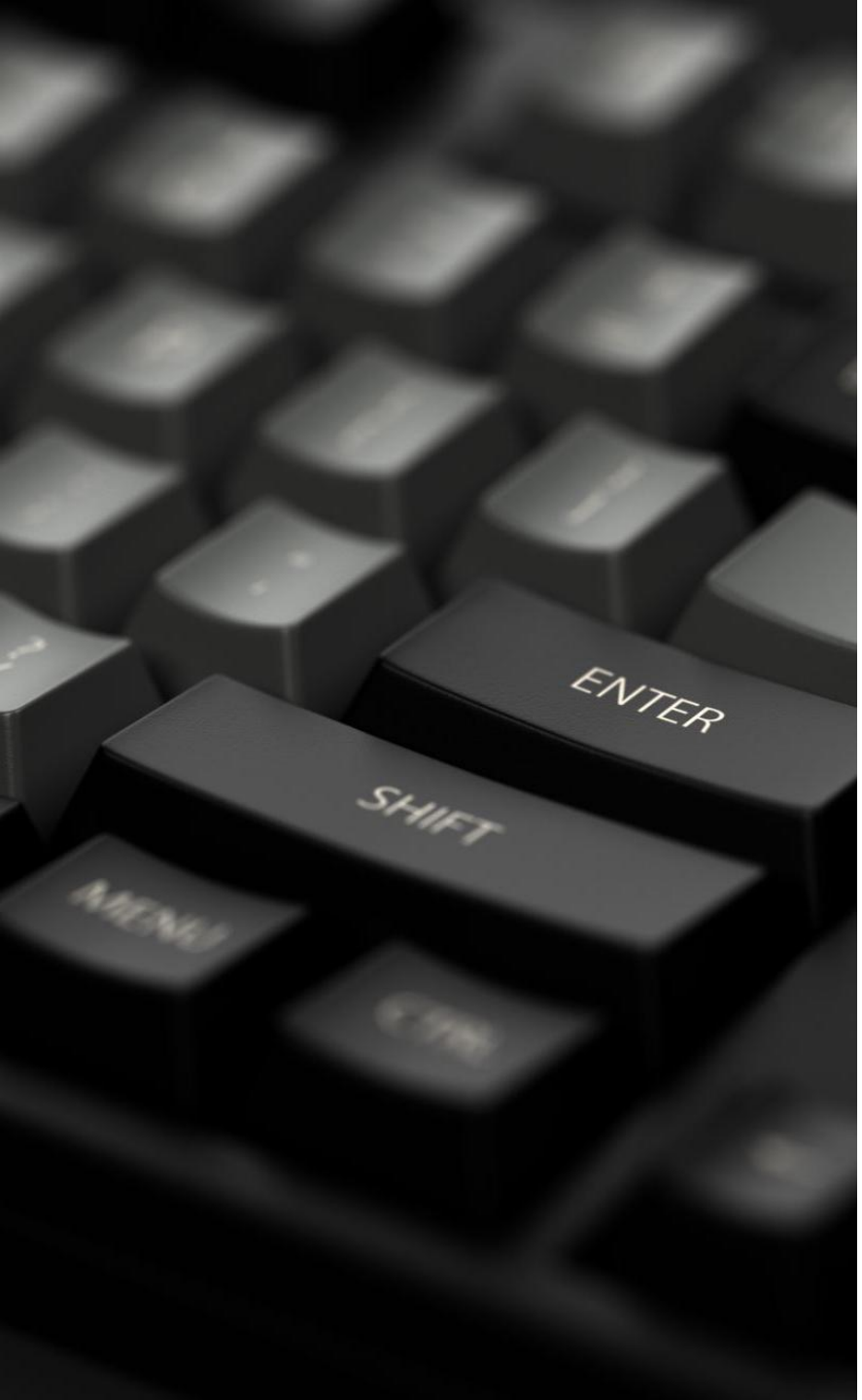
A gamified introduction to Python Programming

Lecture 4
Conditional statements (If/Elif/Else)

Matthieu DE MARI – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



Outline (Chapter 4)

