Engineering Technology (ENGR 101)

Conditional Statements and Iterations

Mohammad Nekooei

Engineering and Computer Science

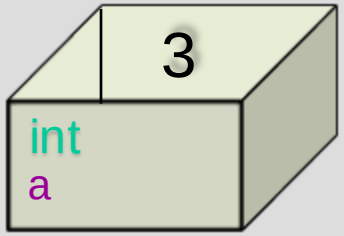
Victoria University of Wellington

Using the if Statement

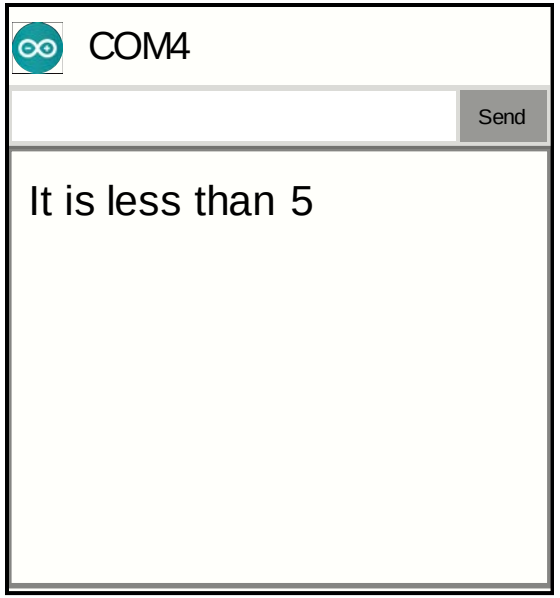
```
if (expression){
        statement
}
```

- If expression evaluates to true, statement is executed

```
int a = 3;
void setup() {
   Serial.begin(9600);
   if( a < 5 ){
      Serial.println("It is less than 5");
   }
}
void loop() {

}
```

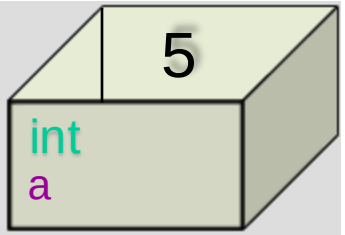


- **Can do an action only in some circumstances**



Using *if-else*

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

}
```



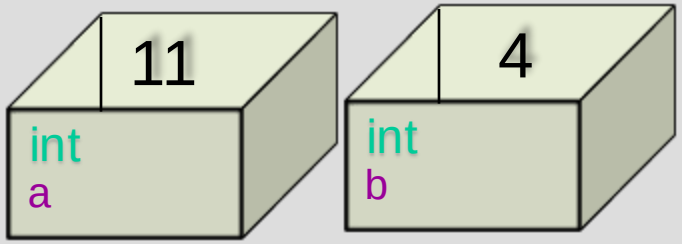
Can choose between different actions

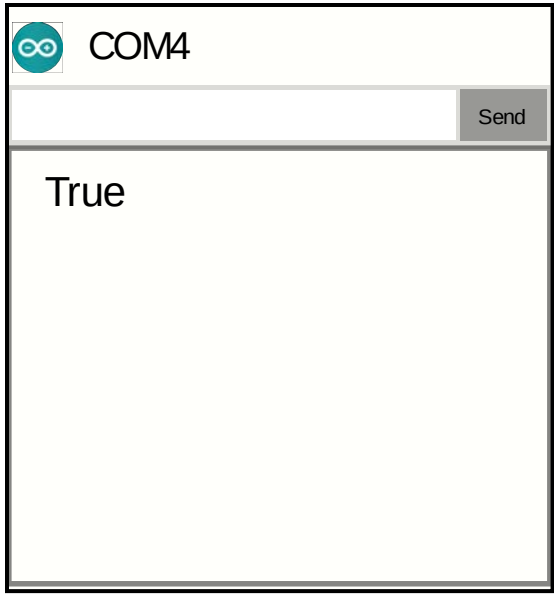
```
if (expression)
      statement1
else
      statement2
```

- The else part is optional
- If expression evaluates to true, $statement_1$ is executed
- Else, $statement_2$ is executed if the else part is present

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

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

}
void loop() {
}
```





