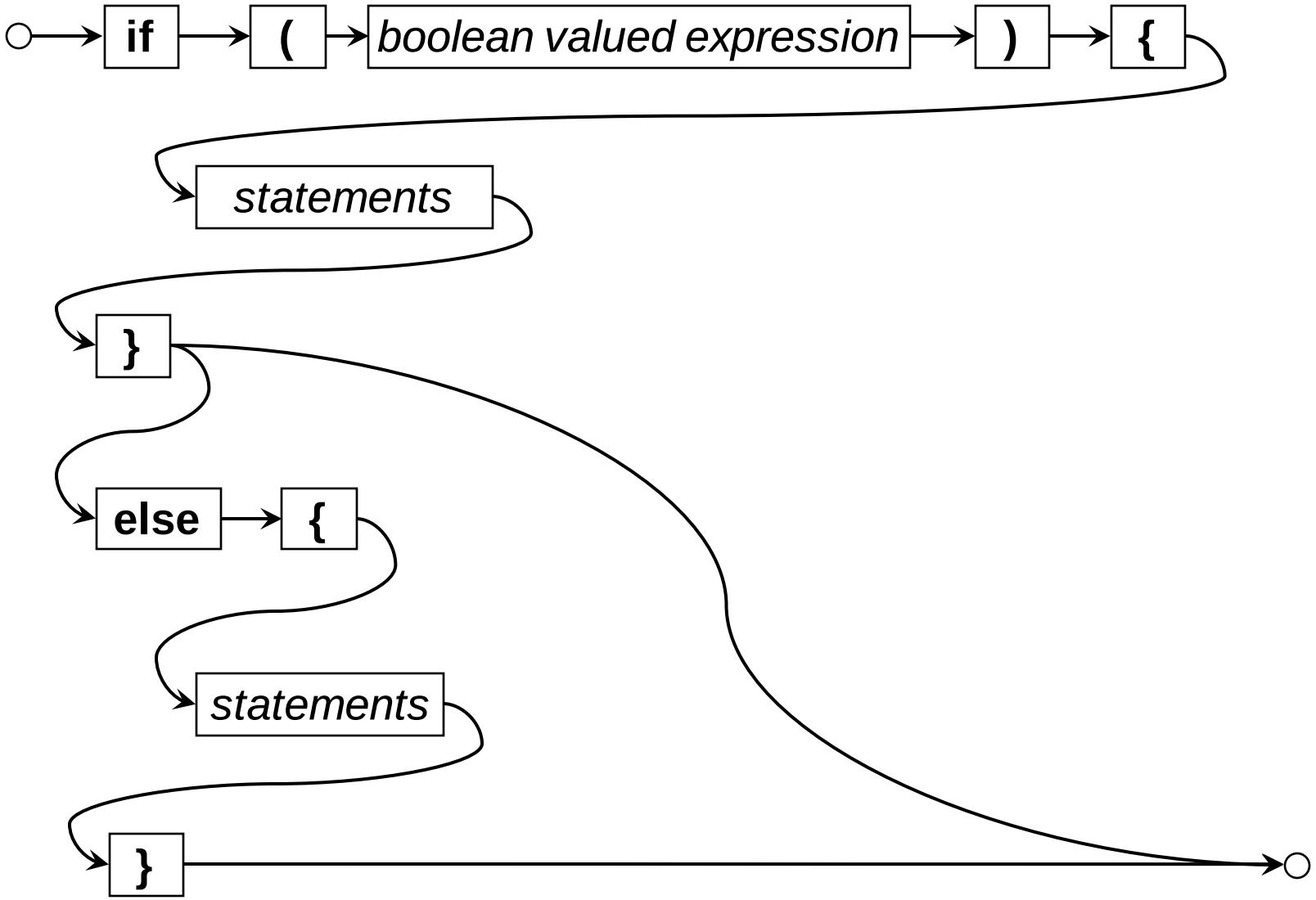
Engineering Technology (ENGR 101)

Conditional Statements and Iterations

Admin

- Lab 1 has been released
  - Due date is 17 March, 19:00 (Xiamen Time)
  - You must present your projects to co-teachers or tutors in the lab.
  - You can either use a real Arduino or Tinkercad simulator to test your codes.

if ... *vs* if ... else ...



Using else-if statement

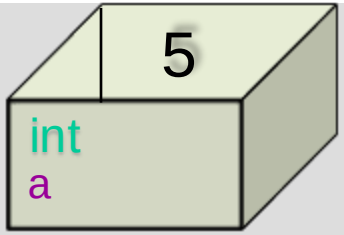
- Can put another **if** statement in the **else** part:

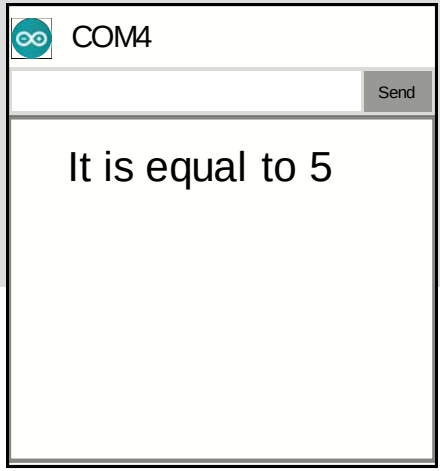
```
if ( 〈condition1〉 ) {
  〈actions to perform if condition1 is true〉
    :
}
else if ( 〈condition2〉 ) {
  〈actions to perform if condition 2 is true (but not condition 1)〉
    :
}
else if ( 〈condition3〉 ) {
  〈actions to perform if condition 3 is true (but not conditions 1, 2)〉
    :
}
else {
  〈actions to perform if other conditions are false〉
    :
}
```

