ILP 2020 – W2S2
If/Elif/Else statements, While/Break statements

Matthieu DE MARI – Singapore University of Technology and Design

Outline (Week2, Session2 – W2S2)

- The if statement
- The elif statement
- The else statement
- Dead code and code structure
- Nested ifs
- While statements
- Infinite looks and how to kill them
- The break statement

The if statement

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

The if statement

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

- **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** 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** 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** 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**.

```
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** 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**.

```
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 instruction is **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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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**.

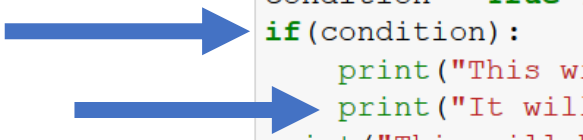
- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.



```
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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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 specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.

```
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 **conditional structure**.

- **How it works:**
  - If the Boolean condition 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.

```
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.
```

```
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.
```

The if statement

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

- **How it works:**
  - If the Boolean condition 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.")
```

```
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.
```

```
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.
```

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.

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
  - **On the same indentation level** 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.")
```

```
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**.

Did not execute despite bool2 being True.

```
# 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.")
```

```
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 an initial **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 an initial **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.”

Let us use the **if/elif** structure to program that!

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.
```

Let us use the if/elif structure to program that!

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.

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

- **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

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, but **x** is already lower than 10.

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.

Dead code is often due to **bad structure/design** in the code.

Be careful!


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

The **else** statement

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.

The **else** statement

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.

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.

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

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

```
1 bool1 = False
2 if(bool1):
3     print("1. This will NOT be printed, because bool1 is False.")
4 else:
5     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)

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

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

```
1  bool1 = False
2  bool2 = True
3  if(bool1):
4      print("1. This will NOT be printed, because bool1 is False.")
5  elif(bool2):
6      print("2. This will be printed, because the first block was not executed and bool2 is True.")
7  else:
8      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.
```

```
1  bool1 = False
2  bool2 = False
3  if(bool1):
4      print("1. This will NOT be printed, because bool1 is False.")
5  elif(bool2):
6      print("2. This will NOT be printed, because bool2 is False.")
7  else:
8      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

```
1  # A number x
2  x = 10
3  # An if/elif/else statement
4  if(x>0):
5      print("The number x is strictly positive.")
6  elif(x<0):
7      print("The number x is strictly negative.")
8  elif(x==0):
9      print("The number x is zero.")
```

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

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

```
1  # A number x
2  x = 10
3  # An if/elif/else statement
4  if(x>0):
5      print("The number x is strictly positive.")
6  elif(x<0):
7      print("The number x is strictly negative.")
8  elif(x==0):
9      print("The number x is zero.")
```

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

```
1  # A number x
2  x = 10
3  # An if/elif/else statement
4  if(x>0):
5      print("The number x is strictly positive.")
6  elif(x<0):
7      print("The number x is strictly negative.")
8  else:
9      print("The number x is zero.")
```

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

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

```
# 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.
```

```
# 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.
```

```
# 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.
```

```
# 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.
```

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[1]), 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.

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.

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 conditions, based on whether another condition has been satisfied or not.

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

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

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

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

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 conditions, based on whether another condition has been satisfied or not.

Each **if** might have its own **elif**/**else** statements, placed on the same indentation level.

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

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

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

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

```
1  x = -2
2  if(x>=0):
3      print("The number x is positive.")
4      if(x>0):
5          print("In fact, the number x is STRICTLY positive.")
6  else:
7      print("The number x is NOT positive.")
```

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

Nested if structures

```
1  x = 5
2  if(x==0):
3      print("The number x is zero.")
4  elif(x>=0):
5      print("The number x is positive.")
6      if(x>0):
7          print("In fact, the number x is STRICTLY positive.")
8  else:
9      print(("The number x is negative."))
10     if(x<0):
11         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).

```
1  x = 5
2  if(x==0):
3      print("The number x is zero.")
4  elif(x>=0):
5      print("The number x is positive.")
6      if(x>0):
7          print("In fact, the number x is STRICTLY positive.")
8  else:
9      print(("The number x is negative."))
10     if(x<0):
11         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.