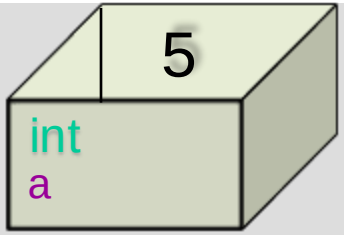
Using else-if statement

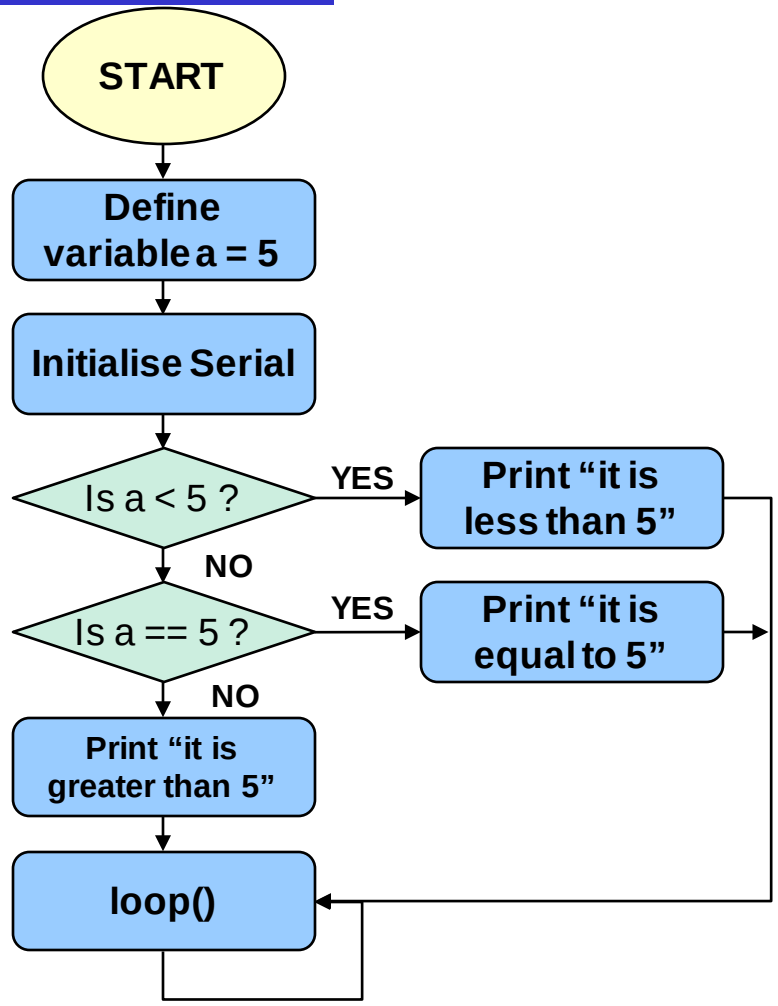
- The *else* part is optional
- If $expression_k$ (where k is 1,2,…,N) evaluates to true, $statement_k$ is executed
  - The rest of *else if* and *else* are not evaluated any further
- If none of $expression_1$ to $expression_N$ is true, then the else statement is executed (if present)
- The statements can be single or compound statements

```
if (expression1)
      statement1
else if (expression2)
      statement2
. . .
else if (expressionN)
      statementN
else
      statement
```