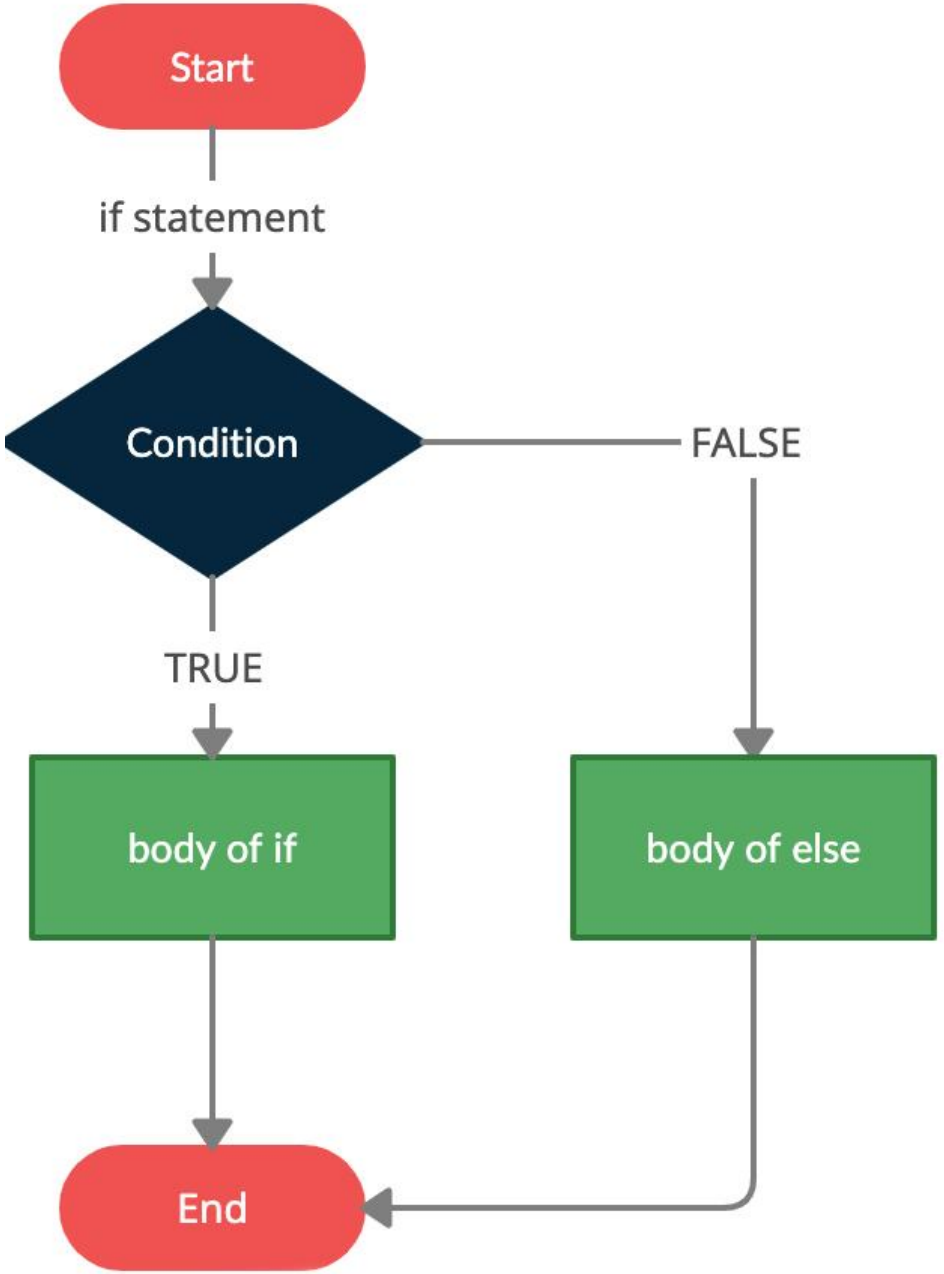
- What are **conditional statements**?
- How to use the **if** statement?
- How to use the **elif** statement?
- How to use the **else** statement?
- What is **dead code?**
- What are **good practices** when it comes to conditional statements?
- What are **nested ifs** statements?
- **Practice** on if/elif/else.

Conditional structures

Definition (conditional structure):

In programming, a **conditional structure** (or **statement**) is a control flow statement that **executes a block of code if and only if a specified condition or Boolean variable evaluates to True**.