```
1  x = 5
2  if(x==0):
3      print("The number x is zero.")
4  elif(x>=0):
5      print("The number x is positive.")
6      if(x>0):
7          print("In fact, the number x is STRICTLY positive.")
8  else:
9      print(("The number x is negative."))
10     if(x<0):
11         print("In fact, the number x is STRICTLY negative.")
```

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

```
1  x = 5
2  if(x==0):
3      print("The number x is zero.")
4  if(x != 0 and x>=0):
5      print("The number x is non-zero and positive.")
6  if(x>0):
7      print("In fact, the number x is STRICTLY positive.")
8  if(x != 0 and x<=0):
9      print(("The number x is non-zero and negative."))
10 if(x<0):
11     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.
```

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.

Personal preference: Whenever possible, try to avoid the nested **if** structures. They are often overly complicated and prone to errors in designing the code.

```
1  x = 5
2  if(x==0):
3      print("The number x is zero.")
4  elif(x>=0):
5      print("The number x is positive.")
6      if(x>0):
7          print("In fact, the number x is STRICTLY positive.")
8  else:
9      print(("The number x is negative."))
10     if(x<0):
11         print("In fact, the number x is STRICTLY negative.")
```

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

```
1  x = 5
2  if(x==0):
3      print("The number x is zero.")
4  if(x != 0 and x>=0):
5      print("The number x is non-zero and positive.")
6  if(x>0):
7      print("In fact, the number x is STRICTLY positive.")
8  if(x != 0 and x<=0):
9      print(("The number x is non-zero and negative."))
10 if(x<0):
11     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.
```

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.

The **while** statement

The **while** statement is another type of **conditional structure.**

The **while** statement

The **while** statement is another type of **conditional structure.**