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() {

}
```



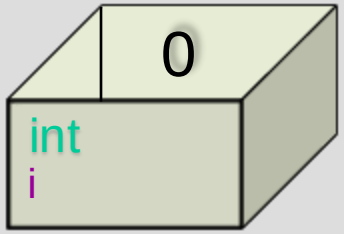


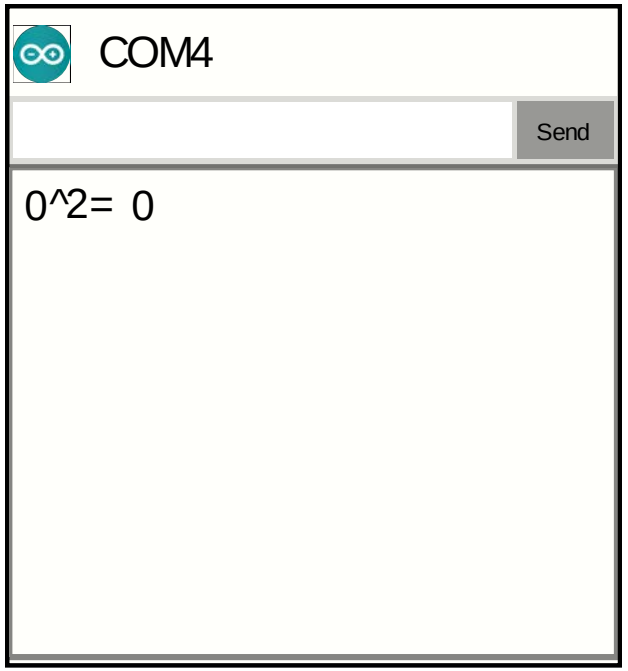
Iteration: while-loop statement

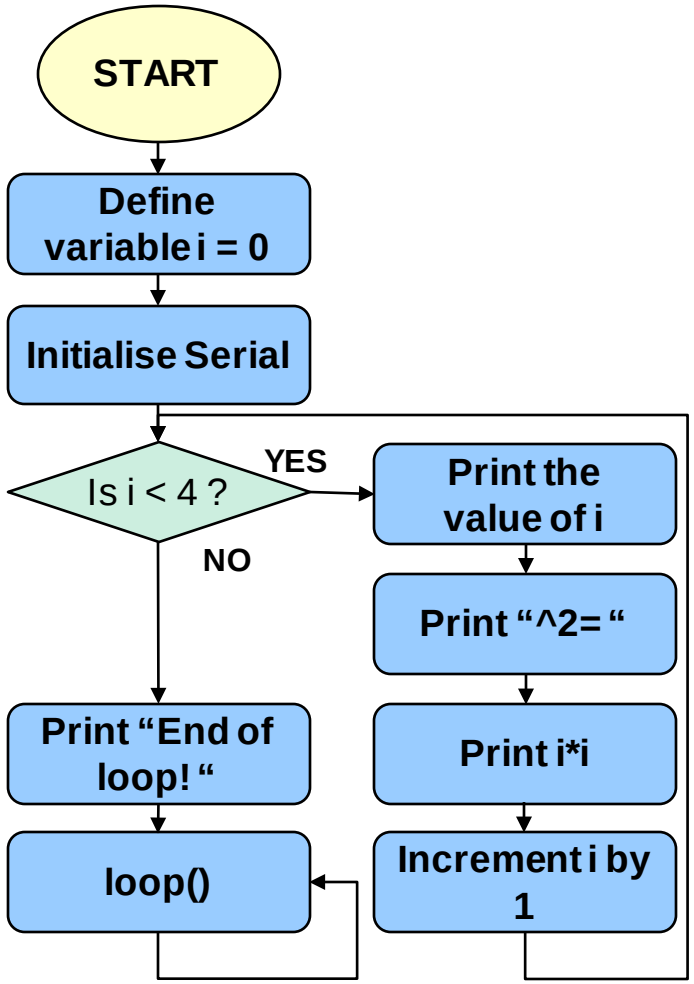
```
while (expression)
    statement
```

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

