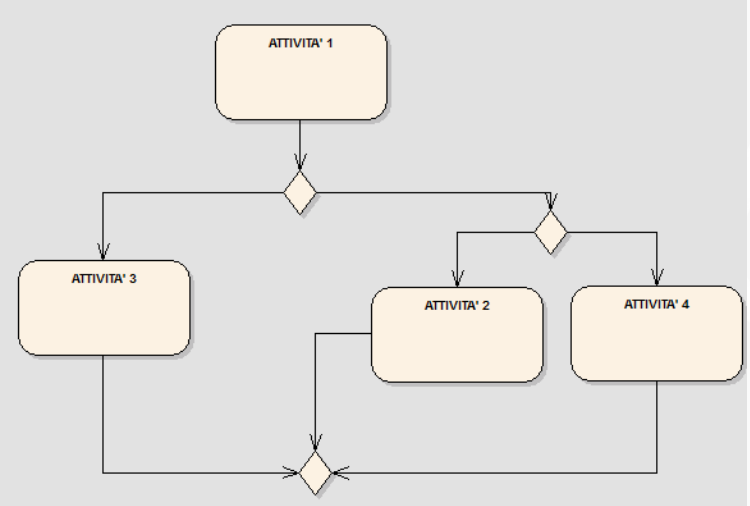
- effettuare o no una attività (*one-way selection)*



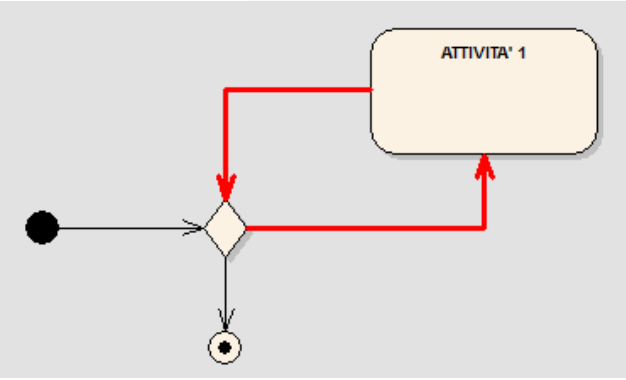
- scegliere quale attività eseguire tra due possibili *(two-way selection)*



- effettuare scelte multiple su più elementi di decisionali (*multiway selection)*



Ripetere Azioni
Lava, risciacqua, riapplica. Classica scritta sullo shampoo.
Che ci laviamo i capelli che ritinteggiamo casa, facciamo azioni che spesso devono essere ripetute per arrivare al risultato desiderato.
Questo è il **loop**, un elemento di controllo nell'algoritmo che permette di ripetere delle azioni basate su condizioni.
Quando si definisce un loop,



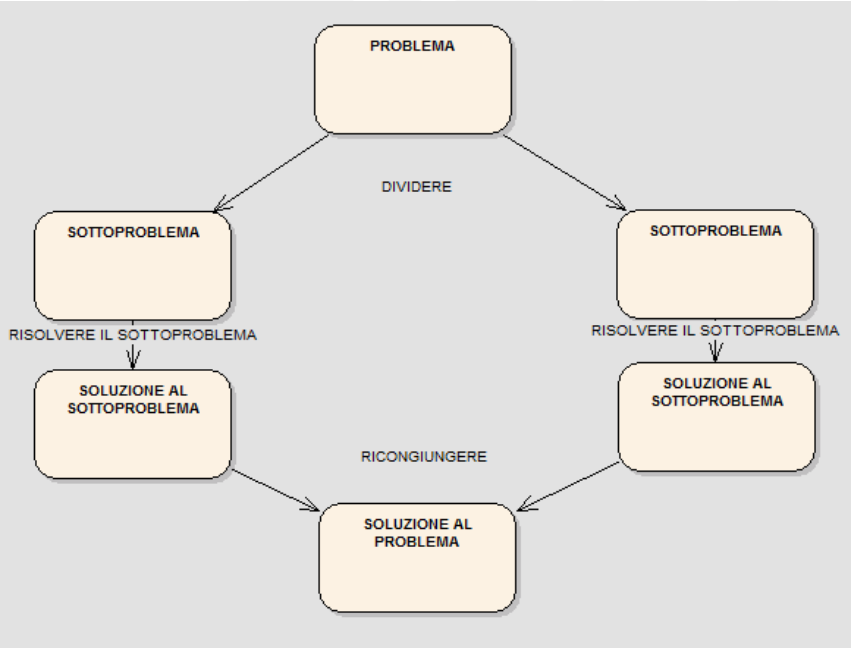
si definisce una condizione per capire se l'azione o le azioni devono essere ripetute una volta o più volte.
La condizione può essere una domanda Vero/Falso o una ripetizione basata su un ciclo che ripete la stessa sequenza di azioni per un numero di volte.