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

- **How it works:**
  - If the Boolean condition specified for the **if** statement is **True**, then execute the block of code inside the **if** statement.
  - If the Boolean condition is not **True**, 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.

The **while** statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition **False**, ignore the block of code in the **while** statement.

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

- **How it works:**
  - If the Boolean condition 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.

The **while** statement

The **while** statement is another type of **conditional structure.**

- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**

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

- **How it works:**
  - If the Boolean condition 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.**

The while statement

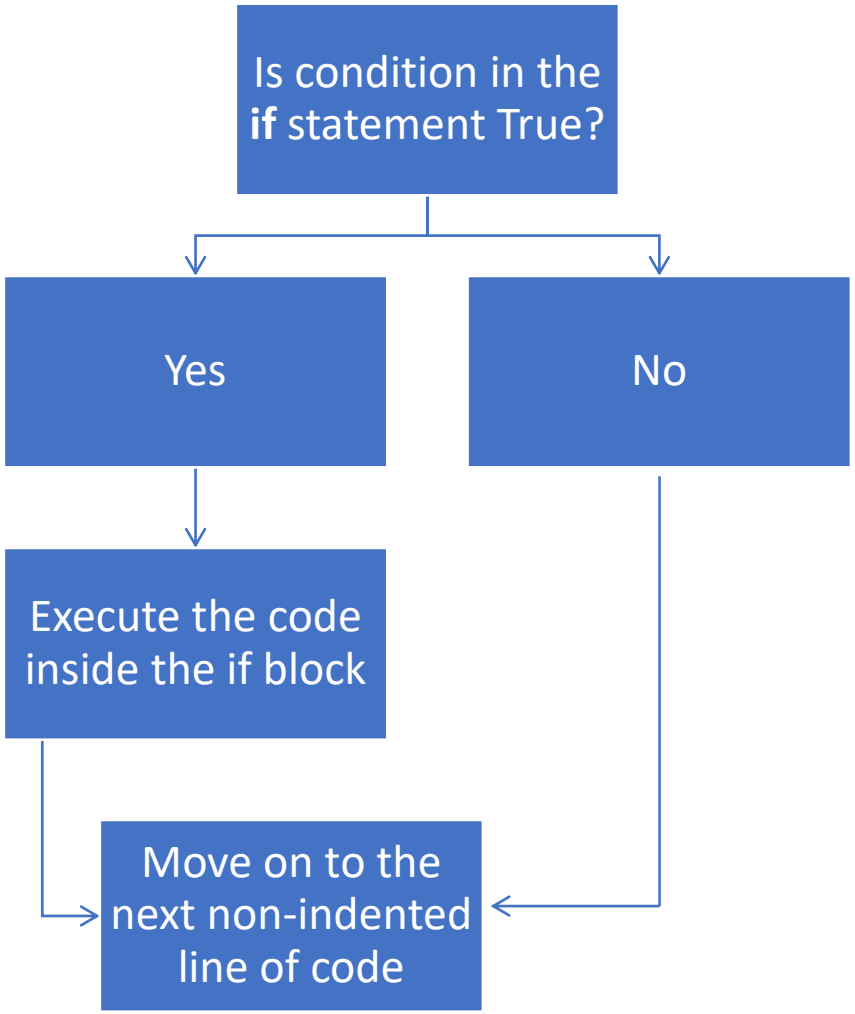
The **while** statement is another type of **conditional structure.**

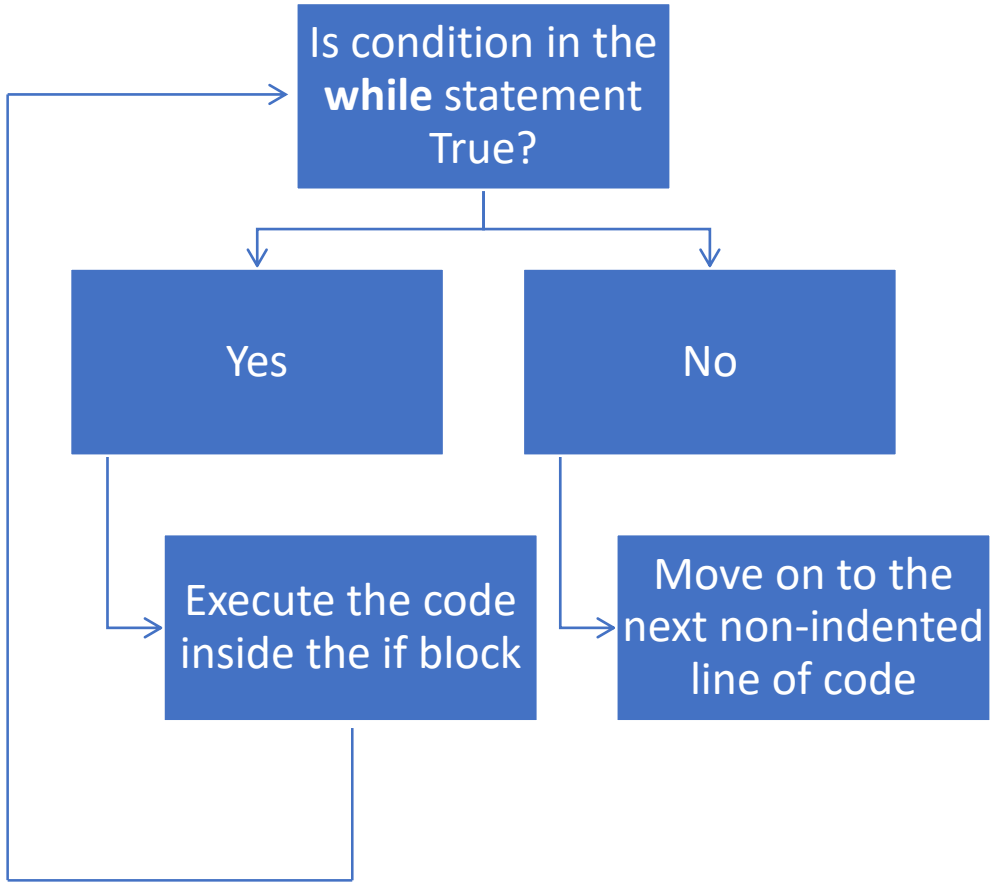
- **How it works:**
  - If the Boolean condition specified for the **while** statement is **True**, then execute the block of code inside the **while** statement.
  - If the Boolean condition is **False**, ignore the block of code in the **while** statement.
  - Once we are done executing the code in **while**, **move back to the while statement, and repeat until the condition is no longer True.**

```
1  # Counting from 1 to 10
2  x = 0
3  print("Counting from 1 to 10...")
4  while(x<10):
5      x = x + 1
6      print(x)
7  print("Done!")
```

```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

Architectures: if vs. while





Infinite loops

The **while** statement repeats a condition until it is no longer **True**.

Infinite loops

The **while** statement repeats a condition until it is no longer **True**.