```
void setup() {
   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() {
}
```





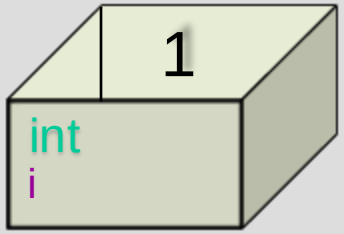


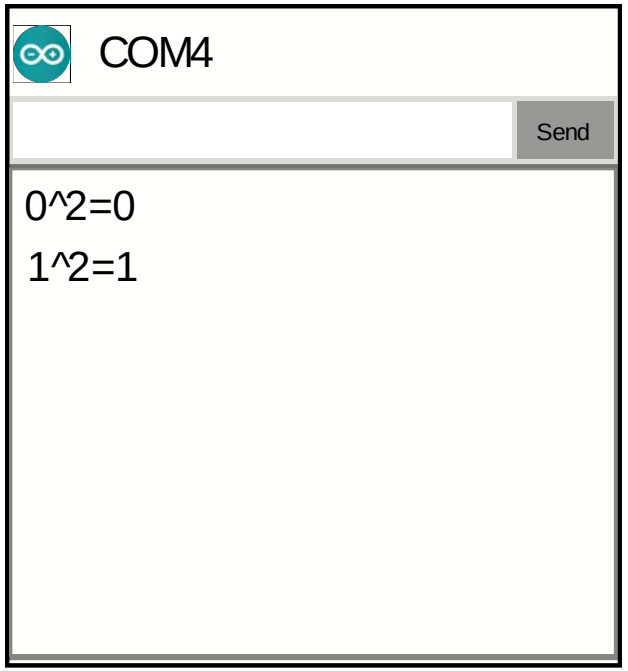
Iteration: while-loop statement

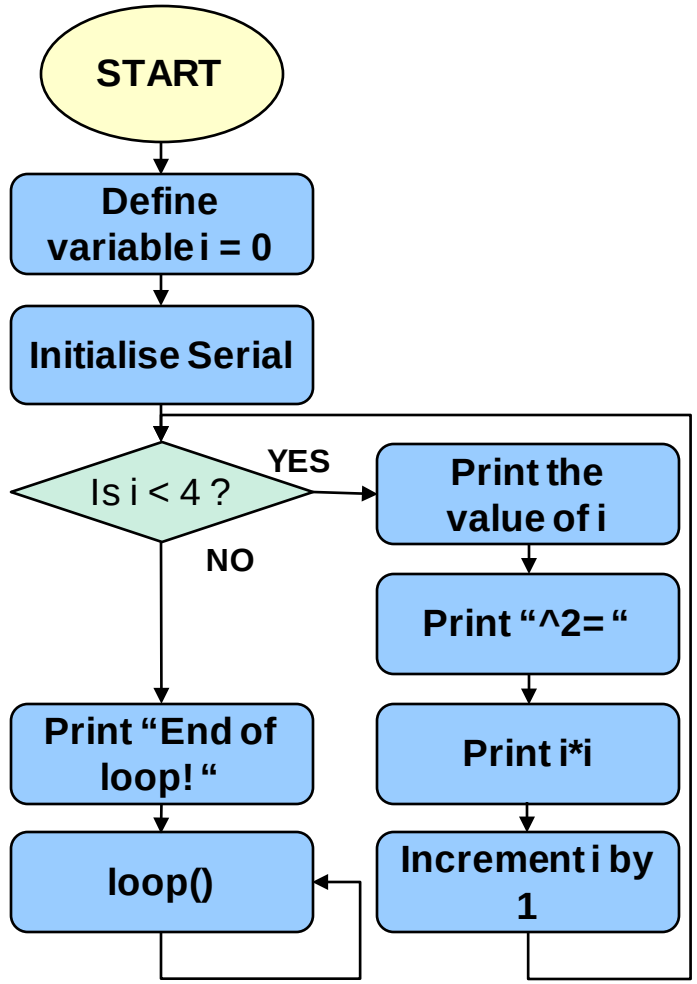
```
while (expression)
    statement
```

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

