ENGG1811: Computing For Engineers

2025 Term 3

Selection Structure

Week 3: Monday 29th September, 2025

Monday 14:00 - 16:00 | HarpM15570

Today

Boolean Expressions

If/Elif/Else Statements

Plotting

Lab Tips

Reminders

- Help sessions will start next week. You can get one-on-one tutoring for content, labs, or assignments. The timetable can be found here.
- Live coding sessions are on Tuesday, 13:00 - 14:00 at Quadrangle G045.
- PASS sessions are still ongoing! The timetable can be found here.
- Remember to ask for help if you need it!
- Make sure you check your marks on the course website.
- Logins to the course website should use your zID, not your student email; otherwise, you cannot attempt the multiple-choice quizzes.

Lab Next Week

- Next week is a public holiday (Labour Day). We will have the lab at the same time but online through Blackboard Collaborate (link and information can be found under Attending Online Labs here) at the usual 2–4pm time.
- If you attend, you will need a working camera and microphone to demonstrate your work.

  If you own a Mac, please see *Allowing screen sharing with a Mac on Blackboard Collaborate* here
- If you cannot attend, send me an email to let me know which other lab you'd like to attend; the timetable is here.

Boolean Expressions

Programs that are Sensitive to Information

Up until now, our programs work top-down with no nuance — no section of our code is skipped, we cannot set aside a procedure for a particular value, and we cannot check that our values are all 'valid'.

Goal for Today

Create programs that allow us to give different answers depending on the input information.

- We need our program to make choices.
- How do we differentiate between values? We need some notion of *comparison* — see the next slide for the answer: boolean expressions.

Boolean Operators

`>`	Greater than	`<`	Less than
`>=`	Greater than or equal to	`<=`	Less than or equal to
`==`	Equal to	`not`	Inverse

- All of these evaluate to either `True` or `False`.
- **Examples:**
  - `15 > 10` is `True`
  - `12 <= 10` is `False`
  - `"string" == "string"` is `True`
  - `not(15 > 10)` is `False`

Boolean Conjunctives

- We can chain together boolean statements using the conjunctives `and` or `or`.
- **Examples:**
  - `(15 > 10) and (12 <= 10)` returns `False`
  - `(15 > 10) or (12 <= 10)` returns `True`
  - `("string" == "string") and (15 > 10)` returns `True`

Distributing a `not` Over a Conjunctive

De Morgan's Laws

Let a and b be boolean expressions. Then

$$\texttt{not}(a \texttt{ and } b) == \texttt{not}(a) \texttt{ or } \texttt{not}(b)$$

and

$$\texttt{not}(a \texttt{ or } b) == \texttt{not}(a) \texttt{ and } \texttt{not}(b).$$

- Takeaway: to distribute, push the `not` to each expression, and then swap every conjunctive.

Examples of Distributing `not`

- Example 1:

```
not((5 < x) and (x < 7))
```

  can be written as

```
not(5 < x) or not(x < 7)
```

  which can be simplified to

```
(5 >= x) or (x >= 7)
```

- Example 2: Similarly, we can simplify

```
not((x != y) and (z == y))
```

  eventually to

```
(x == y) or (z != y)
```

If/Elif/Else Statements

If Statements

- We can now use boolean expressions to control the flow of our program.
- We do this using an `if` statement.
  - Structure:

```
if boolean_expression is True:
    # do this
# move on
```

- The code that executes if the `boolean_expression` is `True` needs to be indented (4 spaces/1 tab).
- The rest of the code below is executed whether the `boolean_expression` is `True` or `False`.

Isolated If-Statements

- An isolated if-statement will still allow the program to execute all code below the indented block after it completes.
- What if we want to skip the rest of the code if the if-statement is executed?

If/Else Statements

- We do this using the `if-else` keywords.
  - Structure:

```
if boolean_expression is True:
    # do this
else:
    # do this instead
```

Isolated If/Else Statements

- Sometimes if-else statements are not enough to cover all the branches we want to handle.
- We can create more branches by using an elif statement.