This means that there should be a clear process that **makes your condition no longer True**, at some point.

```
1  # Counting from 1 to 10
2  x = 0
3  print("Counting from 1 to 10...")
4  while(x<10):
5      x = x + 1
6      print(x)
7  print("Done!")
```

```
Counting from 1 to 10...
1
2
3
4
5
6
7
8
9
10
Done!
```

Infinite loops

The **while** statement repeats a condition until it is no longer **True**.

This means that there should be a clear process that **makes your condition no longer True**, at some point.

Otherwise, the **while** block will keep on repeating indefinitely…
This is called an **infinite loop**.

In [4]:

```
# Counting from 1 to infinity
x = 0
while(x>=0):
    x = x + 1
    print(x)
print("Done!")
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. Your computer runs out of resources (bad thing to do),

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. ~~Your computer runs out of resources (bad thing to do),~~

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done with **CTRL+C** (or **CMD+C** on mac), in console mode and most IDEs.

```
Counting from 1 to infinity...
1
2
3
4
5
6
7
8
9
10
Traceback (most recent call last):
  File ".\infinite_loop.py", line 8, in <module>
    time.sleep(1)
KeyboardInterrupt
```

Infinite loops and how to kill them

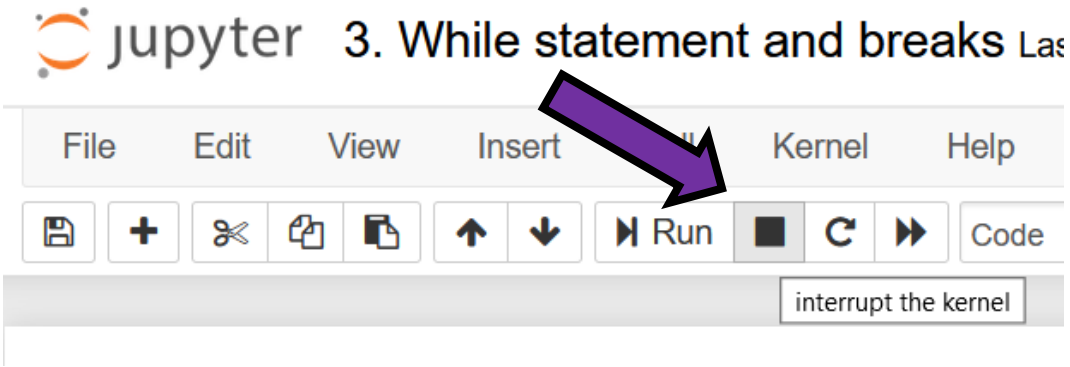
Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done with **CTRL+C** (or **CMD+C** on mac), in console mode and most IDEs.

Or, by using the **stop button** on Jupyter.