Using keywords such as "**if**", "**elif**", and "**else**" to define different execution paths, this mechanism allows programs to make decisions and perform different actions based on given conditions.



The if statement

The **if** statement is the simplest **<u>conditional structure</u>**.

- **Structure:**
  - Use the keyword **if**,

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean variable** or write an **expression that returns a Boolean**,

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean variable** or write an **expression that returns a Boolean**,
  - Add a **colon symbol** (:) after the Boolean term,

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean variable** or write an **expression that returns a Boolean**,
  - Add a **colon symbol** (:) after the Boolean term,
  - Add a block of instructions **inside** the **if** statement, which will be executed if and only if the Boolean is **True** (indented as in **def**).

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **Structure:**
  - Use the keyword **if**,
  - Immediately after, pass a **Boolean variable** or write an **expression that returns a Boolean**,
  - Add a **colon symbol** (:) after the Boolean term,
  - Add a block of instructions **inside** the **if** statement, which will be executed if and only if the Boolean is **True** (indented as in **def**).

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

Note: "inside" means your instructions are **indented** with 4 spaces more than the if statement. Jupyter will suggest indentations.

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition/variable specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```python
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition/variable specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition/variable specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition/variable specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```python
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

The if statement

The **if** statement is the simplest **conditional structure**.

- **How it works:**
  - If the Boolean condition/variable specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```python
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

```
This will be printed if condition is set to True.
It will not print if condition is set to False.
This will be printed: not indented, outside of the if statement.
```

The if statement

The **if** statement is the simplest **<u>conditional structure</u>**.

- **How it works:**
  - If the Boolean condition/variable specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **if** statement.
  - Once we are done executing the code in **if** (or ignoring it), move on to the next (non-indented) line.

```python
condition = True # or False value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

```
This will be printed if condition is set to True.
It will not print if condition is set to False.
This will be printed: not indented, outside of the if statement.
```

```python
condition = False # or True value
if(condition):
    print("This will be printed if condition is set to True.")
    print("It will not print if condition is set to False.")
print("This will be printed: not indented, outside of the if statement.")
```

```
This will be printed: not indented, outside of the if statement.
```

slido

Please download and install the Slido app on all computers you use

What will be printed when we run this code?

ⓘ Start presenting to display the poll results on this slide.

The elif statement

The **elif** statement (short for “else-if”) is used to define another conditional test to be executed, if and only if the previous **if** statement has failed.

- **Structure:**
  - Write your **if** block as before
  - **<u>On the same indentation level</u>** as your **if** statement, write you **elif** statement (elif + Boolean condition + colon symbol)
  - Add your instructions inside the **elif**, by indenting your code as in **if**.

```
bool1 = True # or False value
bool2 = True # or False value
if(bool1):
    print("Do something.")
elif(bool2):
    print("Do something else.")
```

The elif statement

How it works:

- If the Boolean in the **if** statement is **True**, execute the code inside the **if**, ignore the **elif**.

```
# Some booleans
bool1 = True
bool2 = True
# If statement, with True boolean condition
if(bool1):
    print("1. This will be printed, because bool1 is True.")
# Elif statement, with True boolean condition
elif(bool2):
    print("2. This will NOT be printed, because the first if block was executed.")
```

Did not execute even though bool2 was True.

```
1. This will be printed, because bool1 is True.
```

The elif statement

How it works:

- If the Boolean in the **if** statement is **True**, execute the code inside the **if**, ignore the **elif**.
- **Otherwise,** check for the Boolean in **elif**, and execute the code indented inside the **elif**, if this second Boolean condition is **True**. Otherwise, ignore it.

```
# Some booleans
bool1 = False
bool2 = True
# If statement, with False boolean condition
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
# Elif statement, with True boolean condition
elif(bool2):
    print("2. This will be printed, because the first if block was not executed and bool2 is True.")
```

```
2. This will be printed, because the first if block was not executed and bool2 is True.
```

The elif statement (multiple blocks)

Multiple **elif** statements can be added after a single **if** statement.

- In this case, execute the code inside an **elif**, if and only if:
  - all the previous **if/elif** have failed,
  - and its Boolean condition is **True**.