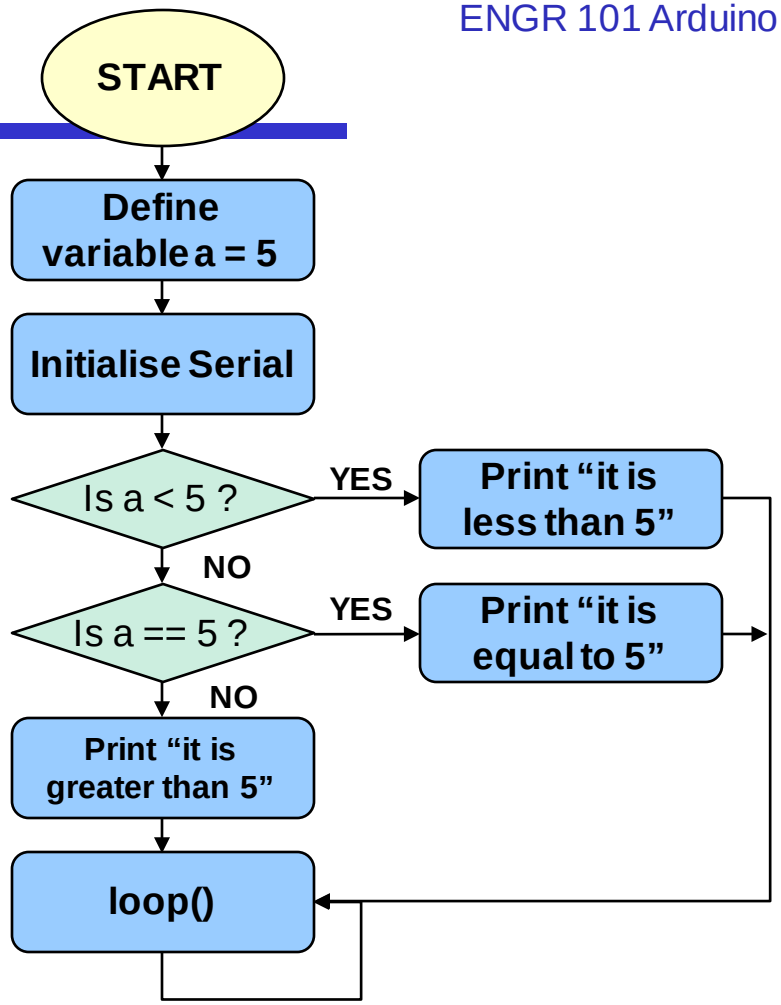
Example: else-if statement

```
int a = 5;
void setup() {
   Serial.begin(9600);
   if( a < 5 ){
      Serial.println("It is less than 5");
   }
   else if( a == 5){
      Serial.println("It is equal to 5");
   }
   else{
      Serial.println("It is greater than 5");
   }
}
void loop() {

}
```







Boolean expressions

What can go in the condition of an **if** statement?

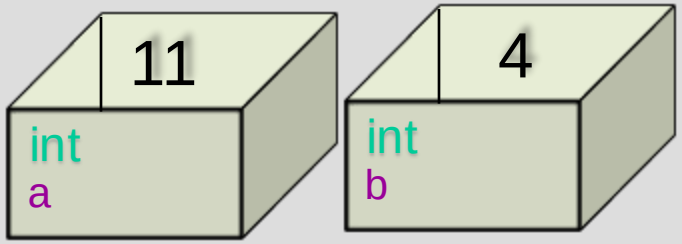
- A Boolean value – a value that is either true or false.
- Boolean expressions:
  - constant values: true, false
  - numeric comparisons: (x > 0) (day <= 7), (x == y), (day != 7)
  - logical operators: !, &&, || (not, and, or)

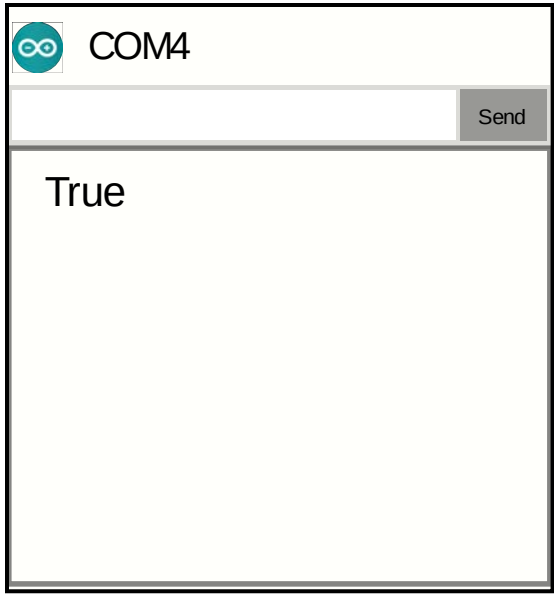
    (x > 0 && x < 7)

Relational Operators: < > <= >= != ==

```
int a = 11;
int b = 4;
void setup() {
   Serial.begin(9600);
   if( (a != 5) && (b > 3) ){
      Serial.print(“Ture!");
   }

}
void loop() {
}
```





Compound Boolean expressions: operators

Using logical operators:

Not: **!** eg (! (x > 0))

And: **&&** eg (x > 0 && x < 7 && y<10)

Evaluates each conjunct in turn.
If any conjunct false, then value of whole expression is false
If all conjuncts true, then value of whole expression is true

Or: **||** eg (x>0 || y<10)

Evaluates each disjunct in turn.
If any disjunct true, then value of whole expression is true
If all disjuncts false, then value of whole expression is false

Can combine into complicated expressions:

(y<10 || (x> 8 && y > 5000))

safest to use lots of (…)

Traps with Boolean expressions

- When combining with && and ||, which binds tighter?

  if (x > 5 && y <= z || day == 0) { ....