If/Elif/Else

- We can create an arbitrary number of branches by sandwiching `elif` between an `if` and an `else`.
- Structure:

```
if boolean_expression_one is True:
    # do this
elif boolean_expression_two is True:
    # do this instead
elif boolean_expression_three is True:
    # do this instead
...
else:
    # do this instead of all the above
```

Example of If/Elif/Else Statement

- This program will print `"Less than 20, not less than 10"` .

```
x = 15

if x < 10:
    print("Less than 10")
elif x < 20:
    print("Less than 20, not less than 10")
else:
    print("Greater than or equal to 20")
```

Constructing an If/Elif/Else Block

- Over the next few slides, we will build a program that takes a mark from the user and assigns a grade.
- We will aim to minimize repetition, redundancy, and errors as much as possible.

The Input Function

- Before we move on, we need to look at a very important function that often accompanies an if/elif/else block.
- The input function is motivated by the question: if we have different branches for different inputs, how do we actually allow different inputs?
- `input(prompt)`
  - Takes input from the user in the console after the prompt.
  - Its output is a `string`.
- Usage examples:
  - `Letter = input("Give me a letter")`
  - `number = float(input("Give me a number"))`
  - ```
    score = int(input("Give me the number of "
                      "questions you got right"))
    ```
    ```

Attempt 1: Edge cases

- What happens if I put a grade of -2 below? or a grade of 200?

```
grade = float(input("What is your grade?"))

if grade < 50:
    print("fail")
elif grade < 85:
    print("distinction")
else:
    print("high distinction")
```

Attempt 2: Repetition

- How can the following be improved?

```
grade = float(input("What is your grade?"))

if grade < 0:
    print("haha nice one")
elif grade < 50:
    print("fail")
elif grade < 85:
    print("distinction")
else grade < 100:
    print("high distinction")
else:
    print("haha nice one")
```

Attempt 3: Redundancy

- Is every `or` / `and` necessary?

```
grade = float(input("What is your grade?"))

if grade < 0 or grade > 100:
    print("haha nice one")
elif grade >= 0 and grade < 50:
    print("fail")
elif grade >= 50 and grade < 85:
    print("distinction")
elif grade >= 85 and grade < 100:
    print("high distinction")
```

Final Design

```
grade = float(input("What is your grade?"))

if grade < 0 or grade > 100:
    print("haha nice one")

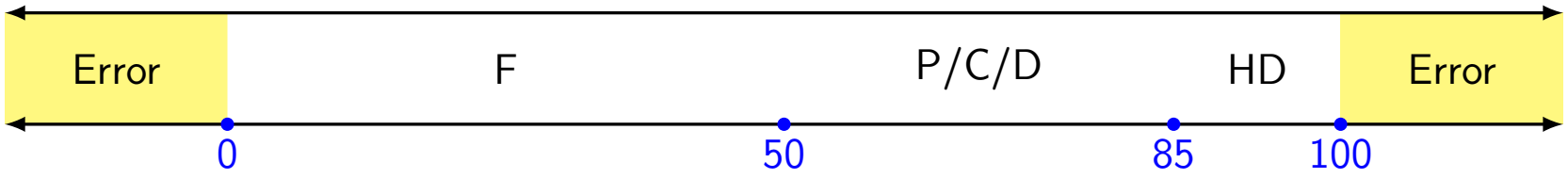
elif grade < 50:
    print("fail")

elif grade < 85:
    print("distinction")

else:
    print("high distinction")
```

On the next slides, we'll look at how this works visually.

Final Design




```
grade = float(input("What is your grade?"))

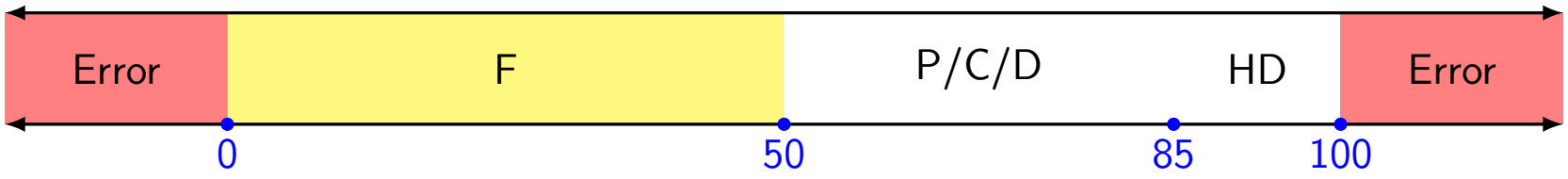
if grade < 0 or grade > 100:
    print("haha nice one")

elif grade < 50:
    print("fail")

elif grade < 85:
    print("distinction")

else:
    print("high distinction")
```

Final Design




```
grade = float(input("What is your grade?"))