In quest'ultimo caso grande attenzione deve essere posta nel scegliere la variabile che fa ripetere il ciclo e comprendere come si incrementa per evitare cicli che non hanno fine e bloccano l'algoritmo (tipicamente *l'infinite loop*)
Ci si può allenare a creare i tuoi algoritmi grazie all'iniziativa **l'ora del codice**.
L'ora del codice è l'attività più utilizzata come primo approccio al coding, al punto di dare il nome alla campagna di alfabetizzazione lanciata da **Code.org** a partire dal 2013 (Hour of code, disponibile all'indirizzo hourofcode.com/it) nel corso della Computer Science Education Week.
I materiali e i metodi sviluppati e resi liberamente disponibili online da Code.org sono cresciuti nel tempo e oggi offrono diverse varianti e percorsi articolati che vanno ben oltre la singola ora di attività.
Non è richiesta alcuna conoscenza pregressa. Sono disponibili sia lezioni interattive che si presentano come semplici videogiochi la cui soluzione comporta l'esercizio del pensiero computazionale, sia lezioni tradizionali.

USARE MODULI PER SEMPLIFICARE

Gli algoritmi possono essere resi modulari creando dei sottoprocessi significativi che svolgono a loro volta algoritmi necessari a quello principale. Anche qui quindi, concetto di "*divide and conquer*" separa e risolvi e soprattutto semplifica.
Più l'algoritmo che elaboriamo diventa grande e complesso più è necessario verificare se è possibile modulare e semplificare la soluzione, è essenziale e ci aiuta in questo approccio l'analisi di questi criteri di modularità:

- **Comprensione**: il modulo deve essere autoconsistente, ovvero è comprensibile nelle sue azioni senza dover comprendere cosa succede esternamente ad esso.
- **Incapsulamento**: i dati e le informazioni gestite nel modulo devono essere trattate solo internamente al modulo per poi diventare eventualmente un output fornito.
- **Composizione**: il modulo può essere inserito in un algoritmo più grande senza nessun tipo di modifica.

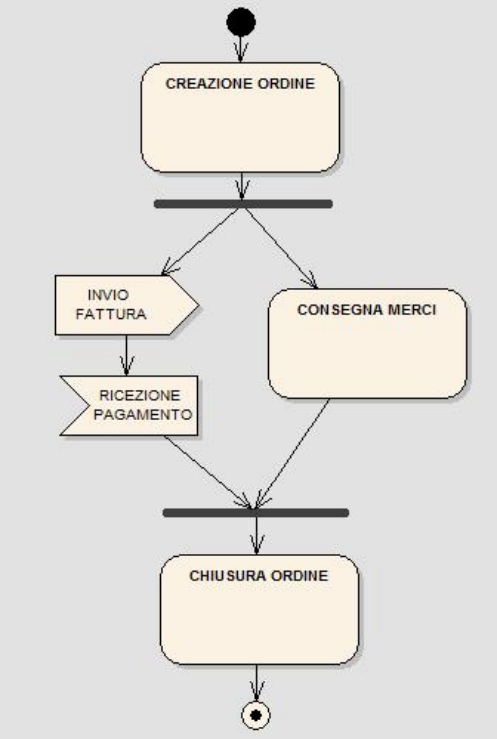


Parallelismo e Concorrenza
Le azioni spesso non si svolgono una alla volta e in sequenza.

Pensate al classico girone di calcio a eliminazione diretta. Posizionando gli incontri nel tempo, con un solo campo di calcio a disposizione i tempi si allungano per arrivare al vincitore del torneo. Ma se avessimo, nella stessa situazione, due risorse: ovvero due campi di calcio; immediatamente avremmo un risparmio di tempo per arrivare alla finale.

Questo è un esempio di **concorrenza** nell'utilizzo di risorse disponibili e in comune
In questo caso la parallelizzazione delle azioni (partite di calcio) è possibile per la disponibilità di più risorse utilizzabili (campi di calcio).

Il **parallelismo** a volte è invece la conseguenza di una situazione che si sta analizzando, esempio:



Il classico esempio di gestione dell'ordine di acquisto di un cliente prevede sia azioni in sequenza che azioni parallelizzabili perché:

- non utilizzano le stesse risorse (a meno che la persona che consegna la merce è la stessa che invia la fattura allora non è possibile parallelizzare),
- ottimizzano i tempi di gestione dell'ordine.