```
void setup() {
   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() {
}
```





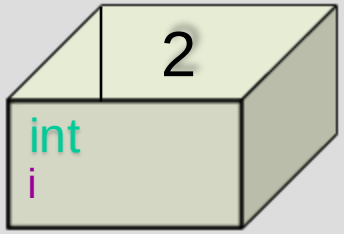


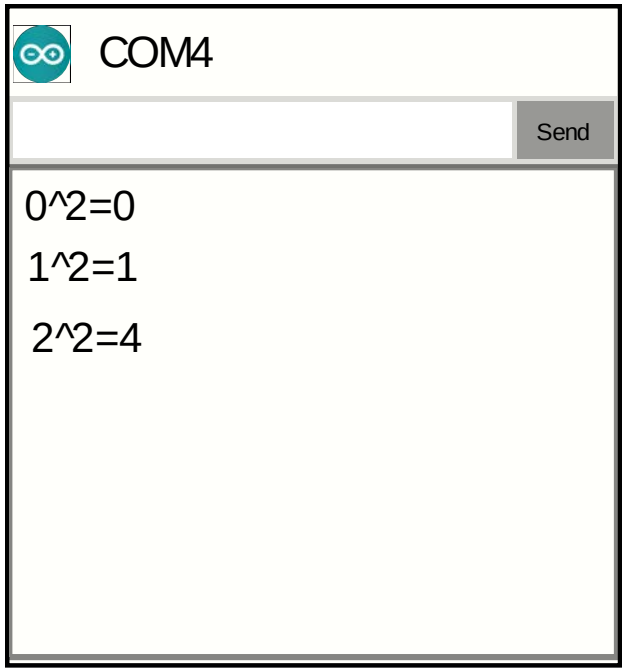
Iteration: while-loop statement

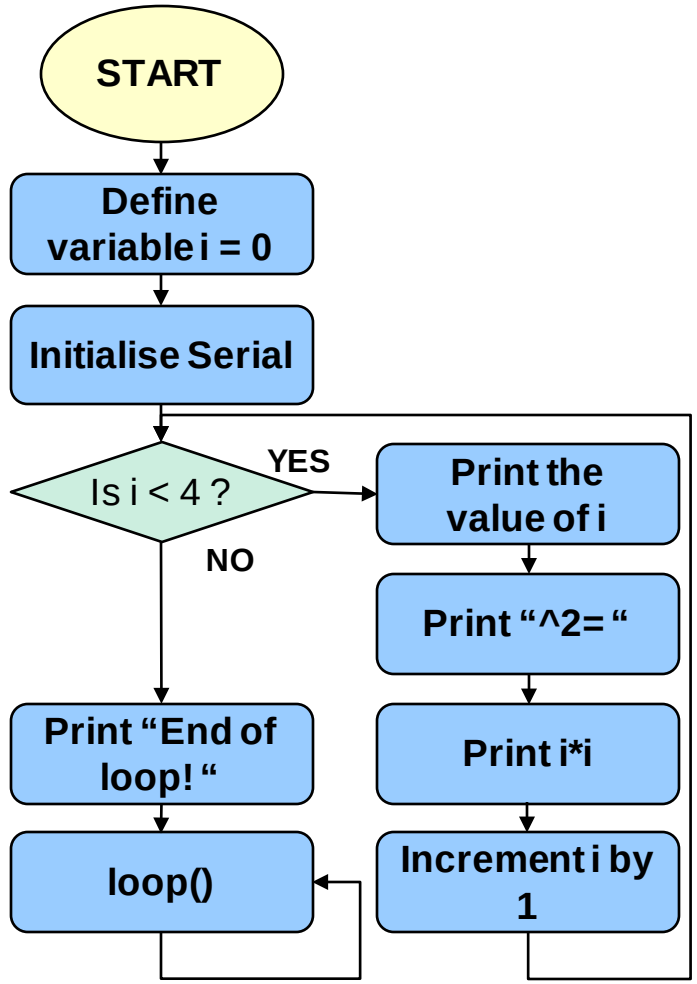
```
while (expression)
    statement
```

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