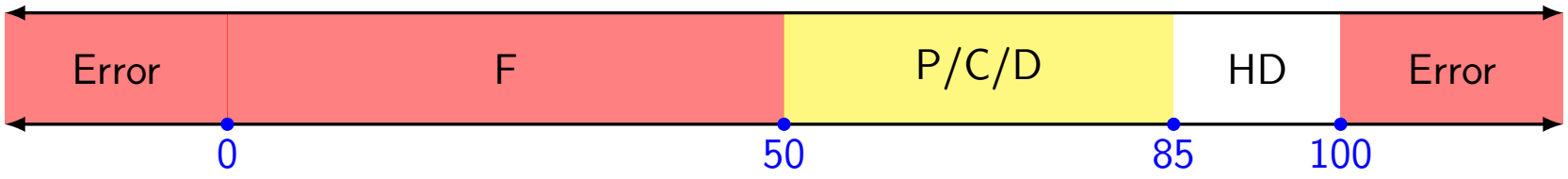
if grade < 0 or grade > 100:
    print("haha nice one")

elif grade < 50:
    print("fail")

elif grade < 85:
    print("distinction")

else:
    print("high distinction")
```

Final Design




```
grade = float(input("What is your grade?"))

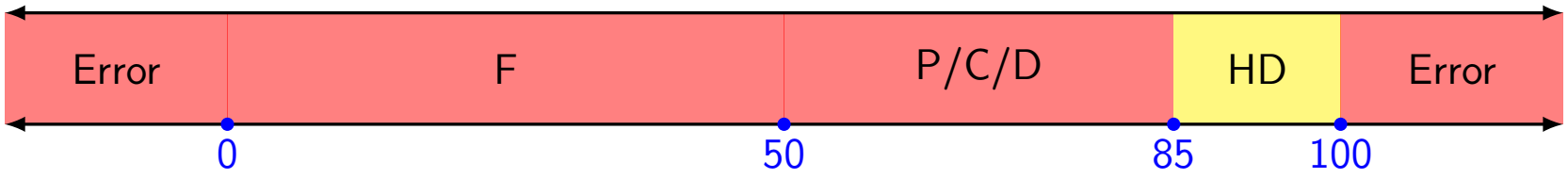
if grade < 0 or grade > 100:
    print("haha nice one")

elif grade < 50:
    print("fail")

elif grade < 85:
    print("distinction")

else:
    print("high distinction")
```

Final Design




```
grade = float(input("What is your grade?"))

if grade < 0 or grade > 100:
    print("haha nice one")

elif grade < 50:
    print("fail")

elif grade < 85:
    print("distinction")

else:
    print("high distinction")
```

Plotting

Plotting

- Python by itself cannot plot
- Need to use a library to help
- ```
  import matplotlib.pyplot as plt
  plt.figure()                    # Creates a new figure
  plt.plot(x_data, y_data)        # Adds x and y data
  plt.xlabel("X label")           # Label for x-axis
  plt.ylabel("Y label")           # Label for y-axis
  plt.title("Title")              # Adds a title
  ```
  ```

Using the matplotlib Library

- You will need to plot some bearings and their corresponding angles in Exercise 1, which can be easily adapted from the format in the previous slide.
- A code example of using `matplotlib` can be found in the Week 3 code on our GitHub repository for this lab.

Lab Tips

Lab Tips

- Exercise 1
  - Use any method or program to work out the equations; `matplotlib` is recommended.
  - Ensure your script works for decimals, not just integers.
  - A bearing of $270°$ corresponds to an angle of $+180°$.
  - Check that the bearing is in the $[0, 360]$ range.
- Exercise 2
  - Avoid redundant checks in subsequent steps.
- Exercise 3
  - Process layers top-to-bottom or bottom-to-top.
  - Minimize redundancy and edge cases.
- Remember to test your code!

Feedback

Feel free to provide anonymous feedback about the lab!

Feedback Form