

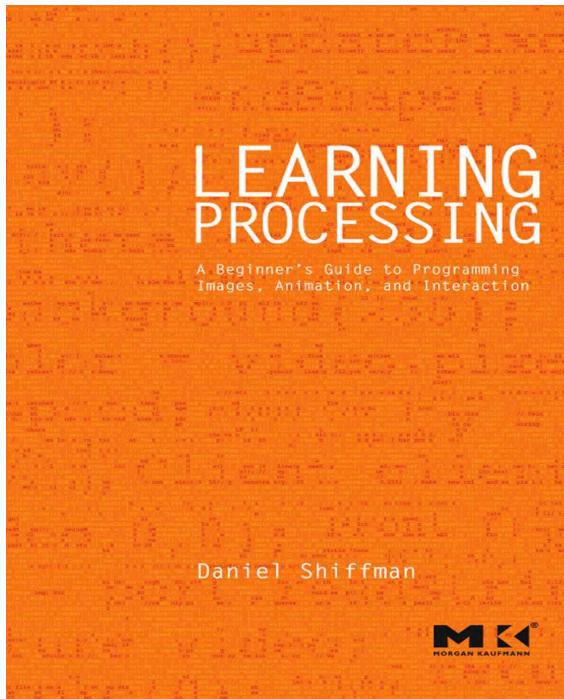


FONDAMENTI DI INFORMATICA

Alma Artis – Anno Accademico 2016/2017
Salvatore Rinzivillo (ISTI, CNR)

Variabili e Istruzioni Condizionali

LIBRI E RIFERIMENTI



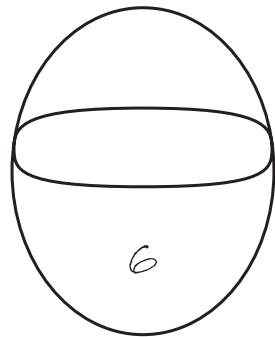
- Introduzione
- Capitolo 1-3
- Registrarsi al sito:
<http://almaartis.rinziv.it/>

Learning Processing– Second Edition
Daniel Shiffman
Available here: <http://learningprocessing.com/>

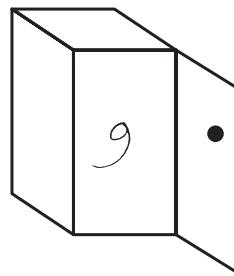


VARIABILI

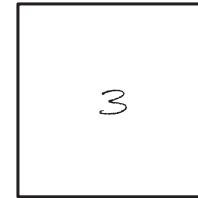
COSA È UNA VARIABILE?



variable
bucket



variable
locker



variable
post-it

fig. 4.1

Un nome per indirizzare una locazione della memoria del computer

USO DI VARIABILI

Jane's Score	Billy's Score
5	10
30	25
53	47
66	68
87	91
101	98

fig. 4.3

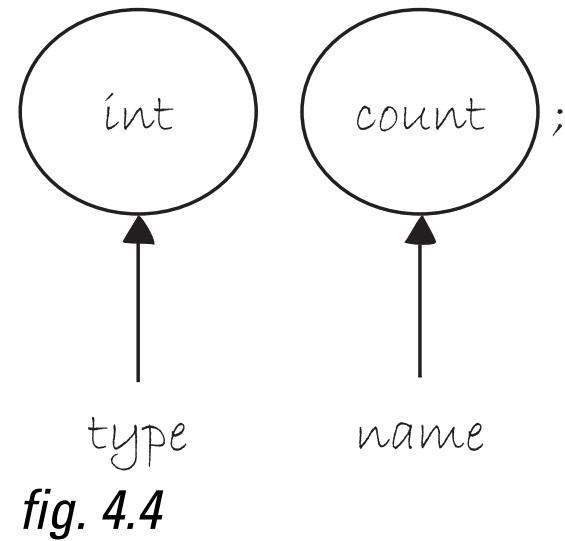
- Oltre a memorizzare un valore, una variabile permette di seguire il cambiamento del suo contenuto

