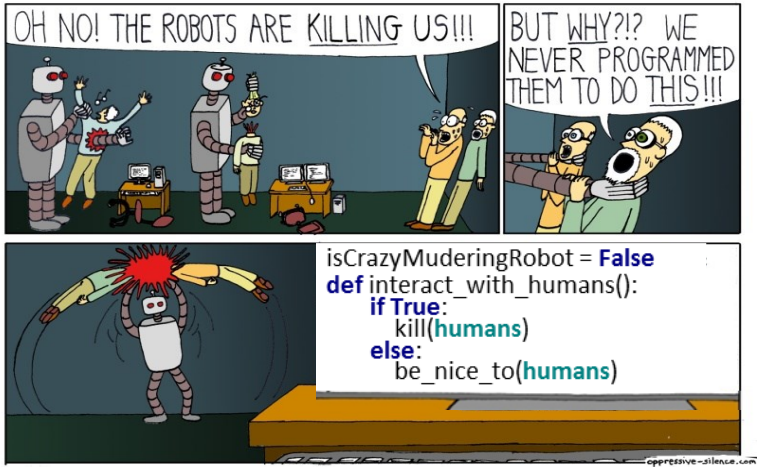
COMP1811
Paradigms of Programming

Python Conditionals and Boolean Expressions

flow control



http://oppressive-silence.com/comic/oh-no-the-robots

bit.ly/COMP1811_Conditionals

Dr. Yasmine Arafa

Today's Agenda...

- Boolean data type
- Boolean Logic
- Flow control / branching
  - programming patterns for program decisions
  - conditional statements (control structures)
    - `if`
    - `if-else`
    - `if-elif-else`
    - *nested* `if` *statements*

Simple Decisions

- So far, we've viewed programs as sequences of instructions that are followed one after the other.
  - While this is a fundamental programming concept, it is not sufficient in itself to solve every problem.
  - We need to be able to alter the sequential flow of a program to suit particular problems.
- Control structures allow us to alter this sequential program flow.
  - Also called decision structures or conditional statements.
  - Allow different blocks of code to be executed based on some condition.

Conditional Statement
the concept...

Conditional Statement

Conditional Expression (*always evaluates to either True or False*)

Block of code containing one or more Python statements.

Block runs if expression evaluates to **True.**

Block of code containing one or more Python statements.

Block runs if expression evaluates to **False.**

Note: Only one of the blocks will run each time the conditional expression is evaluated.

To write conditional statements, we must write conditional expressions first!

Conditional Expressions
also called Boolean Expressions or Conditions

- Are expressions that always evaluate to either `True` or `False`.
  - They produce a value of type `bool` (boolean value).
  - Example: `a > b # true if a is greater than b; false otherwise`
- Named after the English mathematician George Boole.
- These expressions are formed with Relational and/or Logical operators:
  - Relational (comparison) operators:
    - Determines whether a specific relationship exists between two values.
    - Example: greater than (>) `a > b`
  - Logical operators:
    - Used to create complex and compound boolean expressions.
    - Example: `a > b and b < c`

Forming Conditions / Boolean Expressions
What does a condition look like?

- **Syntax:**

```
<expr> <rlop> <expr>
```

where <rlop> is short for *relational or logical operator*;
<expr> each operand can be a value, variable or another expression.

- **Examples:**

```
payment != balance
height >= 180 and age < 18
fruit == "Apples"
```

Relational Operators
used with integers, floats, and strings

- Are comparison operators that check for relations between operands:

Operator	Description	Example (x=10, y=10)	Result
<	Less than	`x < y`	`False`
>	Greater than	`8 > 6`	`True`
<=	Less than or equal to	`x <= y`	`True`
>=	Greater than or equal to	`25 >= 65`	`False`
==	Equal to	`x == y`	`True`
!=	Not equal to	`"oranges" != "apples"`	`True`

Logical Operators

- Operators that can be used to create complex Boolean expressions:

Operator	Description
`and`	`True` if both operands are `True`
`or`	`True` if at least one operand is `True`
`not`	`True` only if operand is `False` (invert)

- `and` and `or` operators:
  - binary operators (i.e. have two operands),
  - connect two Boolean expressions into a compound Boolean expression:
    Example: `height >= 180 and age < 18`

Logical Operators
Cntd.