  - Use (and) whenever you are not sure!

    if ((x > 5 && y <= z) || day == 0) { …

    if (x > 5 && (y <= z || day == 0)) { …

- The not operator ! goes in front of expressions:

  - **if (!(x > 5) && y <= z) {** … **NOT** **if ((x !> 5 && y <= z)**

Writing Boolean expressions

Mostly, boolean expressions are straightforward,
There are just a few traps:

- == is the "equals" operator for simple values,
  = is assignment

  (age == 15) *vs* ~~(age = 15);~~
  - But only use == for numbers (or characters, or references)

Repetition / Iteration

Doing some action repeatedly:

- “Polish each of the cups on the shelf”
- “Put every chair on top of its desk”
- “Give a ticket to everyone who passes you”
- “Keep running around the track until 6pm”
- “Practice the music until you can play it perfectly”

Two patterns:

- Do something to each thing in a collection
- Do something until some condition changes

Repetion/Iteration

Several different ways of specifying repetition.

- Counted **For** statement: Do something to each number from …..

```
for ( int num = ⟨start⟩;  num <= ⟨end⟩ ;  num = num + ⟨increment⟩ ) {
    do something with num
}
```

- **While** statement: Repeat some action while some condition is still true

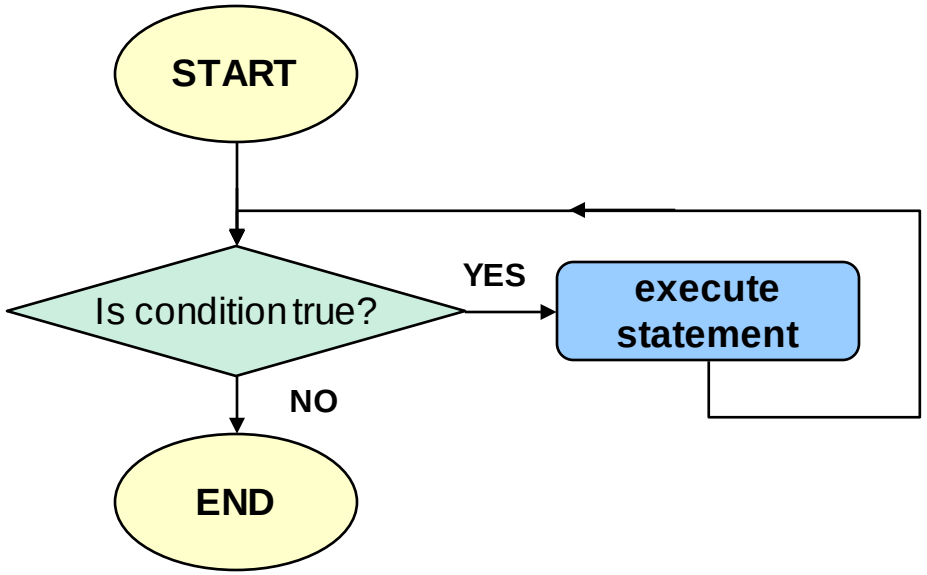
```
while (condition-to-do-it-again ) {
    actions to perform each time round
}
```

Iteration: while-loop statement

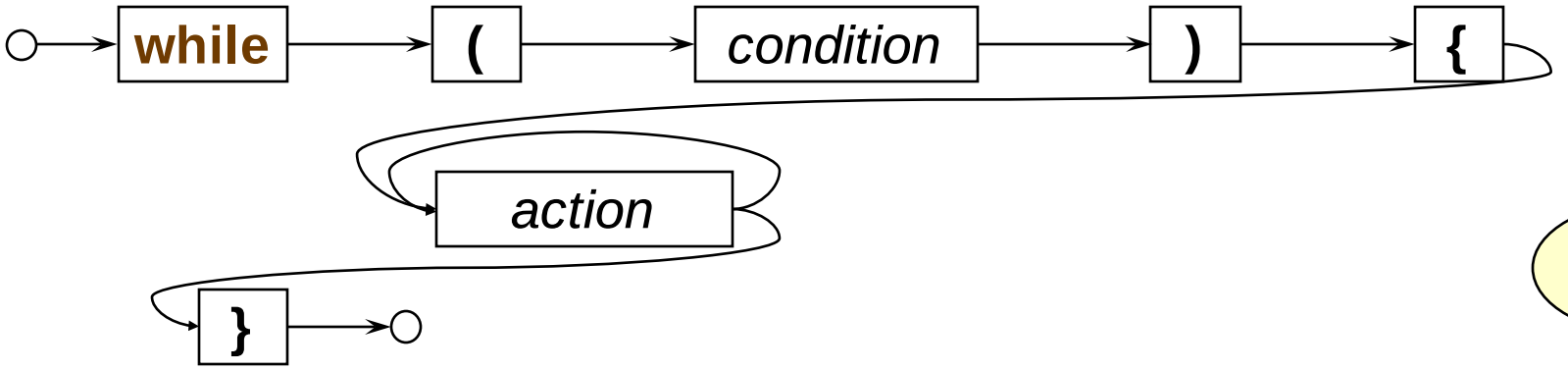
- If `condition` evaluates to true:
  - `statement` is executed
  - `condition` is re-evaluated again
- Cycle continues until `condition` evaluates to false