ESERCIZIO

- Definire le variabili per il gioco
Pong

DICHIARAZIONE DI VARIABILI

- Una variabile può memorizzare valori primitivi o riferimenti ad array e oggetti
- Una variabile deve essere dichiarata prima di essere usata
- Per dichiararla, bisogna specificare
 - il tipo: quale dato dovrà contenere
 - Il nome: come chiamiamo la locazione che conserva il dato



TIPI DI VARIABILE

- boolean: true or false
- char: un carattere 'a', 'b', 'c', etc.
- byte: un numero piccolo, -128 to 127
- short: un numero un poco più grande, -32768 to 32767
- int: un numero intero, -2147483648 to 2147483647
- long: un numero estremamente grande
- float: numero con cifre decimali, such as 3.14159
- double: numero con tante cifre decimali

ASSEGNAMENTO DI UN VALORE

```
int count;  
count = 50;
```

Declare and initialize a variable in two lines of code.

```
int count = 50;
```

Declare and initialize a variable in one lines of code.

ESEMPIO

Example 4-1: Variable declaration and initialization examples

```
int count = 0;           // Declare an int named count, assigned the value 0
char letter = 'a';       // Declare a char named letter, assigned the value 'a'
double d = 132.32;       // Declare a double named d, assigned the value 132.32
boolean happy = false;   // Declare a boolean named happy, assigned the value false
float x = 4.0;           // Declare a float named x, assigned the value 4.0
float y;                 // Declare a float named y (no assignment)
y = x + 5.2;             // Assign the value of x plus 5.2 to the previously declared y
float z = x*y + 15.0;    // Declare a variable named z, assign it the value which
                        // is x times y plus 15.0.
```

USO DI VARIABILI

```
// tra poco aggiungeremo le nostre  
variabili  
  
void setup(){  
    size(200,200);  
}  
  
void draw(){  
    background(255);  
    stroke(0);  
    fill(175);  
    ellipse(100,100,50,50);  
}
```

USO DI VARIABILI

```
// tra poco aggiungeremo le nostre  
variabili  
  
void setup(){  
    size(200,200);  
}  
  
void draw(){  
    background(255);  
    stroke(0);  
    fill(175);  
    ellipse(mouseX,mouseY,50,50);  
}
```

USO DI VARIABILI

```
// dichiariamo le nostre variabili
int circleX = 100;
int circleY = 100;

void setup(){
    size(200,200);
}

void draw(){
    background(255);
    stroke(0);
    fill(175);
    ellipse(circleX,circleY,50,50);
}
```

ASSEGNAMENTO DI UN NUOVO VALORE

```
// dichiariamo le nostre variabili  
int circleX = 0;  
int circleY = 100;  
  
void setup(){  
    size(200,200);  
}  
  
void draw(){  
    background(255);  
    stroke(0);  
    fill(175);  
    ellipse(circleX,circleY,50,50);  
  
    circleX = circleX + 1;  
}
```

1. Remember `circleX = 0` and `circleY = 100`

2. Run `setup()`. Open a window 200×200 →



3. Run `draw()`.

- Draw circle at $(circleX, circleY) \rightarrow (0,100)$



$$circleX = 0 + 1 = 1$$

4. Run `draw()`

- Draw circle at $(circleX, circleY) \rightarrow (1,100)$



- Add one to `circleX`

$$circleX = 1 + 1 = 2$$

5. Run `draw()`

- Draw circle at $(circleX, circleY) \rightarrow (2,100)$



- Add one to `circleX`

$$circleX = 2 + 1 = 3$$

6. And so on and so forth!

fig. 4.5

ESERCIZIO

- Modifica l'esempio precedente per fare in modo che il cerchio cresca di dimensione

