Recall: If-Then-Else

In general, we test a *condition*, and depending on the result, we execute some *statements*

If (*condition*) then
    statements for condition TRUE
else
    statements for condition FALSE
end if

Recall: Conditions

Remember that all conditions must resolve to either TRUE or FALSE values:

- if (age < 16) then...
- if (sum >= 12) then...
- if (ratio > 4/3) then...
- if (street = “Main”)...
- if (city not= "Ottawa")...

Recall: Testing Multiple Conditions

Not all situations can be handled by a single condition such as (a>b). As our selection process becomes more complex, we can expand our use of the basic if-then-else construct.

1. nested “if” statements
2. “else if” statements
3. logical operators

Recall: Variables & Data Types

We store information in memory, and the specific location in memory is called a variable (because its value can vary).

When we declare a variable:

1. a space is reserved in memory for that data
2. a name is reserved to identify that data

When we declare variables, we also specify the data type. This is done to help the computer understand what we expect to use the variable for.

Boolean Variables

We have previous discussed the primitive data types of integer and real. Most programming languages also include a boolean data type, which is used to store values of TRUE or FALSE.

When naming boolean variables, always try to choose a name that answers a clear question:

```
var isRaining : boolean
var needUmbrella : boolean
var gameOver : boolean
```

Example: Prime Numbers (basic code)

```
var number : int

put "Enter a number between 1 and 10: " ..
get number

if (number = 2 or number = 3
        or number = 5 or number = 7) then
    put number, " is prime."
else
    put number, " is not prime."
end if
```

Example: Prime Numbers v2 (detect invalid input)

```
var number : int

put "Enter a number between 1 and 10: " ..
get number

if (number >= 1 and number <= 10) then
    if (number = 2 or number = 3
            or number = 5 or number = 7) then
        put number, " is prime."
    else
        put number, " is not prime."
    end if
else
    put "Invalid number!"
end if
```

Example: Prime Numbers v3 (use boolean variables)

```
var number : int
var isValid, isPrime : boolean

put "Enter a number between 1 and 10: " ..
get number

isValid :=number >= 1 and number <= 10
isPrime := number = 2 or number = 3 or number = 5 or number = 7

if (isValid) then
    if (isPrime) then
        put number, " is prime."
    else
        put number, " is not prime."
    end if
else
    put "Invalid number!"
end if
```

Boolean Variables: Complexity

As with many examples, you might think adding boolean variables has made our code more complex, and was not worth doing.

In reality, it has made our program a bit longer, but at the same time, the code is easier to read. The purpose of each variable is quite clear, and the IF statements are now very clear.

What if we needed to detect a prime number later in our program?

Example: Prime Numbers v4 (use isPrime multiple times & ways)

```
put "Enter a number between 1 and 10: " ..
get number

isValid :=number >= 1 and number <= 10
isPrime := number = 2 or number = 3 or number = 5 or number = 7

if (isPrime and isValid) then
  ...

if (isPrime or not isValid) then
  ...

if (not isPrime and isValid) then
  ...
```