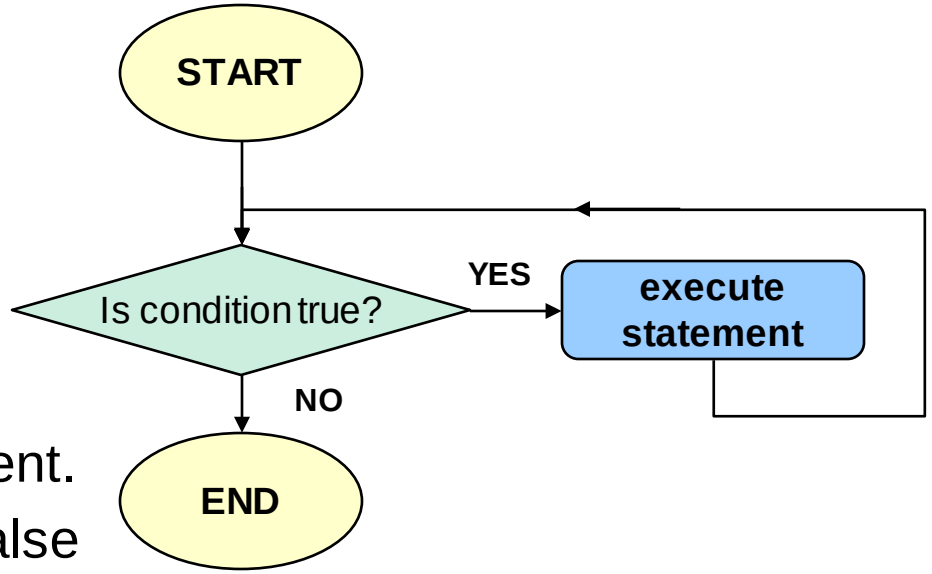
```
while (condition-to-do-it-again )  {
     actions to perform each time round
}
```

Similar structure to the **if** statement



While statement





- Meaning:

  Repeatedly
  - If the condition is still true, do the actions another time
  - If the condition is false, stop and go on to the next statement.
    - Note: don't do actions at all if the condition is initially false

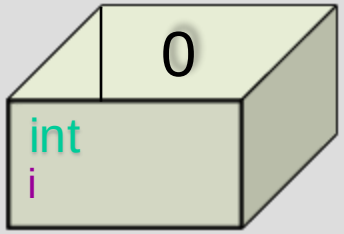
- Similar to **if**, but NOT THE SAME!
  - keeps repeating the actions,
    - as long as the condition is still true each time round
  - no **else** — just skips to next statement when condition is false

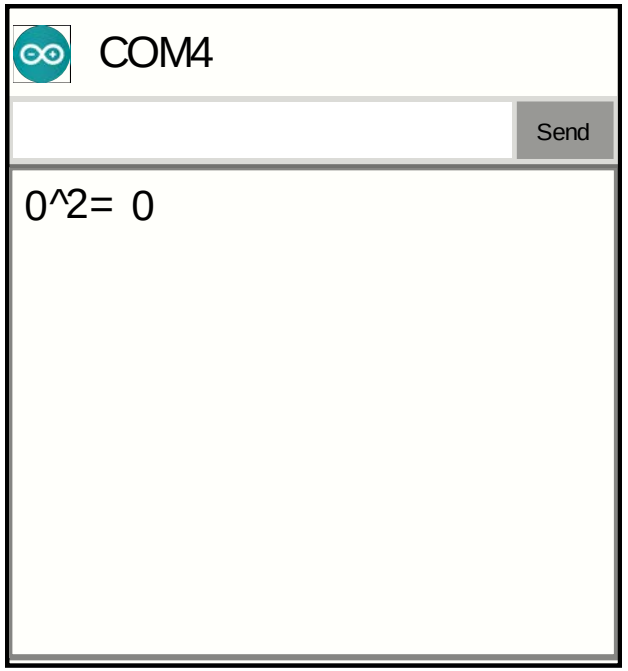
Iteration: while-loop statement

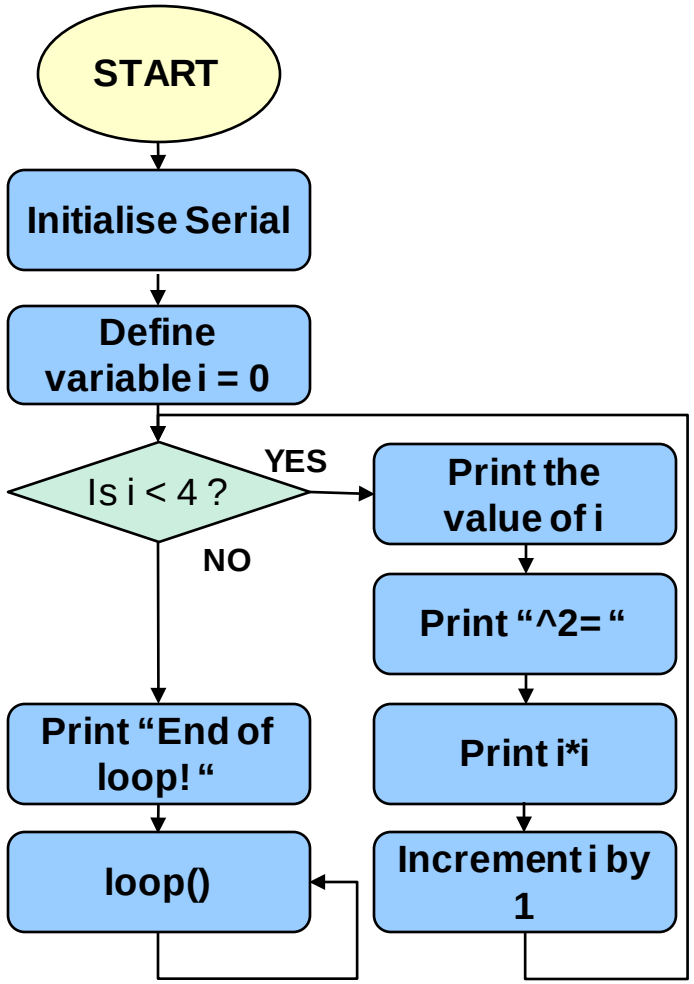
```
while (expression)
    statement
```