ESERCIZIO

Exercise 4-4

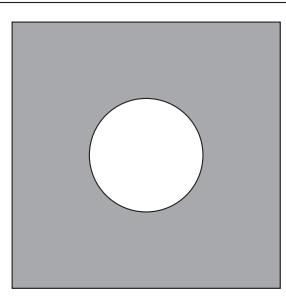
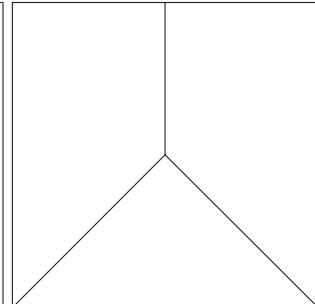
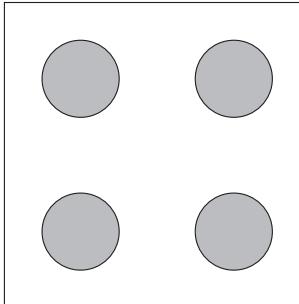


Step 1: Write code that draws the following screenshots with hard-coded values. (Feel free to use colors instead of grayscale.)

Step 2: Replace all of the hard-coded numbers with variables.

Step 3: Write assignment operations in `draw()` that change the value of the variables.

For example, “`variable1 = variable1 + 2;`”. Try different expressions and see what happens!



VARIABILI DI SISTEMA

- Processing crea delle variabili prima dell'inizio della esecuzione
- Abbiamo già visto un esempio con mouseX e mouseY
- Ecco un elenco di variabili utili:
 - width: la larghezza della finestra
 - height: la altezza della finestra
 - frameCount: il numero di refresh della finestra dall'inizio dell'esecuzione
 - frameRate: numero di frame disegnati al secondo
 - screen.width: larghezza dello schermo
 - screen.height: altezza dello schermo
 - key: codice dell'ultimo tasto premuto
 - keyPressed: valore boolean, true se viene premuto un tasto

ESEMPIO

Example 4-5: Using system variables

```
void setup() {  
    size(200,200);  
    frameRate(30);  
}  
  
void draw() {  
    background(100);  
    stroke(255);  
    fill(frameCount/2);  fill(frameCount/2);  
    rectMode(CENTER);  
    rect(width/2,height/2,mouseX+10,mouseY+10);  
}  
  
void keyPressed() {  
    println(key);  
}
```

The rectangle will always be in the middle of the window if it is located at (width/2, height/2).

frameCount is used to color a rectangle.



NUMERI CASUALI

NUMERI RANDOM

- La funzione random() è una funzione speciale che ritorna un valore casuale
- Rispetto alle altre funzioni viste finora (ellipse, line, point), questa risponde con un valore numerico
- La funzione ha bisogno di due numeri: viene ritornato un numero compreso tra i due
- La funzione ritorna un numero con la virgola: float
 - `float w = random(1,00);
rect(100,100,w,50);`

RANDOM ELLIPSES

Example 4-7: Filling variables with random values

```
float r;
float g;
float b;
float a;

float diam;
float x;
float y;

void setup() {
    size(200,200);
    background(0);
    smooth();
}

void draw() {
    // Fill all variables with random values
    r = random(255);
    g = random(255);
    b = random(255);
    a = random(255);
    diam = random(20);
    x = random(width);
    Y = random(height);

    // Use values to draw an ellipse
    noStroke();
    fill(r,g,b,a);
    ellipse(x,y,diam,diam);
}
```

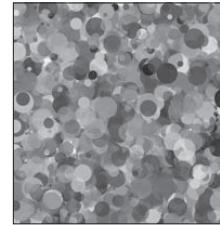


fig. 4.8

Each time through **draw()**, new random numbers are picked for a new ellipse.