- `not` **operator:**
  - unary operator (i.e. has one operand),
  - inverts the truth of its Boolean operand (i.e. inverts its value).

  - Example:

```
x = True
print(not x)
```

Output
`False`

Logical Operators
truth table

- **Truth table:**

x	y	x and y	x or y	not x
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

- **Example:**

```
cold, wet = True, True
print(cold and wet)
hour = 8
print(hour<9 or hour>17)
```

Output
True
True

Order of Precedence
order matters!

- Precedence of execution is from left to right, as follows:
  - note: square brackets signify same level operators.

Highest

```
()    x**y    [x*y, x/y, x//y, x%y]    [x-y, x+y]

[x==y, x!=y, x>=y, x>y, x<=y, x<y]

not x    x and y    x or y
```

Lowest

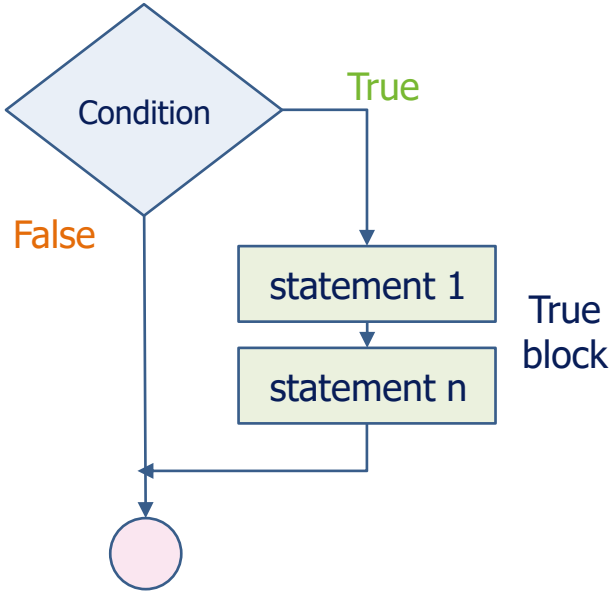
Constructing Conditional Statements

Conditional Statements
if statement

- The Python if statement is used to implement conditionals (decisions).
- Syntax:

```
if <condition>:
    <body>
```

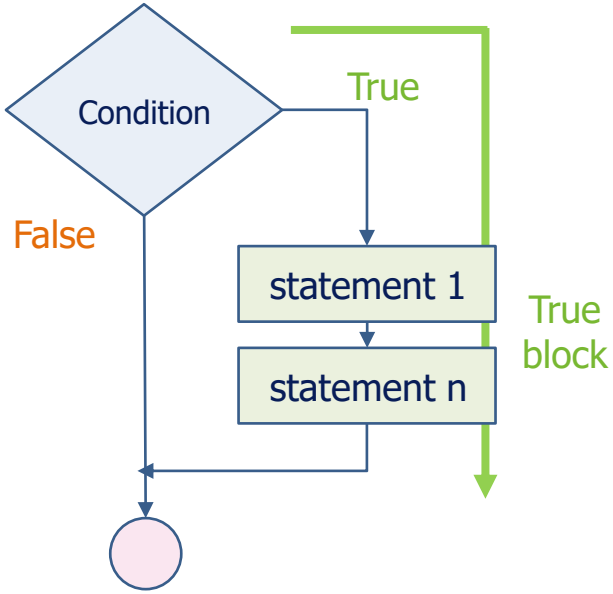
  - `<body>` is a statement or a block of statements indented under the `if` heading;
  - `<body>` must be indented.