- If `expression` evaluates to true:
  - `statement` is executed
  - `expression` is re-evaluated again
- Cycle continues until `expression` evaluates to false

```
void setup() {
   Serial.begin(9600);
   int i = 0;
   while (i < 4) {
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i*i);
      i++; // or i = i + 1
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```





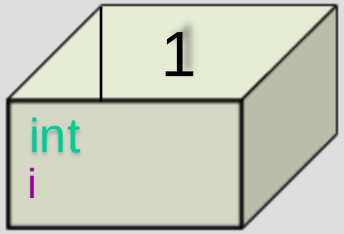


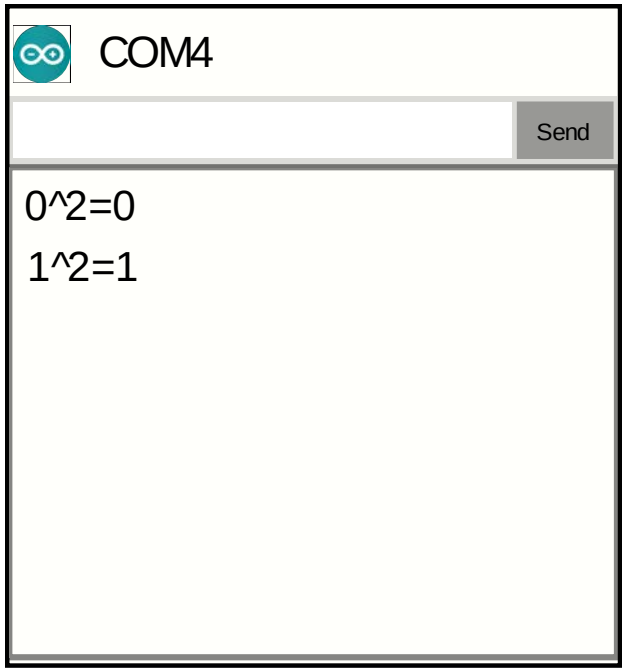
Iteration: while-loop statement

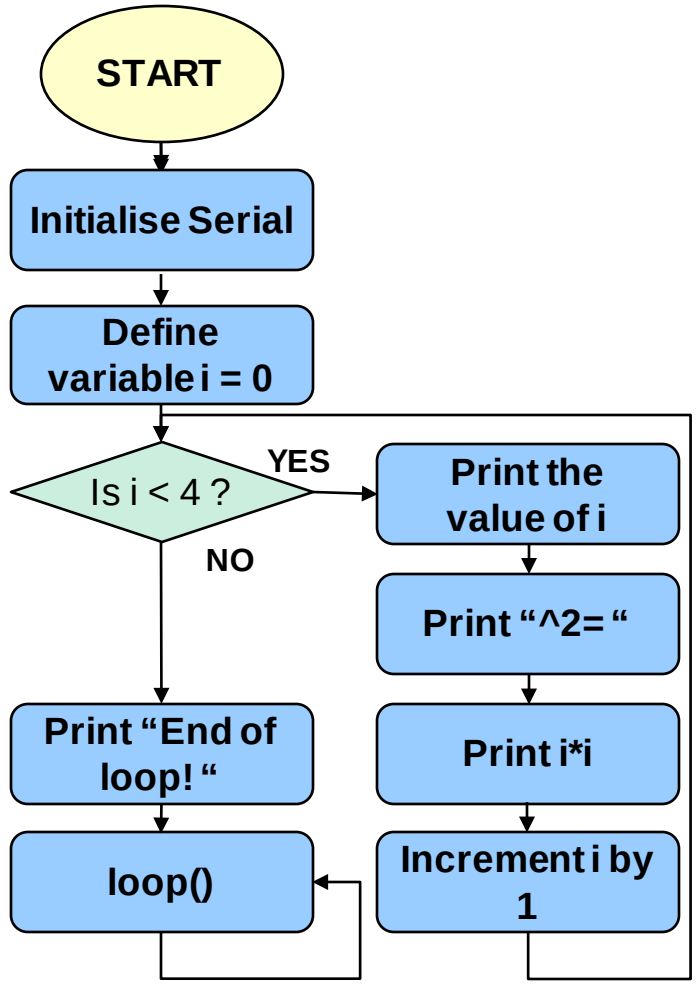
```
while (expression)
    statement
```

- If `expression` evaluates to true:
  - `statement` is executed
  - `expression` is re-evaluated again
- Cycle continues until `expression` evaluates to false

```
void setup() {
   Serial.begin(9600);
   int i = 0;
   while (i < 4) {
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i*i);
      i++; // or i = i + 1
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```