VARIABLE ZOOG





ISTRUZIONI CONDIZIONALI

VALORI BOOLEANI

- Il tipo boolean prevede solo due possibili valori: true o false
- Sono prodotti come risultato di operazioni di confronto

```
println(3 > 2)
```

// → true

```
println(3 < 2)
```

// → false

- Si possono confrontare anche le stringhe

```
println("Aardvark" < "Zoroaster")
```

// → true

- Il confronto tra stringhe segue l'ordinamento dato dalla codifica Unicode, che assegna un valore numerico ad ogni carattere

BOOLEANI: OPERATORI DI CONFRONTO

- < (minore)
- > (maggiore)
- <= (minore o uguale)
- >= (maggiore o guale)
- == (uguale)
- != diverso
- Esempio

```
println("Itchy" != "Scratchy")  
// → true
```

BOOLEANI: OPERATORI LOGICI (1)

- Java supporta tre operatori logici: `&&` (and), `||` (or) e `!` (not)
- I tre operatori sono alla base della Algebra di Boole
 - Permette di usare la logica proposizionale per ragionare su fatti che possano verificarsi o meno
- L'operatore `&&` restituisce `true` se entrambi i suoi argomenti sono `true`

```
println(true && false)
```

// → `false`

```
println(true && true)
```

// → `true`

```
println(false&& false)
```

// → `false`

```
println(false&& true)
```

// → `false`

BOOLEANI: OPERATORI LOGICI (2)

- L'operatore `||` restituisce `true` se uno dei suoi argomenti è `true`

```
println(false || true)
```

// → true

```
println(false || false)
```

// → false

```
println(true || true)
```

// → true

```
println(true || false)
```

// → true

BOOLEANI: OPERATORI LOGICI (3)

- L'operatore ! restituisce il valore inverso del suo argomento

```
println(!false)
```

// → true

```
println(!true)
```

// → false

- L'operatore condizionale (cond? val1 : val2) è un operatore ternario. Se il valore a sinistra del simbolo ? è vero ritorna il primo valore, altrimenti il secondo

```
println(true ? 1 : 2);
```

// → 1

```
println(false ? 1 : 2);
```

// → 2

CONTROLLO DEL FLUSSO

- Quando un programma è formato da più istruzioni, queste vengono eseguite in ordine dall'alto verso il basso
- Il seguente programma è composto da due istruzioni



```
int theNumber = int(random(10, 100));  
println("Your number is " + theNumber);
```

ISTRUZIONI CONDIZIONALI

- Tutti i programmi vengono eseguiti in sequenza
- Queste istruzioni di controllo consentono di scegliere due possibili percorsi, sulla base di un valore booleano
- Sintassi:

```
if (espressione)
    istruzione1
else
    istruzione2
```
- Espressione è una espressione booleana
- Istruzione1 rappresenta il ramo eseguito se la valutazione ritorna true
- Istruzione2 rappresenta il ramo eseguito se la valutazione ritorna false



ESEMPIO

```
if(mouseX < width / 2){  
    fill(255);  
    rect(0,0,wdth/2,height);  
}
```

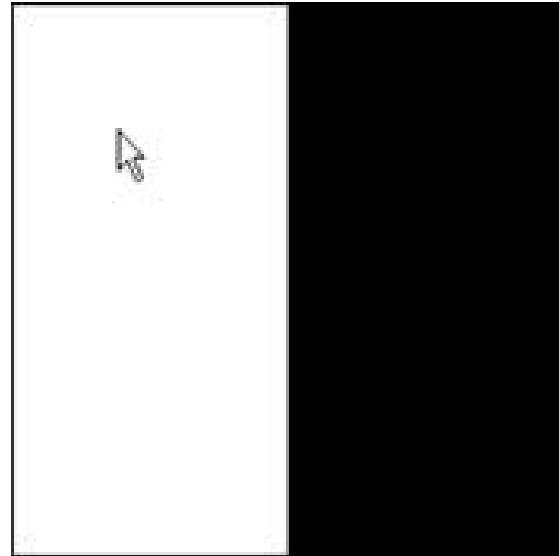


fig. 5.1

ESEMPIO

```
if(mouseX < width/2){  
    background(255);  
}else{  
    background(0);  
}
```