- Semantics
  - First, <condition> is evaluated.
  - If it is **True**, <body> is executed, then control passes to the next statement after the conditional.
  - If **False**, <body> is skipped, and control passes to next statement after conditional.

Conditional Statements
if statement, *Cntd.*

- The if statement is a one-way conditional (decision) statement.
- It has only one branch.
  - if condition is true, control flow passes to the true block (branch) and its statements are executed sequentially;
  - otherwise, control passes to the next statement in program that is directly after the conditional statement.



Example 1
simple grade classification

Pseudocode

Get grade from user.
If grade is greater than or equal to 70, then
    Display "It's a first!"
...

Python

```
grade=int(input("Enter grade:"))

if grade >= "70":
  print("It's a first!")
```

Example 2
Number Guessing Game

- Guess the output when the user enters the number 10:

program

numberGuessingGame.py

```
import  random
random_num = random.randint(1,10)

input_num = int(input("Guess a number: "))
if input_num == random_num:
    print("Success! You win!!")
if input_num > random_num:
    print("Too high! try again...")
if input_num < random_num:
    print("Too low! try again...")
```

console

Example 2
Number Guessing Game

- Guess the output when the user enters the number 10:

program

numberGuessingGame.py

```
import random
random_num = random.randint(1,10)

input_num =
if input_nu
    print("
if input_nu
    print("
if input_nu
    print("
```

Not seen this before!

- **import** allows you to use code defined elsewhere (either from a standard **module**, or your own).
- A **module** is simply a file containing Python code.
- The **random module** contains useful functions for generating random numbers.

Example 2
Number Guessing Game

- Guess the output when the user enters the number **10**:

program

numberGuessingGame.py

```
import  random
random_num = random.randint(1,10)

input_num = int(input("Guess a number: "))
if input_num == random_num:
    print("Success! You win!!")
if input_num > random_num:
    print("Too high! try again...")
if input_num < random_num:
    print("Too low! try again...")
```

random_num = 10

input_num = 10

True

False

False

console

```
Success! You win!!
```

Example 2
Number Guessing Game, *Cntd.*

program

numberGuessingGame.py

```
import  random
random_num = random.randint(1,10)

input_num = int(input("Guess a number: "))
if input_num == random_num:
    print("Success! You win!!")
if input_num > random_num:
    print("Too high! try again...")
if input_num < random_num:
    print("Too low! try again...")
```

Note: this is not really very good coding! Why?

if Statements
some notes...

Conditional expression

No parentheses needed

Colon
beginning of block

```
if is_world_flat :
    print("It sure looks flat… ")
    print("Don't  fall  off!")

print("Oh get real!")
```

Blocks must be indented

Code block executed if condition evaluates to True

No longer part of block

- Parentheses are only needed for compound expressions or to improve readability.
- A block ends when indentation is removed.

Blocks

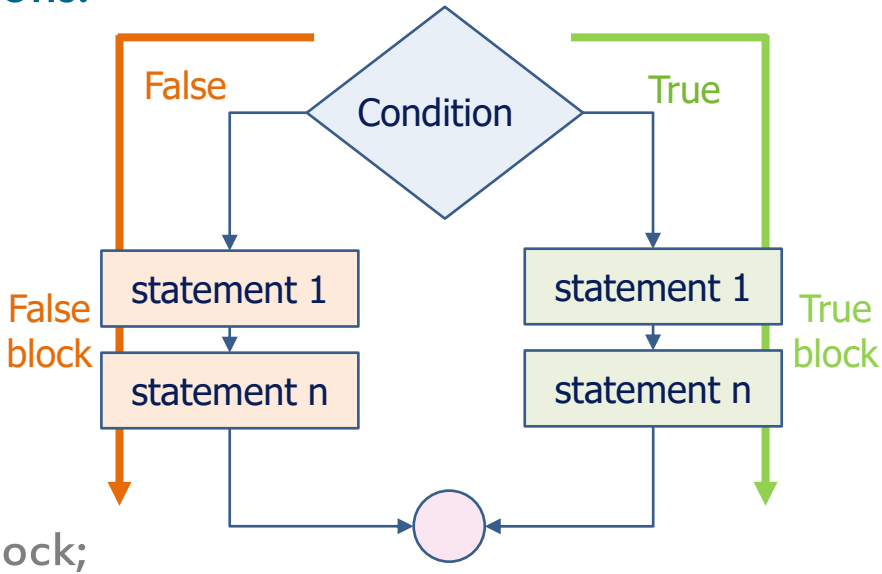
- Python uses indentation (whitespaces at the start of a line) to denote blocks of code.
- A colon: at the end of a line indicates that the next line of code is the beginning of a block.
- Lines within the code block are indented at the **same level**.
- Remove indentation to denote end of a code block.
- Blocks are needed if you want parts of your code to run only when certain conditions are met.