```
# Some booleans
bool1 = False
bool2 = True
bool3 = True
# If statement, with False boolean condition
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
# Elif statement, with True boolean condition
elif(bool2):
    print("2. This will be printed, because the first if block was not executed and bool2 is True.")
# Another elif statement, with True boolean condition
elif(bool3):
    print("3. This will NOT be printed, because the previous block was exectued.")
```

```
2. This will be printed, because the first if block was not executed and bool2 is True.
```

The elif statement (multiple blocks)

Multiple **elif** statements can be added after a single **if** statement.

- In this case, execute the code inside an **elif**, if and only if:
  - all the previous **if/elif** have failed,
  - and its Boolean condition is **True**.

```
# Some booleans
bool1 = False
bool2 = False
bool3 = True
# If statement, with False boolean condition
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
# Elif statement, with False boolean condition
elif(bool2):
    print("2. This will NOT be printed, because bool2 is also False.")
# Another elif statement, with True boolean condition
elif(bool3):
    print("3. This will be printed, because none of the previous blocks were executed and bool3 is True.")
```

```
3. This will be printed, because none of the previous blocks were executed and bool3 is True.
```

An example of if/elif code

Example: write a code that receives a number x, and prints one of the following prompts, accordingly:

- "x is strictly positive."
- "x is strictly negative."
- "x is zero."

```
# A number x
x = 10
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
elif(x==0):
    print("The number x is zero.")
```

```
The number x is strictly positive.
```

We can use the **if/elif** structure to program that!

slido

Please download and install the Slido app on all computers you use

What will be printed when running this code?

ⓘ Start presenting to display the poll results on this slide.

Dead code

Definition (dead code):
We call "**dead code**" a piece of code that was written but is never going to be executed. Often, due to bad structure in code.

```
if(x > 10):
    print("Hello!")
elif(x > 12):
    print("World!")
```

- **Question:** can you spot the line, which will never be executed, no matter what the value of **x** is?
- Why is it dead code?

Dead code

Definition (dead code):
We call "**dead code**" a piece of code that was written but is never going to be executed. Often, due to bad structure in code.

- **Question:** can you spot the line, which will never be executed, no matter what the value of **x** is?
- Why is it dead code?

```
if(x > 10):
    print("Hello!")
elif(x > 12):
    print("World!")
```

Dead code

Reason: Variable **x** cannot be both lower than 10 and greater than 12. We need the **if** block to fail, for the **elif** to be checked. It means **x** must be lower than 10. But then, passing the Boolean condition in the **elif** requires having **x** greater than 12. Impossible conditions.

Dead code

Definition (dead code):
We call "**dead code**" a piece of code that was written but is never going to be executed. Often, due to bad structure in code.

- **Question:** can you spot the line, which will never be executed, no matter what the value of **x** is?
- Why is it dead code?

The life of a CS teacher reviewing student codes


```
if(x > 10):
    print("Hello!")
elif(x > 12):
    print("World!")
```

Matt’s Great advice

Matt’s Great Advice: Avoid dead code, by drawing structural diagrams and spending time figuring out the logic using pen and paper.

Dead code, usually follow from a **poor design** in your code.

Drawing a **structural diagram**, **<u>before coding</u>**, greatly helps figuring out the right structure for your code and avoid dead code.

There is never a scenario where we want our code to contain dead code.

The else statement (no elif example)

The **else** statement is used to define a block of code to execute, if and only if **ALL** the previous **if/elif** statement have failed.

Same structure as an **elif**, but…

- **Comes last**, after all the **if/elif** statements.
- **No Boolean condition** to be checked.

```
bool1 = True
if(bool1):
    print("1. This will be printed, because bool1 is True.")
else:
    print("2. This will NOT be printed, because the previous block was executed.")
```

```
1. This will be printed, because bool1 is True.
```

```
bool1 = False
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
else:
    print("2. This will be printed, because none of the previous blocks were executed.")
```

```
2. This will be printed, because none of the previous blocks were executed.
```

The else statement (multiple elif example)

```python
bool1 = True
bool2 = True
if(bool1):
    print("1. This will be printed, because bool1 is True.")
elif(bool2):
    print("2. This will NOT be printed, because the previous block was executed.")
else:
    print("3. This will NOT be printed, because the first block was executed.")
```