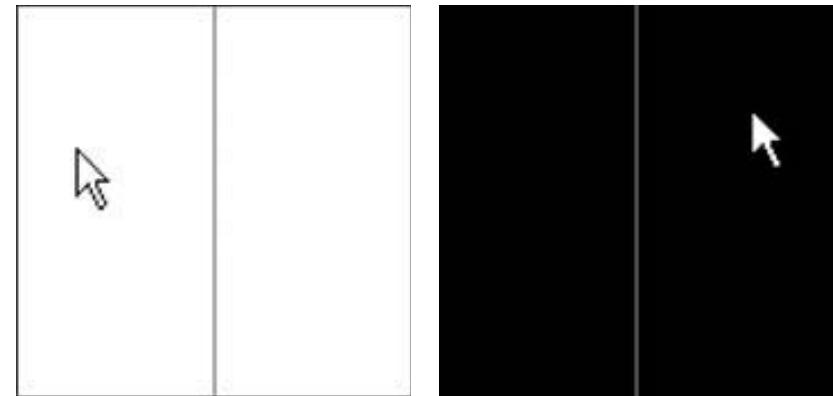


fig. 5.2

CONDIZIONI MULTIPLE

```
if (boolean expression #1) {  
// code to execute if boolean  
expression #1 is true  
} else if (boolean expression #2) {  
// code to execute if boolean  
expression #2 is true  
} else if (boolean expression #n) {  
// code to execute if boolean  
expression #n is true  
} else {  
// code to execute if none of the  
above  
// boolean expressions are true  
}
```

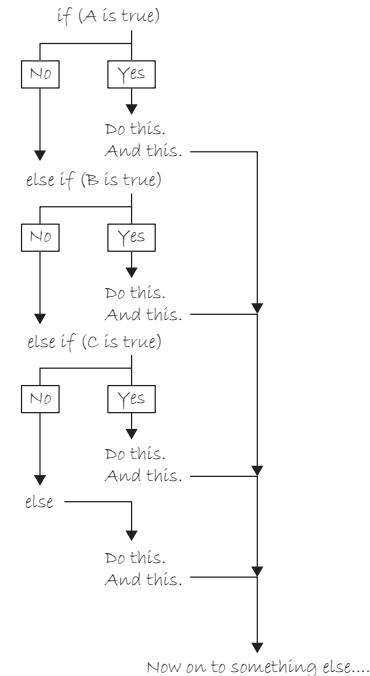


fig. 5.3

ESEMPIO

```
if (mouseX < width/3) {  
    background(255);  
} else if (mouseX < 2*width/3) {  
    background(127);  
} else {  
    background(0);  
}
```

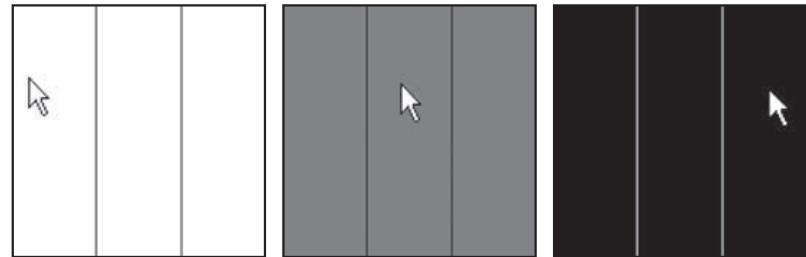


fig. 5.4

SKETCH CONDIZIONALI

Example 5-1: Conditionals

```
float r = 150;  
float g = 0;  
float b = 0;
```

1. Variables.

```
void setup() {  
    size(200, 200);  
}
```

```
void draw() {  
    background(r,g,b);  
    stroke(255);  
    line(width/2,0,width/2,height);
```

2. Draw stuff.

```
if (mouseX > width/2) {  
    r = r + 1;  
} else {  
    r = r - 1;  
}
```

3. "If the mouse is on the right side of the screen" is equivalent to "if **mouseX** is greater than width divided by 2."

```
if (r > 255) {  
    r = 255;  
} else if (r < 0) {  
    r = 0;  
}
```

4. If *r* is greater than 255, set it to 255.
If *r* is less than 0, set it to 0.

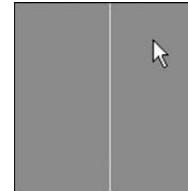


fig. 5.5

FUNZIONE constrain()

```
if (r > 255) {  
    r = 255;  
} else if (r < 0) {  
    r = 0;  
}  
  
r = constrain(r, 0, 255);
```

Constrain with an “if” statement.