More Decision Choice

- The if statement allows a particular block of code to run if its condition evaluates to `True`.
- What if you need to run a different block of code if the condition evaluates to `False` or if you need more than two alternatives.
- A *selection statement* provides the means of choosing between two or more paths of execution.
- Two general categories:
  - Two-way conditionals (`if-else` statements)
  - Multiple-way conditionals (`if-elif` statements)

Conditional Statements
two-way conditionals (decisions)

- `if` statements are used as guarded actions:
  - only execute a block if a condition evaluates to `True`.
- Another programming pattern is to do one thing or another.
- It has two branches or paths.
  - if condition is true, control flow passes to the true block (branch);
  - otherwise, control passes to the false block;
  - when either block completes, control passes to the next statement in program that is directly after the conditional statement.



if-else Statements
the else clause allows for two-way decisions

- In Python, a two-way decision can be implemented by attaching an `else` clause onto an `if` statement. This is called an `if-else` statement.
- Syntax:

```
if <condition>:
    <body>
else:
    <body>
```

then clause

else clause

can be single or compound statements

- Semantics:
  - Python first evaluates the condition.
  - If condition is True, the *then* clause (true block) is executed.
  - If condition False, the *else* clause (false block) is executed.
  - When either body completes, control passes to the line after conditional.

Example 3
have I passed?

- What is the output if the user enters a mark of 75?

program

isItAPass.py

```
my_mark = int(input("Enter your mark: "))
if my_mark >= 40:
    print("You've passed…")
else:
    print("Try to work a bit harder!")
```

my_mark = 75 my_mark = 75

True

console

```
You've passed…
```

Example 3
have I passed? *Cntd.*

- What is the output if the user enters 35:

program

isItAPass.py

```
my_mark = int(input("Enter your mark: "))
if my_mark >= 40:
    print("You've passed…")
else:
    print("Try to work a bit harder!")
```

my_mark = 35 my_mark = 35

False

console

```
Try to work a bit harder!
```

More Examples
check if number is even and less than 40

- What is the output if the user enters 16:

program

checkEvenNumLess40.py

```
x = int("Enter an integer number: ")
# x%2 is 0 if number is even and odd otherwise.
if x % 2 == 0 and x < 40:
    print(x+" is an even number < 40.")
else:
    print(x+" is an odd number or is > 40.")
```

x = 16

x = 16

True

console

```
16 is an even number < 40.
```

Nested Conditionals

- The *then* and *else* blocks of an if-statement can contain other conditionals too.
- This is called nested conditionals or nested if-statements.
- A nested if statement means an if-statement inside another if-statement.
- This is used in situations where you want to check for other condition(s) after a condition evaluates to true.