```
1. This will be printed, because bool1 is True.
```

```python
bool1 = False
bool2 = True
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
elif(bool2):
    print("2. This will be printed, because the first block was not executed and bool2 is True.")
else:
    print("3. This will NOT be printed, because the second block was executed.")
```

```
2. This will be printed, because the first block was not executed and bool2 is True.
```

```python
bool1 = False
bool2 = False
if(bool1):
    print("1. This will NOT be printed, because bool1 is False.")
elif(bool2):
    print("2. This will NOT be printed, because bool2 is False.")
else:
    print("3. This will be printed, because none of the previous blocks were executed.")
```

```
3. This will be printed, because none of the previous blocks were executed.
```

Our previous if/elif example, turned into an if/elif/else example

```python
# A number x
x = 10
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
elif(x==0):
    print("The number x is zero.")
```

```
The number x is strictly positive.
```

```python
# A number x
x = 10
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
else:
    print("The number x is zero.")
```

```
The number x is strictly positive.
```

Technically, these two code on the right are equivalent

- If we reach the last **elif**, that means the first two Boolean questions (x>0 and x<0) have failed.
- At this point, we are guaranteed that x is probably zero.
- It can then be replaced with an **else**.

Our previous if/elif example, turned into an if/elif/else example

```python
# A number x
x = 10
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
elif(x==0):
    print("The number x is zero.")
```

```
The number x is strictly positive.
```

```python
# A number x
x = -5
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
else:
    print("The number x is zero.")
```

```
The number x is strictly negative.
```

```python
# A number x
x = 10
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
else:
    print("The number x is zero.")
```

```
The number x is strictly positive.
```

```python
# A number x
x = 0
# An if/elif/else statement
if(x>0):
    print("The number x is strictly positive.")
elif(x<0):
    print("The number x is strictly negative.")
else:
    print("The number x is zero.")
```

```
The number x is zero.
```

slido

Please download and install the Slido app on all computers you use

How many elif statements can you use in an if/elif/else structure in Python?

ⓘ Start presenting to display the poll results on this slide.

Practice activities for if/elif/else

Let us practice the **if/elif/else** concepts a bit, with two activities.

Activity 1 – Ask for user's age.ipynb

Activity 2 - Strength to lifepoints.ipynb

Activity 1 – Ask for user’s age

Write a function **ask_user_age()**, as described below.

- It **receives no parameters** and **returns no parameters**.
- It first **asks for the user to input its age**, and retrieves the info from the user.
- **If the age is negative** (0 included), the function should **print** a message that reads "Your age cannot be negative, it must be at least 1."
- **If the age given by the user is larger than 122** (oldest person on record, Jeanne Calment), then the **print** should display "I really doubt you are ___ years old..." with the blank filled accordingly.
- **Otherwise**, the function should print "Oh, you are ___ years old? That's cool!", with the **blank filled** accordingly.

Activity 2 - Strength to lifepoints

Write a function **strength_to_lifepoints()**, according to the following requirements.

- This function **receives a single parameter, strength_points**, which corresponds to the number of strength points our main character has, and - for simplicity - will only take integer values.
- This function **returns** a **single output, lifepoints**, which corresponds to the number of lifepoints our main character will have, based on its strength points.

- Our main character has **a base number of 50 lifepoints** (that means it has 50 lifepoints, by default, if its strength is zero).
- **For each strength point**, our hero will **gain 10 extra lifepoints**.
- If the main character has **at least 50 strength points**, it gains a **one-time bonus of 100 lifepoints**, on top of the lifepoints it already has.
- Finally, if the main character has **at least 100 strength points**, it gains another **one-time bonus of 50% extra lifepoints**, on top of all the lifepoints it already has and the previous bonuses.

Nested if structures

Definition (nested if structure):
A **nested if structure** is a structure which includes one or multiple **if** statement(s), inside another **if** statement.

These are typically used to check additional/subsequent conditions, based on whether a first condition has been satisfied or not.

```python
x = 5
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

```python
x = 0
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

```
The number x is positive.
```

(Probably something we could have used in Activity 2?)

Nested if structures

Definition (nested if structure):
A **nested if structure** is a structure which includes one or multiple **if** statement(s), inside another **if** statement.

These are typically used to check additional/subsequent conditions, based on whether a first condition has been satisfied or not.

Each **if** might have its own **elif**/**else** statements. Careful on the indentation levels for each!