Constrain with the ***constrain()*** function.



VALORI, TIPI E OPERATORI

RAPPRESENTAZIONE DELL'INFORMAZIONE

- Il calcolatore può leggere, modificare e gestire solo **dati**
- I dati sono rappresentati come **sequenze di bit**, ovvero sequenze di 1 e 0
- I valori 1 e 0 sono rappresentati dal calcolatore come tensioni elettriche, rispettivamente alta e bassa
- Rappresentazione di numeri basata su base 2 invece che su base 10
- Ad esempio, il numero 13:
 - 0 0 0 0 1 1 0 1
 - 128 64 32 16 8 4 2 1



VALORI E TIPI

- In un calcolatore ci sono sequenze lunghissime di bit
- Sequenze di bit sono organizzate in segmenti logici, chiamati valori
- Diversi valori possono avere diversi ruoli
 - Numeri, stringhe, boolean, oggetti, funzioni, e undefined
- Ogni valore viene creato nel momento in cui viene invocato
- Ogni valore viene conservato nella memoria interna del calcolatore, fino a quando c'è spazio disponibile
- I valori che non vengono utilizzati vengono distrutti e il loro spazio può essere riutilizzato

NUMERI

- I numeri sono rappresentati da sequenze di cifre
 - Ad esempio il numero 13
- Nel momento in cui viene incontrato il numero 13 in un programma, viene creato uno spazio in memoria in cui sono conservati i bit che lo rappresentano
- Ogni numero in Java viene rappresentato con 64bit
 - Possiamo rappresentare 2^{64} valori differenti per singolo valore
 - Dobbiamo rappresentare anche il segno (negativo o positivo) e la posizione della cifra decimale
 - Operazioni in virgola mobile perdono precisione quando rappresentati su uno spazio finito

NUMERI: NOTAZIONE

- Numero intero
 - 13
- Numero decimale
 - 9.81
- Notazione scientifica
 - $2.998\text{e}8 = 2.998 * 10^8 = 299,800,000$
- I numeri decimali vanno sempre trattati come approssimazioni del valore reale
 - Ad esempio: confronto del risultato di una divisione

OPERATORI ARITMETICI

- Gli operatori aritmetici prendono in input due numeri e producono in output un numero
- Gli operatori sono: *, /, %, +, -
- Gli operatori possono esser concatenati
 - $100 + 4 * 11$
- I simboli + e * sono operatori
- Gli operatori hanno una precedenza. In questo caso l'operatore di moltiplicazione viene eseguito prima dell'operatore +
- La lista di operatori precedenti è in ordine di precedenza decrescente
- Si può cambiare l'ordine di precedenza utilizzando le parentesi
 - $(100 + 4) * 11$

OPERATORI ARITMETICI: MODULO

- L'operatore di modulo (%) restituisce il resto della divisione intera di due numeri
- $X \% Y$ è il resto dell'operazione di divisione intera di X per Y
- Ad esempio:
 - $314 \% 100 = 14$
 - $144 \% 12 = 0$

STRINGHE

- Le stringhe sono utilizzate per rappresentare testi
- Sono definite tramite l'uso di virgolette singole o doppie
 - "Patch my boat with chewing gum"
 - 'Monkeys wave goodbye'
- Java riconosce qualunque carattere all'interno delle virgolette
- Alcuni caratteri richiedono qualche attenzione
 - Ad esempio, rappresentare il carattere di virgolette dentro la stringa

STRINGHE: ESCAPING

- Per includere caratteri speciali è necessario utilizzare un operatore di escaping: \
- Quando il carattere \ viene trovato all'interno di una stringa tra virgolette, esso indica che il carattere successivo ha un significato speciale
 - Un carattere con virgolette preceduto da \ non indica la fine della stringa ma una parte di essa
 - ‘Egli disse \'Ciao\' appena lo vide’
 - Il carattere n preceduto da \ viene interpretato come un rimando a capo
 - "This is the first line\nAnd this is the second"
 - Contiene il seguente testo
 - This is the first line
And this is the second
 - Per rappresentare il carattere \ dentro una stringa si usano due simboli \ in sequenza
 - "A newline character is written like \"\\n\"."