```
void setup() {
   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() {
}
```





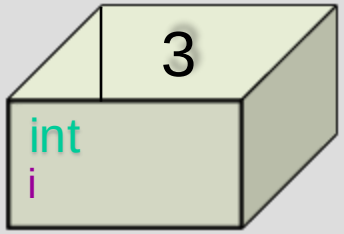


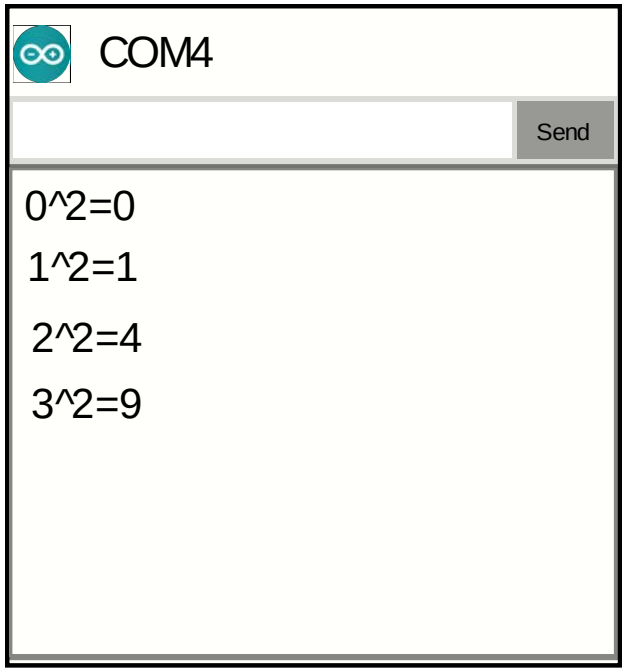
Iteration: while-loop statement

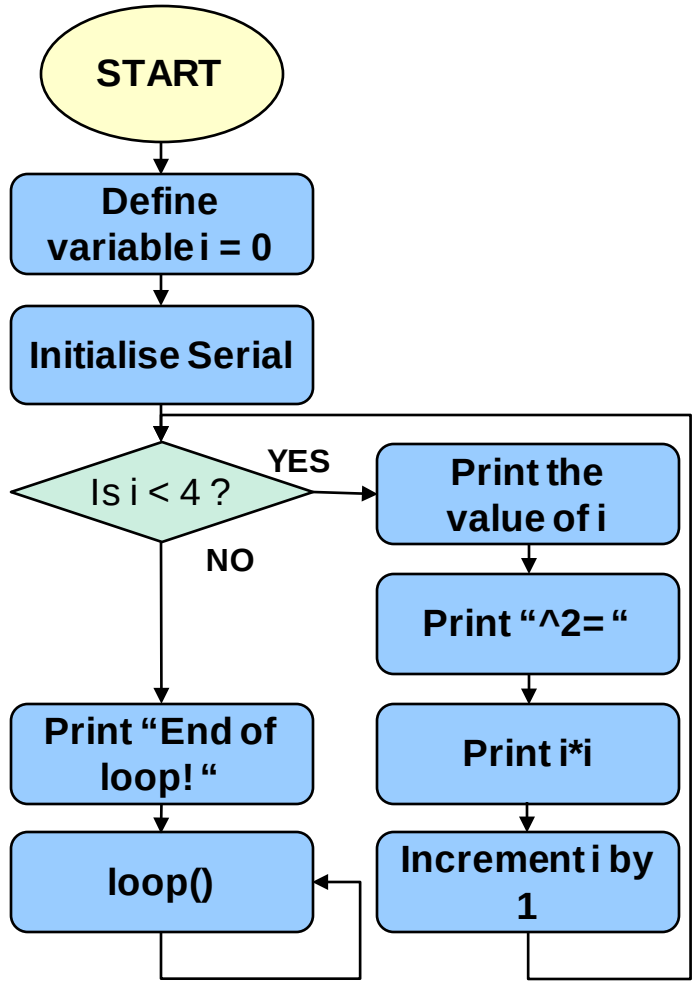
```
while (expression)
    statement
```

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

