

# Lecture 3.3 – Conditionals

## Specific Learning Objectives:

**1.2.1 – Understand the way computers execute commands.**

**1.2.7 – Understand and successfully execute conditional if/else statements (vectorized and non-vectorized).**

**3.5 – Think and work independently with code.**

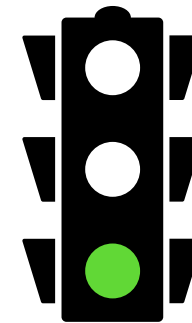
# What is “Flow Control” in Computing?

- Computers do one thing really, really well: the same task over and over again exactly the same way. However, they are really bad about making judgments and decisions!
  - Flow control helps get around this by giving basic instructions for possible scenarios ahead of time.
  - Two main components of flow control: **conditional statements** and **looping**.
  - These are the same for all computing languages and differ mostly in syntax!
  - We will cover conditionals today and looping next time!

# Conditional Statements

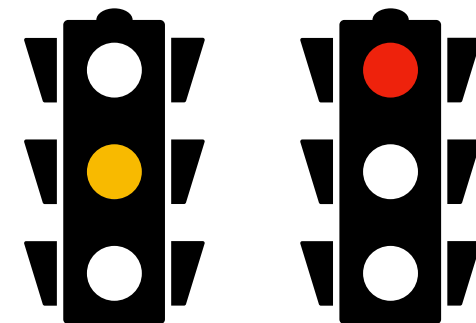
- **Conditional statements** break down decisions for the computer in binary decisions (true or false).
- Basic structure goes “if this condition is met, I want you to do this.”

**If the traffic light is green, you should go.**



- If the condition is not met, the computer does not follow the instruction.

**If the light is not green, you should not go.**



# Conditional if Statements

- How do you write conditional statements in R?

I'm giving you a condition,  
please test it.



```
if(TRUE) print("This should always print.")
```

The diagram illustrates the R `if` statement syntax. A blue arrow points from the text "I'm giving you a condition, please test it." to the `if` keyword. An orange arrow points from the text "This is the condition to test." to the `TRUE` value. A purple arrow points from the text "This is what you should do if the result of the test is TRUE." to the `print("This should always print.")` statement.

This is the  
condition to  
test.

This is what you should do  
if the result of the test is  
TRUE.

**If FALSE, then don't run the purple code, skip it and keep moving.**

# Conditional if Statements

This prints.



```
if(TRUE) print("This should always print.")
```

```
if(FALSE) print("This shouldn't ever print.")
```



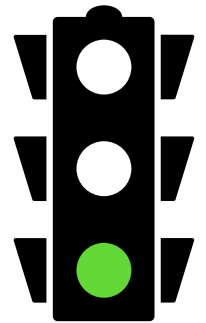
This does not.

# Conditional if Statements

- Let's write a conditional statement to describe our first example: green lights mean go.

- What's the condition that means go? **if the light is green**

```
light == "green"
```



- What should happen if the condition is true?

```
print "go!"
```

```
print("go!")
```

- Put it together in one statement:

```
if(light == "green") print("go!")
```

- Try it!

**change light to "red" and try it again**

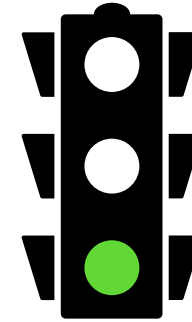
# Check Your Understanding

**Write a conditional that prints “higher” if the number `n` is above 10 and “lower” if the number `n` is below 10.**

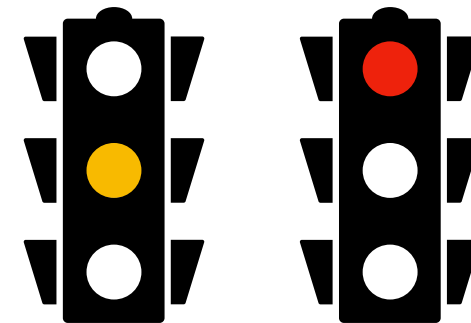
# Conditional if-else Statements

- What happens if you need the computer to do something else besides the thing if it's true? (Like an either or)

**If the traffic light is green, you should go.**



**If the light is not green, you should STOP.**



- Use an if-else statement:

**Note: use  
whitespace to  
help with  
readability!**

```
if(light == "green") {  
    print("go!")  
} else {  
    print("STOP!")  
}
```

← If TRUE, do this.

← If FALSE, do this.



# Conditional else if Statements

- Remember! if statements are **binary** and only work for things that return TRUE or FALSE!
- If you have more than two things you want to happen, you **HAVE** to make it into a binary choice for the computer! Sorry 🙄
- `else if` statements can help by giving additional opportunities.

```
if(light == "green") {  
    print("go!")  
} else if(light == "yellow") {  
    print("slow down!")  
} else {  
    print("STOP!")  
}
```

← If TRUE, do this.

← If second thing is TRUE, do this.

← If neither are TRUE, do this.

- You can string as many `else if` statements together as you want.

# The switch function

- R has a special function that helps make if else statements more compact:  
`switch()`

```
if(light == "green") {  
  print("go!")  
} else if(light == "yellow") {  
  print("slow down!")  
} else {  
  print("STOP!")  
}
```

← If TRUE, do this.

← If second thing is TRUE, do this.

← If neither are TRUE, do this.

```
switch(light,  
  "green" = print("go!") ,  
  "yellow" = print("slow down!") ,  
  print("STOP!")  
)
```

# Check Your Understanding

**Write a conditional that prints “higher” if the number `n` is above 10 and “lower” if the number `n` is below 10.**

**Add two other conditions that prints “a lot lower” if `n` is below 0 and “a lot higher” if `n` is above 20.**

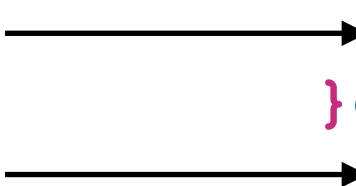
**Why won't `switch()` work with this example?**

# Nesting Conditional Statements

- You can also nest conditional statements inside other conditional statements (although, don't go nuts).

**Nested statement is indented an additional level**

```
if(light == "green") {  
    if(carinfront == "stopped") {  
        print("Honk!")  
    }else{  
        print("go!")  
    }  
} else if(light == "yellow") {  
    print("slow down!")  
} else {  
    print("STOP!")  
}
```



- Be sure to use whitespace to help conditionals remain readable.
- Too many nested statements can get really hard to read, so you should avoid this!

# Vectorized conditionals with `ifelse()`

- In regular if else statements, if you are testing a vector, R will produce an error. This isn't great if you need your code to run and all the positions of the vectors tested!

```
light <- c("green", "yellow", "green", "red")  
  
if(light == "green") {  
  print("go!")  
} else {  
  print("STOP!")  
}
```

Error in if (light == "green") { : the condition has length > 1

- For these, use `ifelse()` which is a vectorized if else function!

```
ifelse(light == "green", "go!", "STOP!")
```

```
[1] "go!"      "slow down!" "go!"      "STOP!"
```

# Action Items

- 1. Complete Assignment 3.3.**
- 2. Read Davies Ch. 10.2 for next time.**