Parallelizzare azioni permette di risparmiare tempo per arrivare alla conclusione dell'algoritmo.

VERIFICARE LA SOLUZIONE: LA CORRETTEZZA

Verificare se la soluzione pensata, progettata e applicata a un problema è valida significa comprendere il concetto di "correttezza".
Correttezza vuol dire che la soluzione ha il comportamento atteso per il problema atteso o, ancor meglio, la soluzione fa quel che deve fare.
Come abbiamo visto precedentemente il primo passo è sempre la "definizione del problema" e questo significa definire dei requisiti che devono essere realizzati. I requisiti analizzati e definiti in chiaro sono, nell'ambito della progettazione informatica, le specifiche che devono essere "verificate e validate".
Verificare vuol dire che tutti i requisiti funzionali e non funzionali sono stati presi in carico nella soluzione, ovvero sono presenti e gestiti.
Validare significa che chi ha richiesto la soluzione (chiamiamolo "cliente") la prova e vede se arriva alla risoluzione della sua esigenza.
Effettuare i **Test** significa, in fase di progettazione, vedere se la soluzione si comporta come ci aspettiamo a fronte di una serie di prove (**casi di test**) che definiamo.

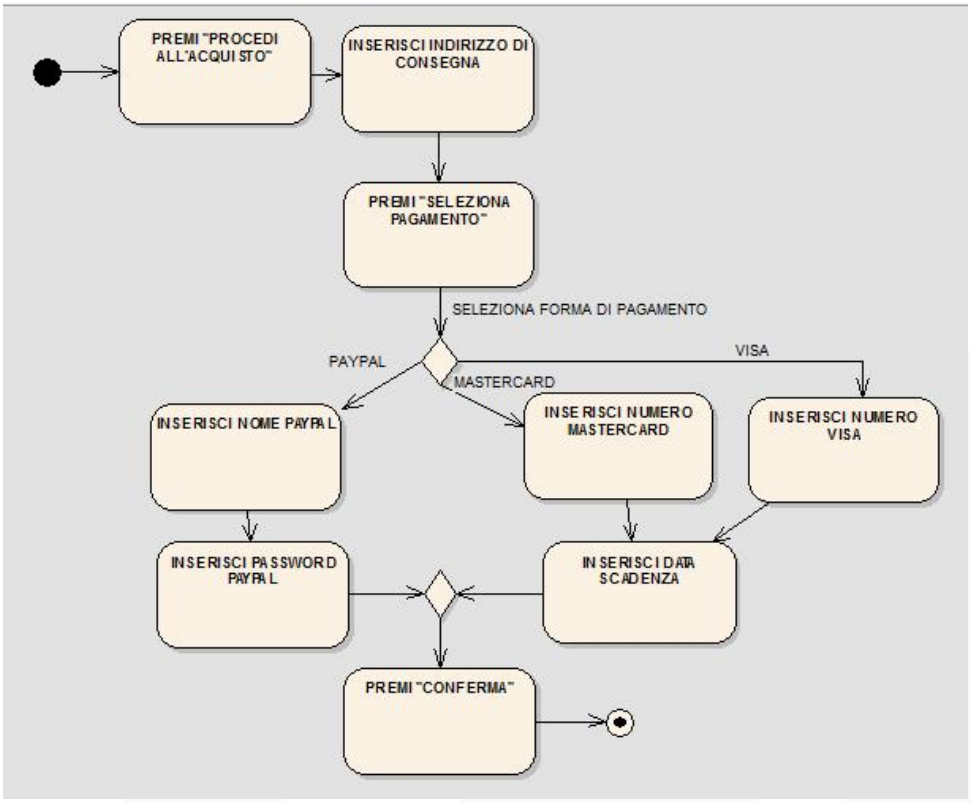
La prima cosa di cui prendere atto quando proviamo e simuliamo una soluzione, e dobbiamo poi individuare dei casi di prova, è che **è impossibile provare tutto** e, in ambiti reali di progetti digitali, **è costoso provare quasi tutto**.

Esaminiamo le tecniche migliori per individuare i casi di prova (o **casi di test**).
Si tratta di individuare al meglio degli input o degli eventi che simulano l'attivazione dell'algoritmo e sapere quali output ci attendiamo per la corretta gestione della soluzione, quindi del comportamento atteso e attuato sull'input adottato.