```
void setup() {
   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() {
}
```





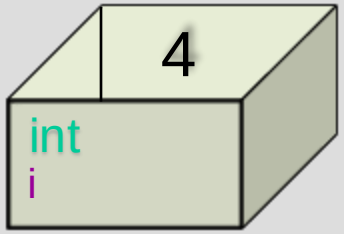


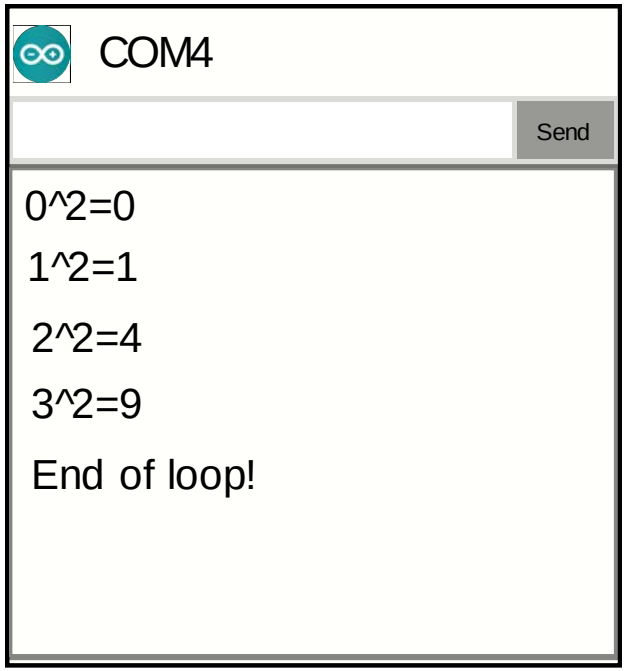
Iteration: while-loop statement

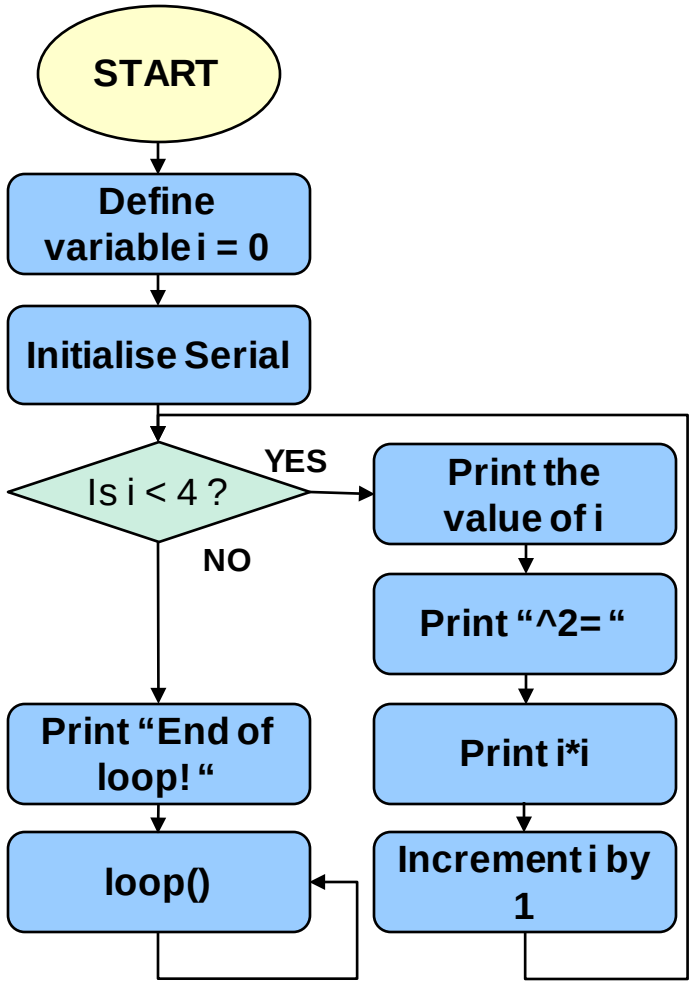
```
while (expression)
    statement
```

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

```
void setup() {
   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() {
}
```



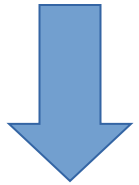




Iteration: for-loop statement

- 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;
}
```

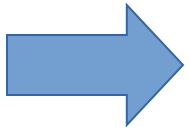


```
for (expr1; expr2; expr3){
    statement
}
```

```
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() {
}
```

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() {
}
```

Arduino Functions

- What are functions good for?
  - structuring our thoughts (structured programming)
  - allowing us to re-use code, reducing work and reducing errors

- A C program can be modularised by functions
  - A big program can be broken down into a number of smaller ones