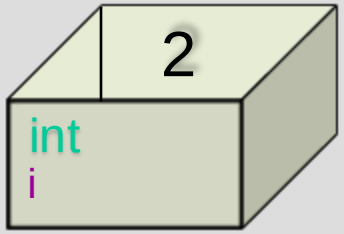


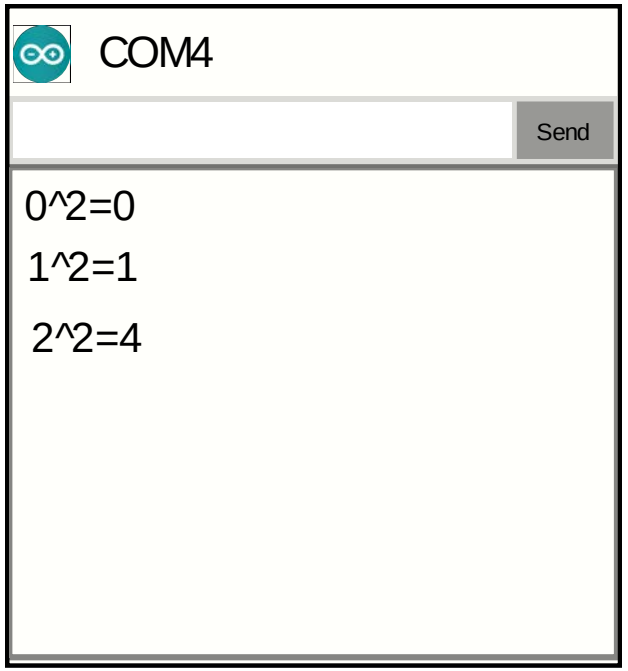
Iteration: while-loop statement

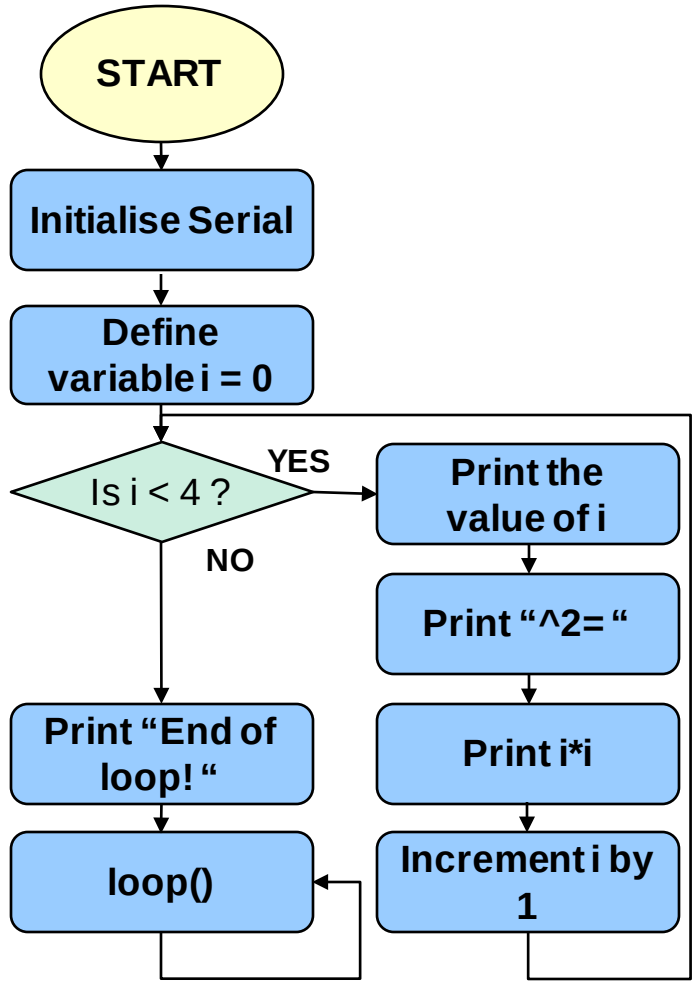
```
while (expression)
    statement
```

- If `expression` evaluates to true:
  - `statement` is executed
  - `expression` is re-evaluated again
- Cycle continues until `expression` evaluates to false

```
void setup() {
   Serial.begin(9600);
   int i = 0;
   while (i < 4) {
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i*i);
      i++; // or i = i + 1
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```





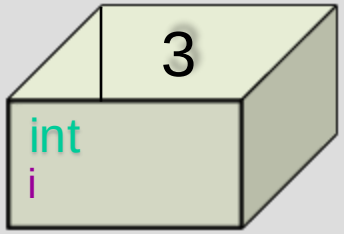


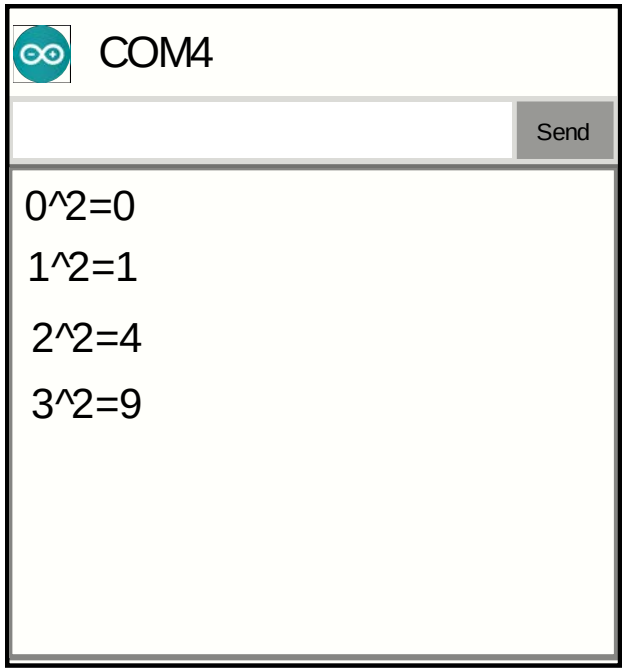
Iteration: while-loop statement

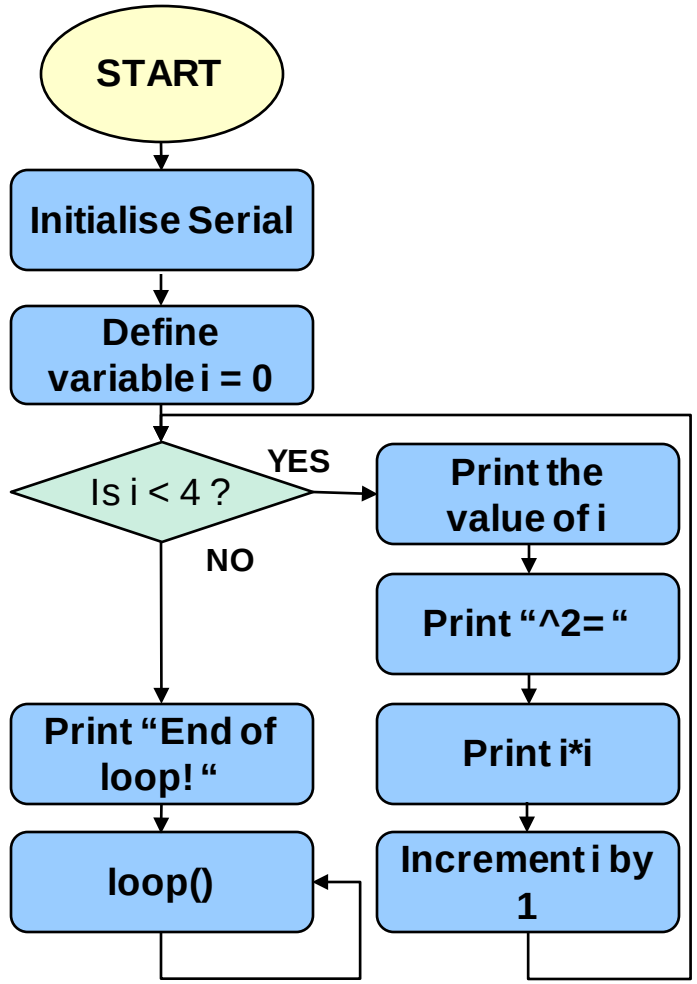
```
while (expression)
    statement
```

- If `expression` evaluates to true:
  - `statement` is executed
  - `expression` is re-evaluated again
- Cycle continues until `expression` evaluates to false