Copertura dei flussi

Una tecnica è la copertura di tutti i percorsi proposti nell'algoritmo, tipicamente ci sono più percorsi perché ci sono punti di decisione o trattamenti di condizioni o valori diversi.

Di seguito un esempio di diagramma con percorsi di utilizzo carte di pagamento, le condizioni sono tutte da coprire per verificare il comportamento corretto dell'algoritmo.

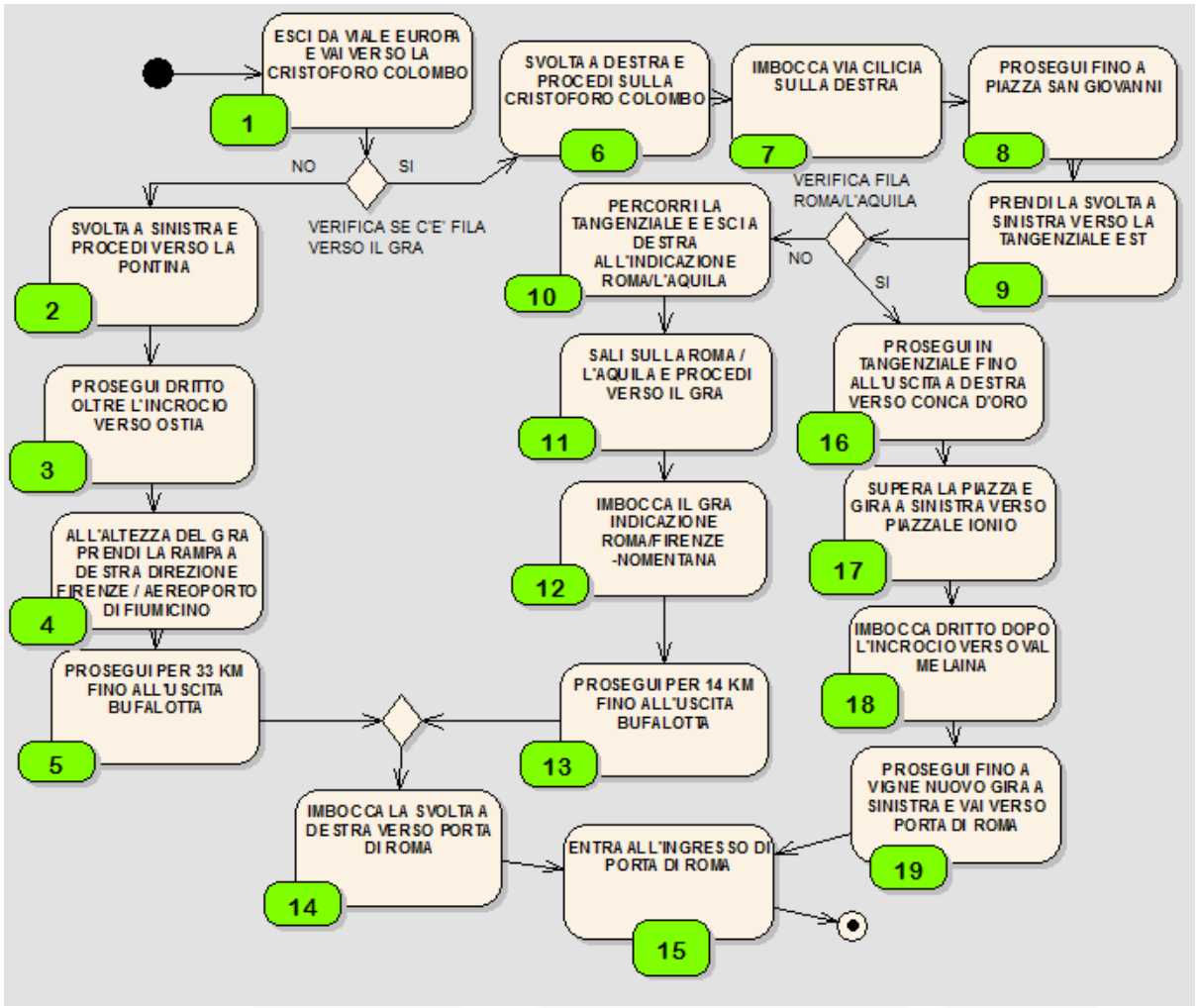


Dovremmo quindi provare i percorsi:

- PAYPAL
- MASTERCARD
- VISA

Per poter verificare che l'algoritmo svolga correttamente la sua funzione.