```python
x = 5
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

```python
x = 0
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
```

```
The number x is positive.
```

```python
x = -2
if(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print("The number x is NOT positive.")
```

```
The number x is NOT positive.
```

Nested if structures

```
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

Nested **if** structures vs. combined conditionals

Nested **if** structures can, most of the time, be rewritten with combined conditionals (using **and**/**or** Boolean operators).

For instance, both structures on the right are equivalent.

→ Which design is better?

At this point, it is a matter of personal preference, use whatever feels more natural!

```python
x = 5
if(x==0):
    print("The number x is zero.")
elif(x>=0):
    print("The number x is positive.")
    if(x>0):
        print("In fact, the number x is STRICTLY positive.")
else:
    print(("The number x is negative."))
    if(x<0):
        print("In fact, the number x is STRICTLY negative.")
```

```
The number x is positive.
In fact, the number x is STRICTLY positive.
```

```python
x = 5
if(x==0):
    print("The number x is zero.")
if(x != 0 and x>=0):
    print("The number x is non-zero and positive.")
if(x>0):
    print("In fact, the number x is STRICTLY positive.")
if(x != 0 and x<=0):
    print(("The number x is non-zero and negative."))
if(x<0):
    print("In fact, the number x is STRICTLY negative.")
```

```
The number x is non-zero and positive.
In fact, the number x is STRICTLY positive.
```

The ternary operator

Definition (ternary operator):

The **ternary operator** is a concise way to write some conditional expressions. Often used to dynamically decide on which value to assign to a variable based on certain conditions being met.

It is defined as "**x = a if c else b**". Its effect is to assign the value a to x if c is **True**; and the value of b to x if c is **False**.

It can be seen as a short, condensed version of its equivalent indented if/else statement.

```
a, b, c = 3, 5, True
# A simple if statement
if(c):
    var = a
else:
    var = b
print(var)
# Its ternary equivalent
var = a if c else b
print(var)
```

```
3
3
```

```
a, b, c = 3, 5, False
# A simple if statement
if(c):
    var = a
else:
    var = b
print(var)
# Its ternary equivalent
var = a if c else b
print(var)
```