```
void setup() {
   Serial.begin(9600);
   int i = 0;
   while (i < 4) {
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i*i);
      i++; // or i = i + 1
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```





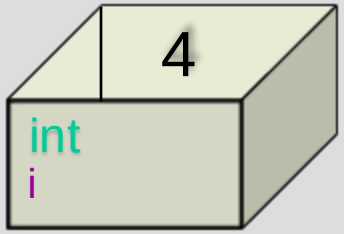


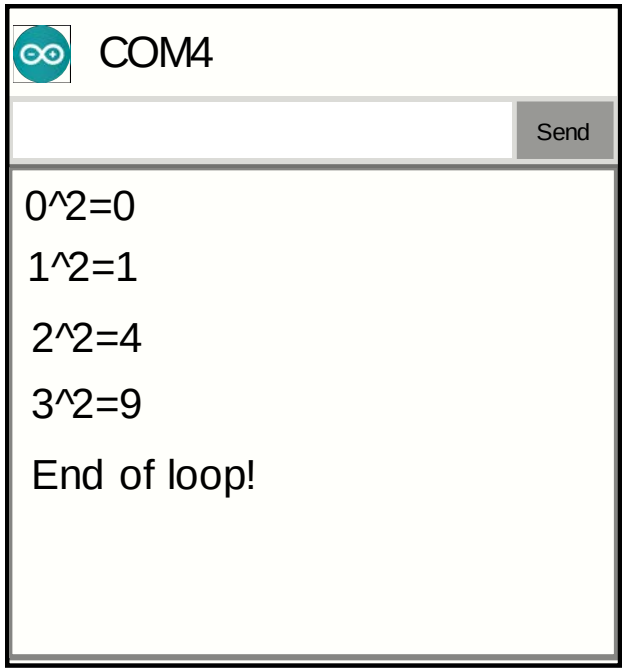
Iteration: while-loop statement

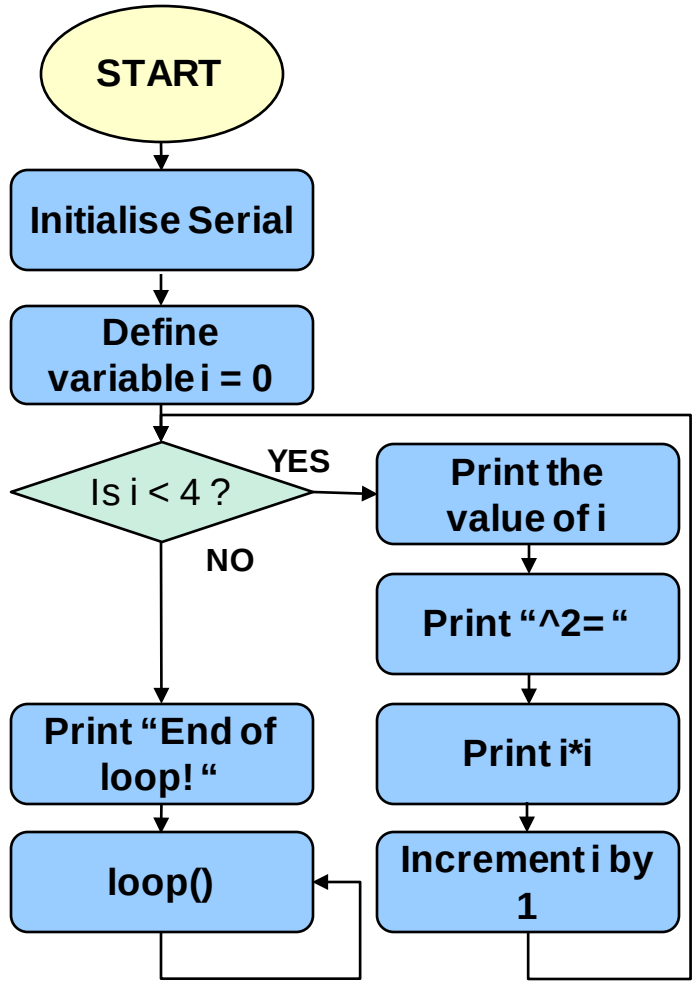
```
while (expression)
    statement
```