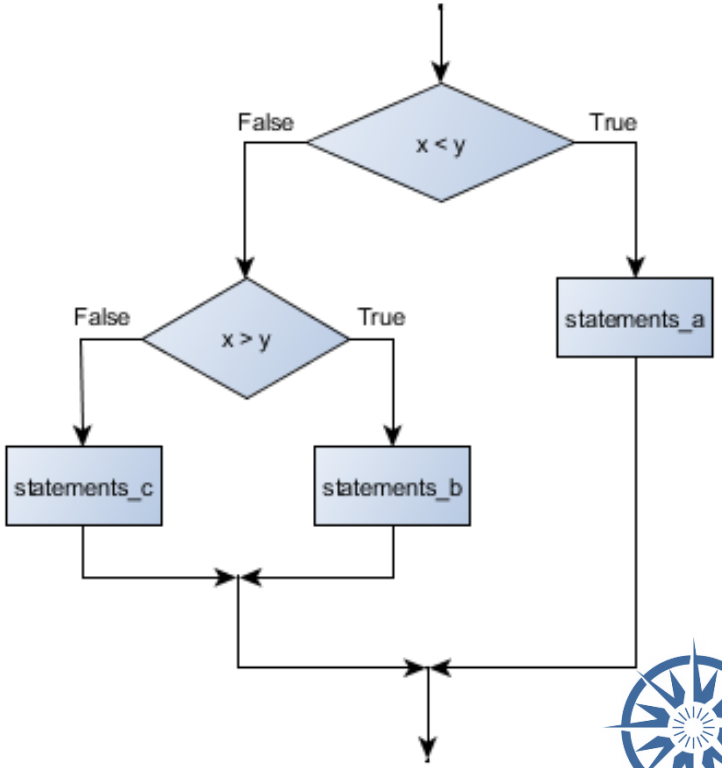
Nested Conditional
Cntd.

- **Syntax:**

```
if <condition>:
    <body>
    if <condition1>:
        <body>
    else:
        <body>
        …
else:
    <body>
    if <condition2>:
        <body>
    else:
        <body>
        …
```

Example 4
nested conditional

```
if x < y:
    STATEMENTS_A
else:
    if x > y:
        STATEMENTS_B
    else:
        STATEMENTS_C
```



- One conditional **nested** within another.

Example 5
what shall I wear?

program

```
cold = True
wet = True
if wet:
    if cold:
        print("Wear a waterproof coat.")
    else:
        print("Take an umbrella.")
else:
    if cold:
        print("Wear a warm coat.")
    else:
        print("A jacket is enough.")
```

whatToWear2.py

True
True

console

```
Wear a waterproof coat.
```

Nested Conditionals
Cntd.

- Although the indentation of statements makes the structure apparent,
  - sometimes nested conditionals may become difficult to read!
- Logical operators often provide a way to simplify nested conditionals.
  - Example, the following code can be rewritten using a single conditional:

```
if 0 < x:
    if x < 10:
        print("x is a +ve number between 0 and 10.")
```

```
if 0 < x and x < 10:
    print("x is a +ve number between 0 and 10.")
```

```
if 0 < x < 10:
    print("x is a +ve number between 0 and 10.")
```

Syntactic Sugar

More Conditional Statements
multi-way conditionals (decisions)

- Allow selection of one of any number of statements or statement blocks.
  - Useful if the program needs more than two alternative decisions.
- Syntax:

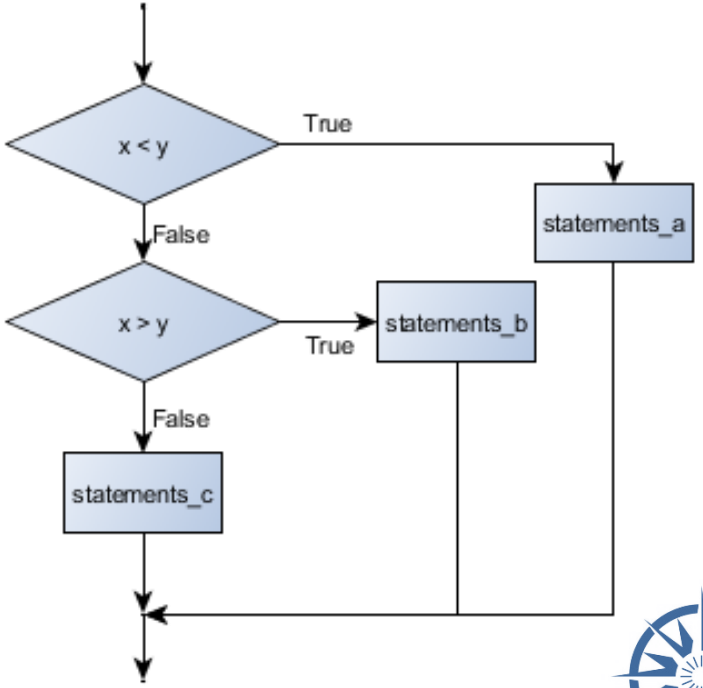
```
if <condition1>:
    <body>
elif <condition2>:
    <body>
elif <condition3>:
    <body>
…
else:
    <default body>
```

Use `elif` to chain a sequence of condition tests.

  - This form of conditional statement sets any number of mutually exclusive code blocks.

Example 6

```
if x < y:
    STATEMENTS_A
elif x > y:
    STATEMENTS_B
else:
    STATEMENTS_C
```



- Exactly one branch will be executed.

http://www.openbookproject.net/books/bpp4awd/ch04.html

Example 7
grade classification

- What is the classification when the user enters 52:

program

```
my_grade = int(input("Enter your grade: "))
if my_grade >= 70 :
    print("First")
elif my_grade >= 60:
    print("Upper second")
elif my_grade >= 50:
    print("Lower second")
elif my_grade >= 40:
    print("Third")
else:
    print("Fail")
```

gradeClassify.py

my_grade = 52

True

console

```
Lower second
```

Example 7
grade classification, *Cntd.*

- What is the classification when the user enters 35:

program

```
my_grade = int(input("Enter your grade: "))
if my_grade >= 70 :
    print("First")
elif my_grade >= 60:
    print("Upper second")
elif my_grade >= 50:
    print("Lower second")
elif my_grade >= 40:
    print("Third")
else:
    print("Fail")
```

gradeClassify.py

my_grade = 35

console

```
Fail
```

Example 8
Number Guessing Game, *Modified*

- Guess the output when the user enters the number **10**:

program

numberGuessingGame.py

```
import  random
random_num = random.randint(1,10)

input_num = int(input("Guess a number: "))
if input_num == random_num:
    print("Success! You win!!")
elif input_num > random_num:
    print("Too high! try again...")
elif input_num < random_num:
    print("Too low! try again...")
```