```
5
5
```

Quick note on If/Else vs. Try/Except

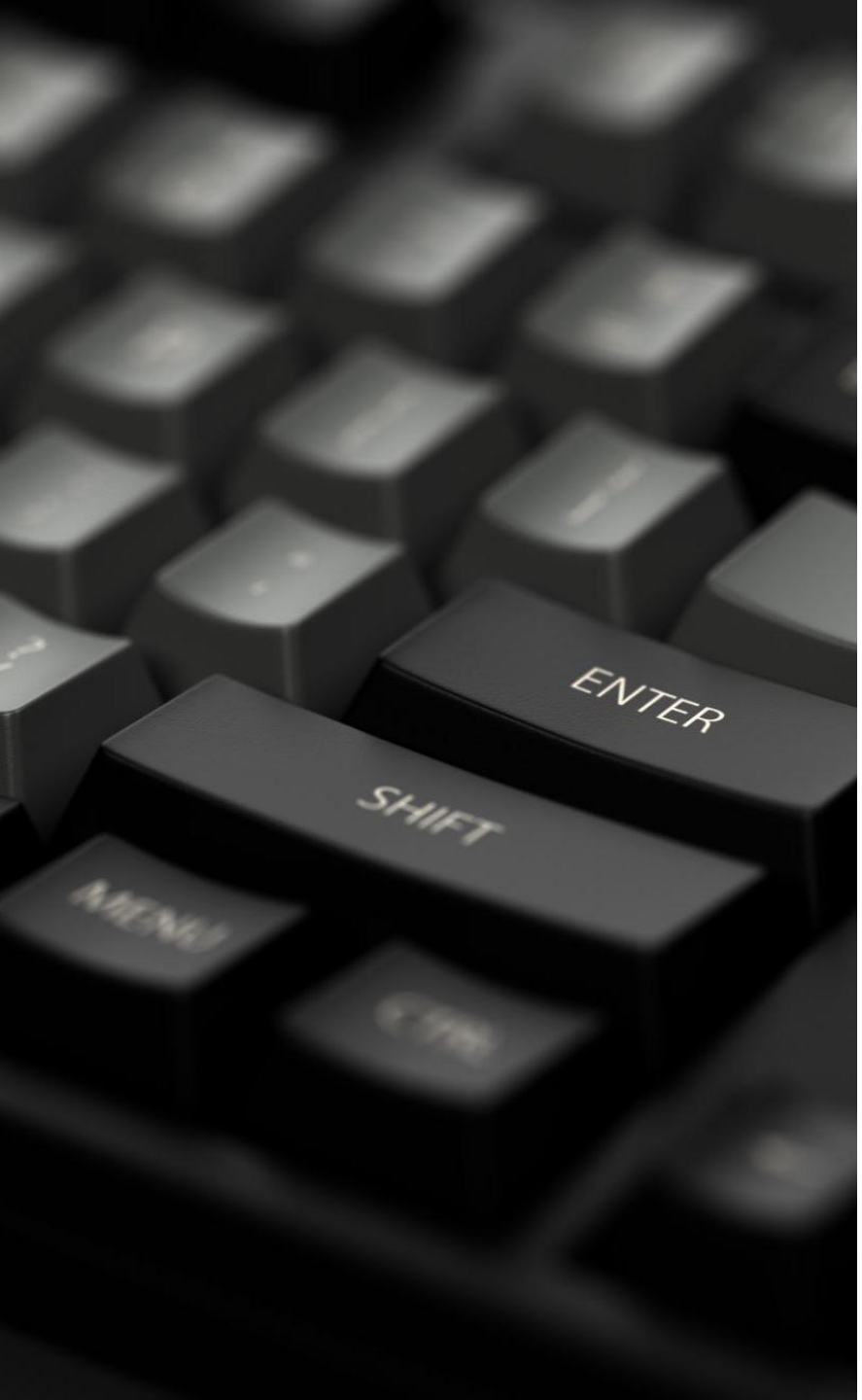
Bad practice: Using try/except instead of if/else conditional structures.

Many beginner students (especially those with background with the C language) use the **try/except** structure in place of **if/else**.

While it might lead to “similar outcomes”, **try/except has a different purpose in Python**, which is **error handling**.

For now, we should consider it malpractice. More details on how to properly use this structure in an upcoming lecture.

```python
# No need to worry about this code.
# Just using it for illustration.
try:
    user_input = int(input("Enter a number: "))
    if user_input > 0:
        print("Positive number")
    elif user_input < 0:
        print("Negative number")
    else:
        print("Zero")
except ValueError:
    print("That's not a valid number!")
```



Conclusion (Chapter 4)

- What are **conditional statements**?
- How to use the **if** statement?
- How to use the **elif** statement?
- How to use the **else** statement?
- What is **dead code?**
- What are **good practices** when it comes to conditional statements?
- What are **nested ifs** statements?
- **Practice** on if/elif/else.

Activity 3 - Race and class check

Write a function **character_creation()**, according to the following requirements.

- The function will **receive two parameters**: **user_race** and **user_class**.
- For simplicity, only **three races** are available: **Human, Elf,** and **Dwarf**.
- For simplicity, **only four classes** are available: **Warrior, Hunter, Mage** and **Priest**.
- **Humans** can play **all classes**.
- **Elves** cannot be **warriors**.
- **Dwarves** cannot be **mages or priests**.
- The function should **not return anything.**
- It should **print** "You cannot play a character that is ...{race} and ...{class}.", with **blanks filled** accordingly, **if the combination of user_race and user_class is not acceptable**.
- Not acceptable here means that its race and/or class is not among the ones listed above, or the combination is not permitted, as listed above.
- **If the combination is valid**, it should **print** "Your character's race is ...{race} and your character's class is ...{class}.", with blanks filled accordingly.

Up for a challenge?
(in the Extra challenges folder)

Activity 2+ - Strength to lifepoints (extra challenge).ipynb

- Redo the activity 2, but this time…
- Do not use any conditional statement (**if**/**while**) or ternary operator.
- The function should only contain one line, which starts with **return**.