STRINGHE: OPERATORI

- Le stringhe non prevedono gli operatori di divisione, moltiplicazione o sottrazione.
- E' possibile utilizzare l'operatore `+`, con il significato di concatenazione
 - `println("con" + "cat" + "e" + "nate");`
 - // “concatenate”

VALORI INDEFINITI

- Ci sono due valori speciali che vengono usati per denotare l'assenza di valori significativi
 - `null`
 - `undefined`

PRECEDENZA EGLI OPERATORI LOGICI

- Gli operatori logici `&&` e `||` valutano o convertono in tipo booleano l'operando a sinistra per determinare il risultato del confronto
- In base al risultato di questa prima valutazione, il comportamento può variare
- Questa proprietà può essere utile per assegnare un valore di default in caso uno sia mancante

```
println(null || "user")
// → user
println("Karl" || "user")
// → Karl
```

- L'espressione a destra viene valutata solo se necessario

```
true || x
false && x
```

SOMMARIO

- Linguaggio Java
- Definizione di valori e tipi
- Operatori aritmetici: +, -, *, /, %
- Concatenazione di stringhe: +
- Confronto: ==, !=, ===, !==, <, >, <=, >=
- Operatori logici: &&, || , !
- Operatori unari: -, !
- Operatori ternari: (?:)



ITERAZIONI E CICLI

ITERAZIONI

- Le istruzioni iterative permettono di ripetere l'esecuzione di parti del programma una o più volte
- Il numero di ripetizioni può essere fissato (iterazione determinata)
- Oppure può dipendere da una condizione (iterazione indeterminata)

ISTRUZIONE while

- L'espressione `while` è uno dei costrutti di iterazione di Javascript
- Sintassi
 - `while` (espressione)
 istruzione
- espressione è una espressione booleana (detta **guardia**): se viene valutata a true allora viene eseguita l'istruzione interna (detta **corpo**) e si ripete il ciclo; se l'espressione booleana ritorna false, la ripetizione termina
- Se espressione è false fin dall'inizio, l'istruzione non viene mai eseguita

BLOCCHI DI ISTRUZIONI

- Nei costrutti che abbiamo visto finora, le espressioni booleane hanno controllato l'esecuzione di singole istruzioni
- Nel caso in cui debbano essere eseguite più istruzioni in sequenza possiamo definire un **blocco**, ovvero una sequenza di istruzioni rinchiusa tra parentesi graffe (`{ e }`)
- Esempio: scrivere un programma che stampi i numeri pari minori o uguali a 12:

```
var number = 0;  
while (number <= 12) {  
    console.log(number);  
    number = number + 2;  
}  
// → 0  
// → 2  
// ... etcetera
```

ESEMPIO

- Scrivere un programma che calcoli il valore di 2^{10}

```
var result = 1;  
var counter = 0;  
while (counter < 10) {  
    result = result * 2;  
    counter = counter + 1;  
}  
console.log(result);  
// → 1024
```

ISTRUZIONE for

- Alcuni casi di cicli visti finora utilizzano una variabile di controllo per contare il numero di iterazioni eseguite
- Questo tipo di soluzione si presenta molto spesso in fase di programmazione
- Il linguaggio prevede un costrutto apposito per delle iterazioni determinate
- Sintassi:

```
for (expr1; expr2; expr3)
    istruzione
```
- Dove expr1 serve a inizializzare la variabile di controllo; expr2 è la guardia di fine ciclo; expr3 aggiorna la variabile di controllo; istruzione è il corpo del ciclo