Questo approccio è più comprensibile se analizziamo un semplice algoritmo che ci permette di decidere il percorso da scegliere per arrivare ad una destinazione con punti di decisione e scelta basati sulla condizione del traffico. Nell'algoritmo che segue e simula una situazione di questo tipo sono numerati tutti i passaggi e rappresentati dei casi di test associati con indicate le attività coperte e l'*output* atteso.



Caso di Test	Attività da eseguire
Viale Europa/Porta di Roma (Via GRA)	1-2-3-4-5-14-15
Viale Europa/Porta di Roma (Via Autostrada Roma/L'Aquila)	1-6-7-8-9-10-11-12-13-14-15
Viale Europa/Porta di Roma (via Tangenziale est)	1-6-7-8-9-16-17-18-19-15

Caso di Test	Condizioni di Input	Risultato Atteso
Viale Europa/Porta di Roma (Via GRA)	GRA libero da traffico	Arrivo a Porta di Roma in 35 minuti Per 44 Km di tragitto
Viale Europa/Porta di Roma (Via Autostrada Roma/L'Aquila)	Troncone Autostrada Roma/L'Aquila libero da traffico	Arrivo a Porta di Roma in 35 minuti Per 33 Km di tragitto
Viale Europa/Porta di Roma (via Tangenziale est)	Troncone Autostrada Roma/L'Aquila con traffico	Arrivo a Porta di Roma in 1h Per 30 Km di tragitto

Poi c'è da decidere i casi di prova sui valori *boundary*, ovvero i valori di confine: significa analizzare i contesti in cui gli *input* sono spesso basati su valori numerici.
Applicare la *boundary value analysis* sulle condizioni di *input* da trattare e provare significa creare almeno 4 differenti casi di test:

1. con valore alla soglia di confine;
2. con valore al limite del confine (o appena dentro);
3. con valore fuori dal confine (o appena fuori);
4. con tipici valori dentro il confine.

Solo quest'ultimo caso di prova è un caso di prova normale, gli altri tendono a esercitare gli *input* che sono posizionati sul limite definito.

Spesso poi, nei casi limiti di *input* in cui il valore è un *range* tra un minimo e massimo di confine, i casi minimi di test diventano sette perché ai 4 precedenti che potremmo considerare come casi di prova sul limite massimo, sono da aggiungere 3 casi di test sul limite minimo con le stesse condizioni da esercitare, ovvero:

1. con valore alla soglia di confine minima;
2. con valore al limite del confine minimo (o appena dentro);
3. con valore fuori dal confine minimo (o appena fuori).

Applicandolo come esempio alla realizzazione di un software che stabilizza un'automobile per impedirle di ribaltarsi, supponendo che il nostro software operi ogni volta che l'automobile viaggia tra i 0 e i 150 Km/h con verifica che l'angolo di inclinazione non superi i 45°; i nostri casi di test saranno sui seguenti valori "limite" e il diagramma presenta la combinazione di entrambe le condizioni:

Caso di Test	Input	Output (Comportamento atteso)
Minimo	Velocità = 0Kmh	Auto non si muove
Minimo+1	Velocità = 1Kmh	Auto non si ribalta
Minimo-1	Velocità = -1Kmh	Auto in retromarcia senza rischio di ribaltamento
Massimo	Velocità = 150Kmh	Auto non si ribalta
Massimo-1	Velocità = 149Kmh	Auto non si ribalta
Massimo+1	Velocità = 151Kmh	Il software emette il messaggio di "Guida troppo veloce"
Caso tipico	Velocità = 70Kmh	Auto non si ribalta

APPLICARE I CONCETTI

Le tecniche che abbiamo analizzato nascono nel mondo computazionale ma sono applicabili ampiamente in qualsiasi ambito.

Sono conoscenze applicabili anche in situazioni di vita, ad esempio l'acquisto di una nuova auto. In questo caso è opportuno definire i requisiti che ci aspettiamo: consumo accettabile, costi, colori, valore dell'usato. Poi possiamo verificare i nostri requisiti con le informazioni disponibili sui siti e le riviste che forniscono questi dati e raccogliere le opportune valutazioni per decidere.

Concludendo: queste tecniche del mondo computazionale sono applicabili a qualsiasi decisione e più è importante la decisione più estesa deve essere la strategia di prova e simulazione.

BIBLIOGRAFIA

TITOLO	AUTORE	EDIZIONI	ANNO
Algoritmi	Carlo Toffalori	Il Mulino	2014
Linguaggi, modelli, complessità	Giorgio Ausiello Fabrizio D'Amore Giorgio Gambosi Luigi Laura	Franco Angeli	2014
Coding a scuola	Alberto Pian	De Agostini Scuola	2016