```
void setup() {
   Serial.begin(9600);
   DashedLine();
   Serial.println("| Program Menu |");
   DashedLine();
}
void loop() {
}
void DashedLine() {
   Serial.println("----------------");
}
```

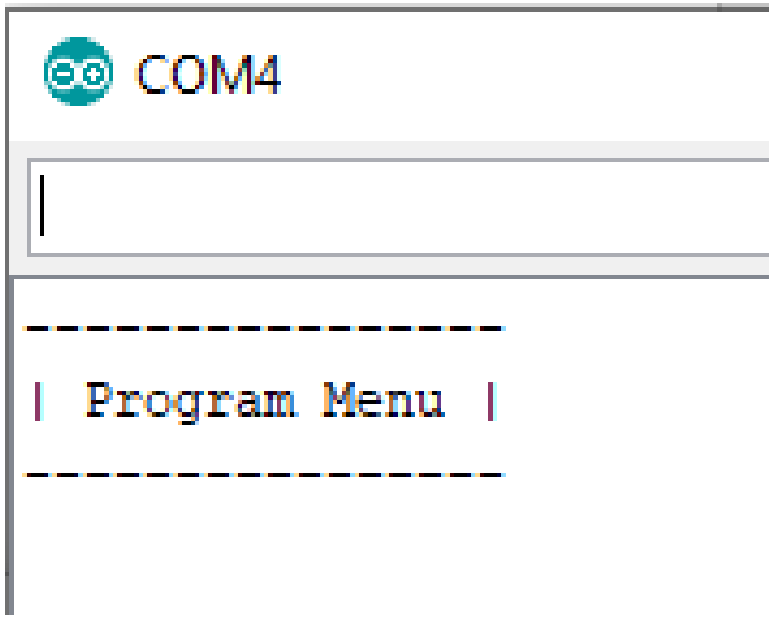
Function is called here

Function is called here

Return type

Function Name

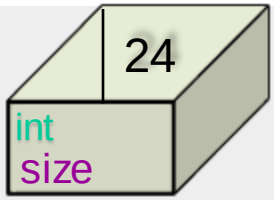
Statement(s) that run when function is called



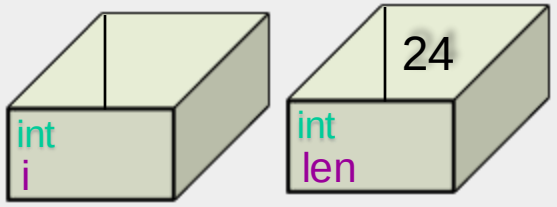
Passing a Value to a Function

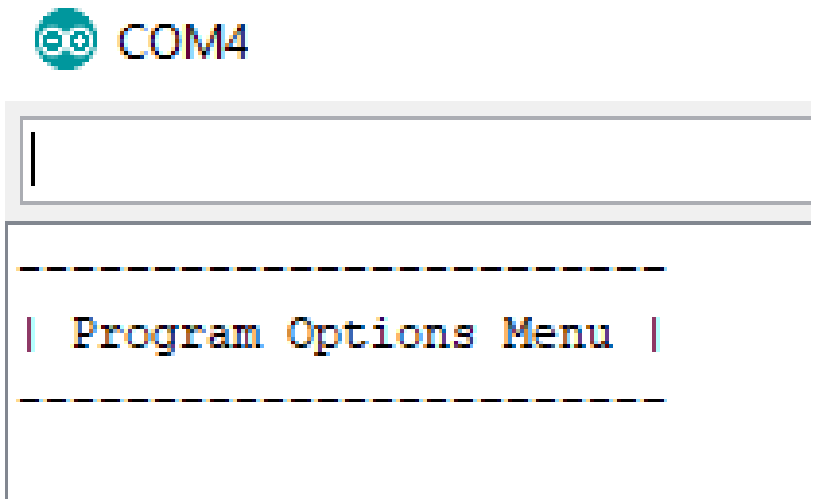
```
void setup() {
   Serial.begin(9600);
   int size = 24;
   DashedLine(size);
   Serial.println("| Program Options Menu |");
   DashedLine(size);
}

void loop() {
}

void DashedLine(int len) {
   int i;
   // draw the line
   for (i = 0; i < len; i++) {
      Serial.print("-");
   }
   Serial.println("");
}
```

Passing a value



Arguments





Parameters

Special variables which are given values each time the method is called. Body of method can use the values in the parameters