ESEMPIO

- Scrivere un programma che calcoli il valore di 2^{10}

```
var result = 1;  
for (var counter = 0; counter < 10; counter = counter + 1)  
    result = result * 2;  
println(result);  
// → 1024
```

INTERRUZIONE DI UN CICLO

- Abbiamo visto che un ciclo viene interrotto quando l'espressione a guardia del ciclo diventa `false`
- E' possibile forzare l'interruzione di un ciclo con l'istruzione `break`
- Esempio: trovare il primo intero che sia maggiore di 20 e divisibile per 7

```
for (var current = 20; ; current++) {  
    if (current % 7 == 0)  
        break;  
}  
console.log(current);  
// → 21
```
- È possibile interrompere l'esecuzione del corpo del ciclo e saltare all'iterazione successiva con l'istruzione `continue`

INDENTAZIONE DEL CODICE

- In molti esempi potete notare la presenza di uno o più spazi davanti ad alcune istruzioni
- Questi spazi servono ad indentare blocchi di codice
- Questa indentazione serve a enfatizzare la struttura del codice, evidenziando i blocchi contenuti all'interno delle istruzioni condizionali o ai cicli
- L'indentazione è solo una convenzione. Non è necessaria per la corretta esecuzione del programma

AGGIORNAMENTO SUCCINTO DI VARIABILI

- Abbiamo visto diversi esempi di aggiornamento di varibili all'interno dei cicli
- Ad esempio alcune guardie avevano la seguente sintassi
`counter = counter + 1`
- Per evitare di scrivere più volte la stessa variabile si può usare la seguente forma compatta
`counter += 1`
- La stessa forma si può usare per altri operatori e operandi: *=2, -=1
- Per incrementi o decrementi di singole unità si possono anche usare le espressioni `counter++` e `counter--`

ISTRUZIONE SWITCH

- In alcuni casi è necessario confrontare una variabile con diversi possibili valori

```
if (variable == "value1") action1();
else if (variable == "value2") action2();
else if (variable == "value3") action3();
else defaultAction();
```

- Per semplificare questo tipo di confronti si può utilizzare il costrutto switch

```
switch (prompt("What is the weather like?")) {
  case "rainy":
    console.log("Remember to bring an umbrella.");
    break;
  case "sunny":
    console.log("Dress lightly.");
  case "cloudy":
    console.log("Go outside.");
    break;
  default:
    console.log("Unknown weather type!");
    break;
}
```

- A volte questo tipo di istruzione può generare errori. Una catena di istruzioni if può fornire una soluzione più facile da gestire

CAPITALIZATION

- I nomi delle variabili non possono contenere spazi
- In alcuni casi è utile chiamare una variabile con un nome composto
- Ci sono alcune convenzioni per risolvere questo problema

fuzzylittleturtle

fuzzy_little_turtle

FuzzyLittleTurtle

fuzzyLittleTurtle

COMMENTI AL CODICE

- Sebbene il linguaggio sia di alto livello, spesso il solo codice non è sufficiente per rendere un programma comprensibile ad un essere umano
- Spesso si utilizzano delle annotazioni e commenti al codice per spiegare alcune parti di esso
- Il **commento** è una parte del programma ma viene completamente ignorato dall'interprete che lo esegue
- Si possono scrivere due tipi di commenti: su singola riga o a blocchi
- I commenti su singola riga iniziano con due caratteri // e finiscono con la fine della riga

```
var accountBalance = calculateBalance(account);
// It's a green hollow where a river sings
accountBalance.adjust();
// Madly catching white tatters in the grass.
```

COMMENTI

- Per definire un commento su più righe, si racchiude il testo da commentare tra i due simboli /* e */

```
/*
```

```
I first found this number scrawled on the back of one of  
my notebooks a few years ago. Since then, it has often  
dropped by, showing up in phone numbers and the serial  
numbers of products that I've bought. It obviously likes  
me, so I've decided to keep it.
```

```
*/
```

```
var myNumber = 11213;
```