```
Counting from 1 to infinity...
1
2
3
4
5
6
7
8
9
10
Traceback (most recent call last):
  File ".\infinite_loop.py", line 8, in <module>
    time.sleep(1)
KeyboardInterrupt
```



Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

This is called a **keyboard interrupt.** It is done with **CTRL+C** (or **CMD+C** on mac), if in console mode and most IDEs.

Or, by using the **stop button** on Jupyter.

Matt’s Great advice #7

Matt’s Great Advice #7: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose, usually follow from a **poor design** in your code.

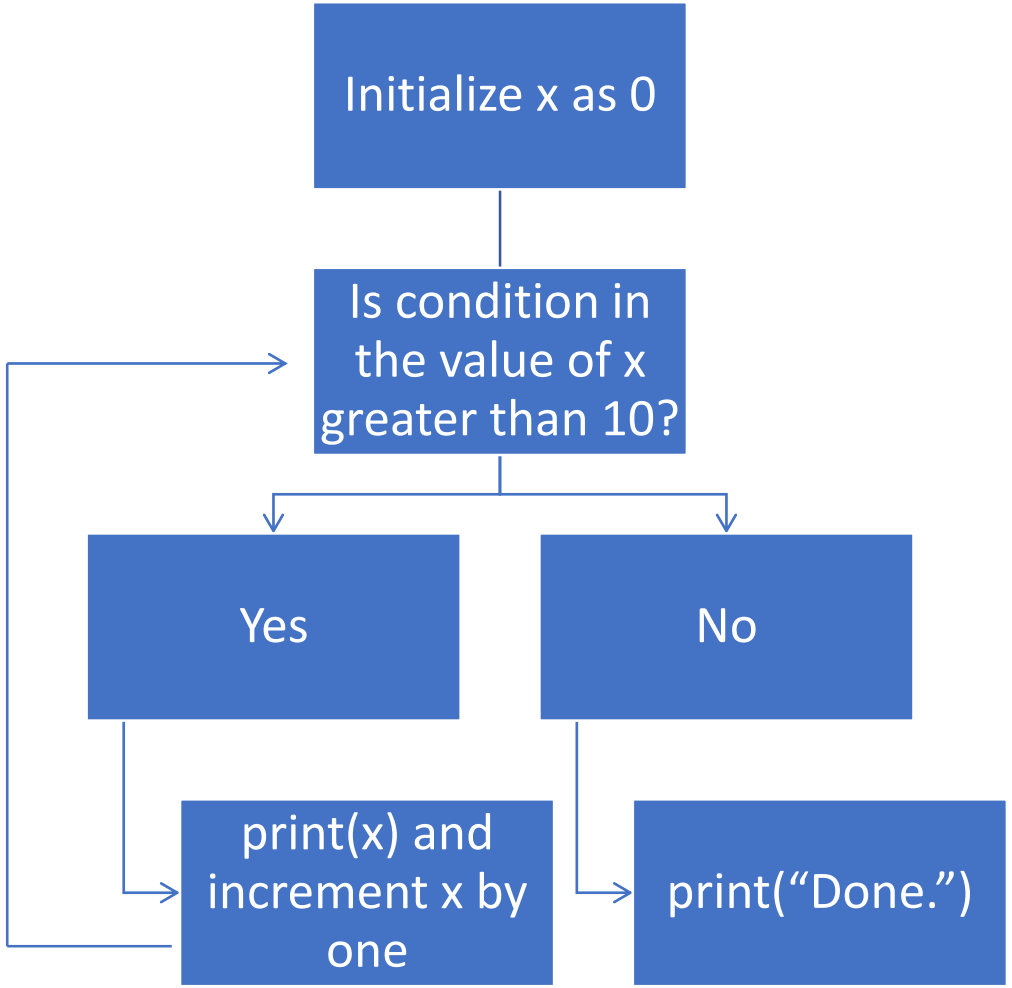
Drawing a **structural diagram**, **before coding**, greatly helps figuring out the right structure for your code.

Matt's Great advice #7

Matt's Great Advice #7: Avoid the infinite loops and dead code, by drawing structural diagrams.

Infinite loops and **dead code**, unless created on purpose, usually follow from a **poor design** in your code.

Drawing a **structural diagram**, **before coding**, greatly helps figuring out the right structure for your code.



Example: diagram for our while loop, counting from 1 to 10.

Infinite loops and how to kill them

Infinite loops will keep on executing forever, unless

1. You decide to crash the program on purpose and kill the loop manually.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

The code then resumes its execution with the next line outside of the **while** block.

Infinite loops: the break statement

Infinite loops will keep on executing forever, unless

2. You use a **break** statement.

When encountered, the **break** statement will immediately end the current **while** loop.

The code then resumes its execution with the next line outside of the **while** block.

```
1  # Counting from 1 to 10, with a break
2  x = 0
3  while(True):
4      x = x + 1
5      print(x)
6      # If x has reached the value 10, break the while loop
7      if(x>=10):
8          break
9          # Careful!
10         print("This is DEAD CODE, because the break is reached before.")
11 print("Done!")
```

```
1
2
3
4
5
6
7
8
9
10
Done!
```

Standard **while** vs. infinite **while** + **break**

1. Standard **while** loop with condition in the while statement.

```
# Counting from 1 to 10
x = 0
print("Counting from 1 to 10...")
while(x<10):
    x = x + 1
    print(x)
print("Done!")
```

2. Infinite **while** loop with condition in an **if** statement, and **break** in the **if** block.

```
# Counting from 1 to 10, with a break
x = 0
while(True):
    x = x + 1
    print(x)
    # If x has reached the value 10,
    # break the while loop
    if(x>=10):
        break
print("Done!")
```

→ Both loops work and do the job, which one is better though?

Matt’s Great advice #8

Matt’s Great Advice #8: Avoid the infinite loops, if possible.

Relying on an **infinite** **while** loop with a **break** is **risky**, and should be avoided when possible.

Matt's Great advice #8

Matt's Great Advice #8: Avoid the infinite loops, if possible.

Relying on an **infinite** **while** loop with a **break** is **risky**, and should be avoided when possible.

It is often easily avoided, by using the Boolean expression of the **if** statement used for **break**, as the condition in the **while** statement.