- If `expression` evaluates to true:
  - `statement` is executed
  - `expression` is re-evaluated again
- Cycle continues until `expression` evaluates to false

```
void setup() {
   Serial.begin(9600);
   int i = 0;
   while (i < 4) {
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i*i);
      i++; // or i = i + 1
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```





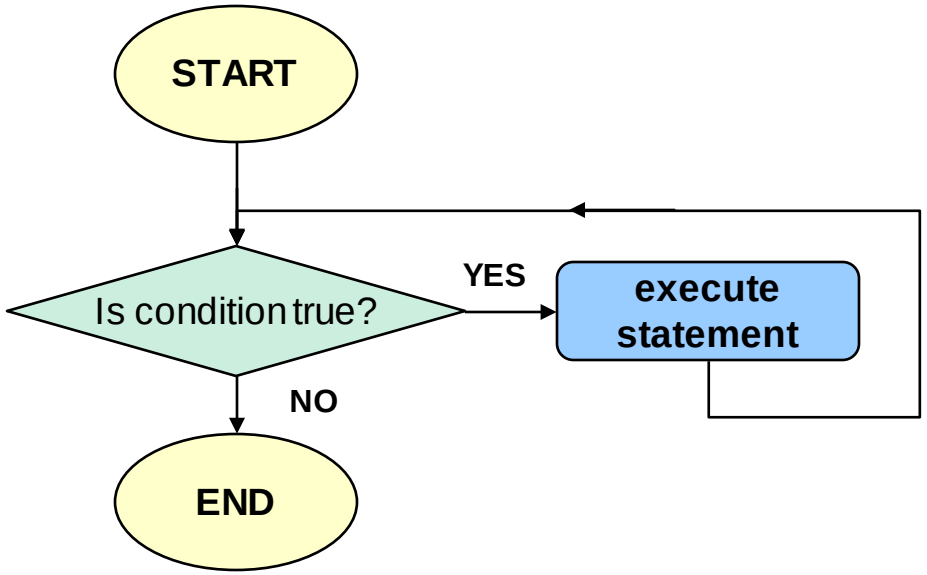


Iteration: while-loop statement

- while `condition` evaluates to true:
  - `statement` is executed
  - `condition` is re-evaluated again
- Cycle continues until `condition` evaluates to false

```
while (condition-to-do-it-again )  {
     actions to perform each time round
}
```

Similar structure to the **if** statement



While-loop statement

- Print a table of numbers and their squares:

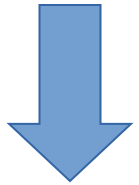
```
void setup(){
  int num = 1;                    Initialise
  while ( num < 4 ) {             Test
    Serial.println( (num*num));   Body
    num = num + 1;                Increment
  }
}
```

- Repetition with **while** generally involves
  - initialisation: get ready for the loop — Put before while loop
  - test: whether to repeat
  - body: what to repeat
  - “increment”: get ready for the next iteration — Put at end of actions.

Iteration: for-loop statement

- The expressions are optional
- $expr_1$ and $expr_3$ are usually assignments
- $expr_2$ is usually a relational expression
  - If $expr_2$ is missing, it is taken as permanently true

```
expr1;
while (expr2) {
    statement
    expr3;
}
```



```
for (expr1; expr2; expr3){
    statement
}
```

```
void setup() {
   Serial.begin(9600);
   for (int i = 0; i < 4; i++){
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i * i);
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```

Numeric For statement

For statement.

Most commonly used to step through a sequence of numbers

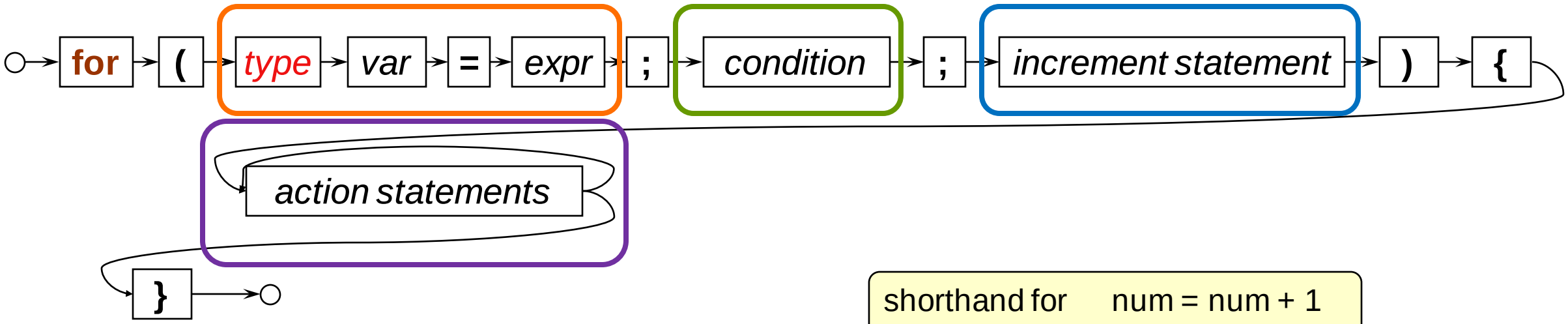
Four components

- a variable and its initial value.
- a condition when to keep going / stop
- how to increment the variable each time
- actions to perform for each time

num:

```
// print each number from 1 to 100:
for ( int num =1 ; num <= 100 ; num = num + 1 ) {
    Serial.println(num);
}
```

For statement



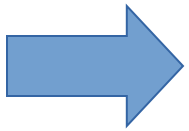
shorthand for num = num + 1

```
for ( int num = 0 ; num < 1000 ; num++ ) {
    Serial.println(num);
}
```

- Meaning:
  - initialise the variable
  - repeat, as long as the condition is true:
    - do the actions
    - do the increment

Iteration: for-loop statement

```
expr1;
while (expr2) {
    statement
    expr3;
}
```



```
for (expr1; expr2; expr3){
    statement
}
```

```
void setup() {
   int i = 0;
   Serial.begin(9600);
   while (i < 4) {
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i*i);
      i++; // or i = i + 1
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```

```
void setup() {
   Serial.begin(9600);
   for (int i = 0; i < 4; i++){
      Serial.print(i);
      Serial.print("^2= ");
      Serial.println(i * i);
   }
   Serial.println ("End of loop!");
}
void loop() {
}
```

Repetion/Iteration

Several different ways of specifying repetition.

- Counted **For** statement: Do something to each number from …..

```
for ( int num = ⟨start⟩;  num <= ⟨end⟩ ;  num = num + ⟨increment⟩ ) {
    do something with num
}
```

- **While** statement: Repeat some action while some condition is still true

```
while (condition-to-do-it-again ) {
    actions to perform each time round
}
```