Random_num = 10

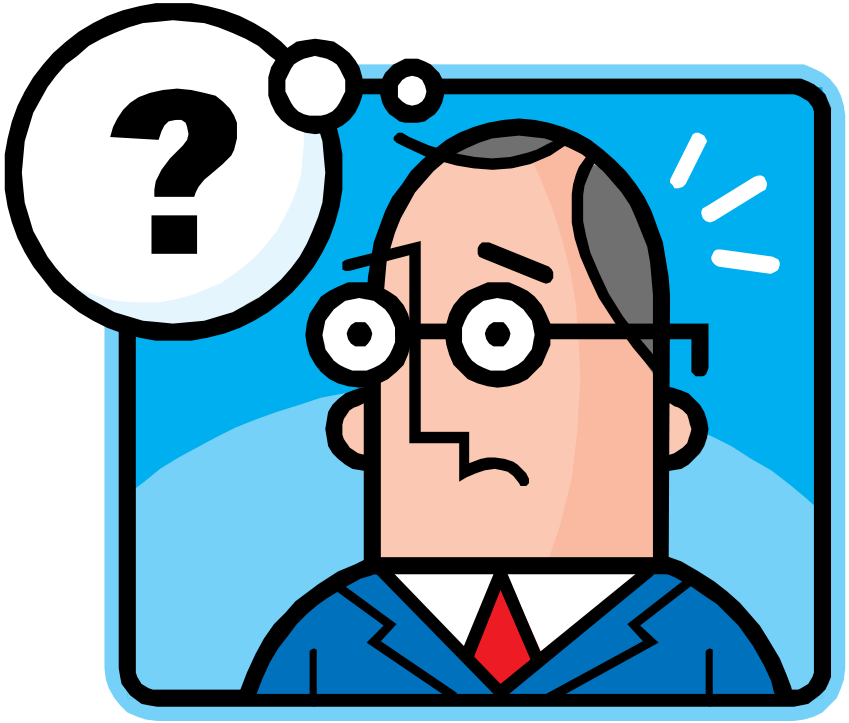
Input_num = 10

True

console

```
Success! You win!!
```

Questions?



Alternative Syntax for Conditionals
Ternary Operators

- Ternary operators allow the evaluation of a condition in a single line
  - by replacing the multiline if-else statement,
  - making the code compact.

- Alternative syntax:

```
<on_True> if <condition> else <on_False>
```

- Example:

```
#get smallest of two values
smallest = a if a < b else b
```

Example 9

- Check if odd number:

```
is_odd = True if x%2 != 0 else False
```

  - is the equivalent to:

```
if x%2 != 0:
    is_odd = True
else:
    is_odd = False
```

Multiple Statements on a Single Line

alternative syntax –*legal in Python*

- `if x!=4: x+=y; print(x); print(y)`
  - Semi colons are used to delimit statements if multiple statements are put on the same line.
  - **If <condition> is True, all three statements are executed**, otherwise, that block is skipped.

- The following is also legal in python:

```
x=4; y=5; z=y*x; print(z)
```

Notes on Conditionals

- You have to have at least one statement inside each branch but you can use the *pass* command while you're still stubbing.

```
if x < 0:
   pass  # Handle negative values
```

The pass Statement
"do nothing"

- The `pass` keyword indicates that nothing happens when it is executed.
- Sometimes useful, e.g. during development:

```
if a <= 40 and b <= 40:
    print("Both marks are below 40. Try harder!")
if a > 40 and b <= 40:
    pass                    # figure out what to do later
if a <= 40 and b > 40:
    pass                    # figure out what to do later
if a > 40 and b > 40:
    pass                    # figure out what to do later
```

Summary

- **Conditionals (Control structures**) allow flow control to change such that particular statement blocks can be executed based on some condition.
- A **Boolean** is a data type that can have either a True or False value.
- A **Boolean expression** evaluates to either True or False and are used as the condition in a control structure.
  - Boolean expressions are constructed using **relational operators** (comparison) and Logic operators.
- There are two basic types control structures that use Boolean expressions: selection and loops (iterations – coming soon).

Conditional Statements

- These types of control structures allow different blocks of code to be executed based on the Boolean expression.
- There are three basic types of conditional in Python:
  - if
  - if-else
  - if-elif-else
  - The **if statement** allows to only execute one or more statements when some condition is met. The addition of the **else statement** allows an alternative action and the addition of **elif**, which stands for 'else if', allows for different conditions and having different actions for each of them.

Further Reading

- RealPython – Boolean Expressions
  - Relational Operators
  - Logical Operators

- **w3school pages**
  (includes topics covered in the lecture and some more advanced)
  - Python booleans
  - if … else

In the lab today …

- There is a list of tasks to complete.
  1. Revise the Python Conditionals notebook and repeat the activities until you're competent in coding them.
  2. Take the quiz on Python Conditionals.
  3. Complete the Python exercises in Lab sheet 1.02. You will need to download and unzip the code needed for these exercises.

Next week…

- More on control flow
- Iteration
  - while loops
  - for loops