```
1  # Counting from 1 to 10, with a break
2  x = 0
3  while(True):
4      x = x + 1
5      print(x)
6      # If x has reached the value 10,
7      # break the while loop
8      if(x>=10):
9          break
10 print("Done!")
```

```
1  # Counting from 1 to 10
2  x = 0
3  print("Counting from 1 to 10...")
4  while(x<10):
5      x = x + 1
6      print(x)
7  print("Done!")
```

Matt's Great advice #8

Matt's Great Advice #8: Avoid the infinite loops, if possible.

Relying on an **infinite** **while** loop with a **break** is **risky**, and should be avoided when possible.

It is often easily avoided, by using the Boolean expression of the **if** statement used for **break**, as the condition in the **while** statement.

Note: a few cases, however, require the use of a **break** statement.
For instance, **emergency shutdowns**.

```
1  while(True):
2      print("All systems normal.")
3      print("Running operations as expected.")
4      if(overheating):
5          print("Overheating detected.")
6          print("Engaging emergency shutdown.")
7          break
```

Practice activities for while/break

Let us practice the **while**/**break** concepts a bit, with two activities.

Activity 4 – How many hits can you take.ipynb

Activity 4 - How many hits can you take

Your main character currently has a number of lifepoints, stored in **lifepoints_number**.

Your mentor gives you the following challenge: he will hit you, for a given number of times $\boldsymbol{n}$.

- The **first hit** will make you **lose one lifepoint**,
- the **second**, **two lifepoints**,
- the **third**, **three lifepoints**,
- and **so on**.
- **If** you take too many hits and **your lifepoints fall below zero, you fail the challenge.**
- Assuming you survive after all $\boldsymbol{n}$ hits, your mentor will give you $\boldsymbol{n^2}$ coins.

Write a function, named **maximal_coins_number()**, which

- **receives** your current number of **lifepoints**, i.e. the variable **lifepoints_number**,
- and **returns** the **maximal number of coins** you can hope to obtain from the challenge,
- as well the **number of lifepoints** that will be **remaining after taking the maximal number of hits**.

Conclusion

- The if statement
- The elif statement
- The else statement
- Dead code and code structure
- Nested ifs
- While statements
- Infinite looks and how to kill them
- The break statement

Activity 5 – Guess the number game v2

Remember the guess the number game in W3S1, Activity 1? Back then, we had defined a function **guess_the_number()**,

- which received a **hidden number that the user had to guess** (passed as input **hidden_number**),
- **asked the user to input a number**, via the **input()** method and would store it in a variable **guessed_number**,
- and based on the two numbers would **display two messages**, reading:
  - "You have found the hidden number: True/False."
  - "Your number in guessed_number is lower than the hidden number: True/False.“

Your task is to write a **second version** (v2!) of this function, called **guess_the_number_v2().**

Activity 5 – Guess the number game v2

This v2 function will have the following features, replacing the previous ones:

- The game will **keep on asking the user to input()** values, **until the right number is found**.
- It will **display the message** "Your have found the right number!", **once the user has found the right number.**
- When that happens, it **also displays** "It only took you ... tries!" with the blank filled with the number of times the user had to type a number via input().
- Once the number has been found, the function no longer asks the user for inputs and stops.
- **While the user has not found the right number**, the game will **display either**
  - "Your number is lower than the hidden number." (if the last number entered by the user is lower than the hidden number)
  - or "Your number is higher than the hidden number." (if the last number entered by the user is higher than the hidden number).

Up for a challenge?
(in Extra challenges folder)

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

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

Second challenge: Activity 4+ - How many hits can you take (extra challenge).ipynb

- Similarly, as in 2+ challenge…
- Do not use any conditional statement (**if**/**while**)
- **Hint:** use a bit of maths on sequences!

Up for a challenge?
(in Extra challenges folder)

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

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

Second challenge: Activity 4+ - How many hits can you take (extra challenge).ipynb

- Similarly, as in 2+ challenge…
- Do not use any conditional statement (**if**/**while**)
- **Hint:** use a bit of maths on sequences!

→ Solutions to extra challenges can be found in ./Extra challenges/Extra challenges solutions subfolder.