Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

COMP7004 - Systems Scripting

Lecture 11: Python Flow Control

Dr. Vincent Emeakaroha
vincent.emeakaroha@mtu.ie

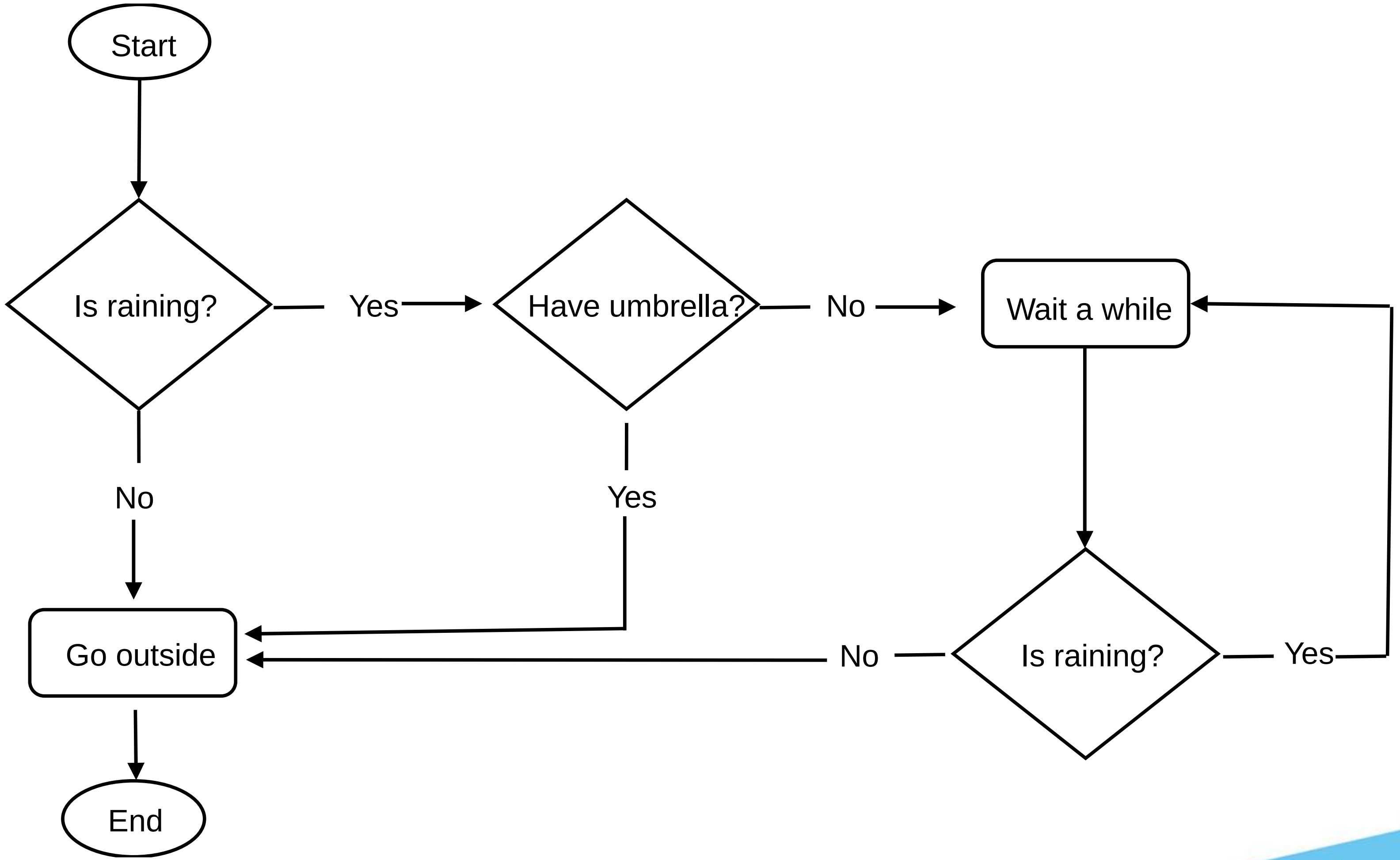
Semester 2, 2025

www.mtu.ie

Lecture Goals

- Flow control concept
- Boolean operators
- Comparison operators
- Elements of flow control
- Flow control statements

Flow Control Concept



Boolean Values

- Boolean data type has only two value
  - True
  - False
- Named after a mathematician
  - George Boole
  - Mathematic Professor in UCC
- Values must be written starting with capital **T** and **F**
  - No quotes necessary as for strings

Boolean Operators

- There are three types of Boolean operators
  - AND
  - OR
  - NOT
- Similar to other operators
  - They evaluate down to a single Boolean value

Boolean Operator Truth Table

AND

Expression	Evaluates To…
True and True	True
True and False	False
False and True	False
False and False	False

OR

Expression	Evaluates To…
True or True	True
True or False	True
False or True	True
False or False	False

NOT

Expression	Evaluates To…
Not True	False
Not False	True

Comparison Operators

- These values evaluates to True or False

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Comparison Operator Example

```
>>> 42 == 42
True
>>> 42 == 99
False
>>> 2 != 3
True
>>> 3 != 3
False
>>>  'hello'  ==  'hello'
True
>>>  'hello'  ==  'Hello'
False
```

Mixing Boolean and Comparison Operators

- Boolean operators operate only on Boolean values
  - True and False
- Comparison operators are not
  - Boolean values
  - But evaluates down to Boolean values
- Example combination

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
```

Elements of Flow Control

- Two important elements of flow control statement are:

1. Conditions
   - Boolean expressions
   - Evaluates down to True or False
   - Decides which action to take
2. Block of Code
   - Lines of codes that can be grouped together
   - Begins with an indentation increase
   - Can contain other blocks
   - Ends when indentation decreases to zero or to a containing block indentation

if Statement

- Syntax
  ```
  if <expression>:
      <code block>
  ```
- Characteristics
  - The `if` keyword
  - A conditional expression
  - A colon
  - Starting on the next line, an indented block of code
- Example
  - if.py

if.py

```
# if statement example

print('who are you?')
name = input()
if name == 'Alice':
        print('Hi, Alice.')
```

else Statements

- An `if` clause can **optionally** be followed by an `else` statement
- Only executed when the `if` condition is `False`
- Characteristics
  - The `else` keyword
  - A colon
  - Starting on the next line and indented block of code
- Example
  - else.py

else.py

```
# else statement example

print('who are you?')
name = input()
if name == 'Alice':
        print('Hi, Alice.')
else:
        print('Hello, Stranger.')
```

elif statement

- A statement that can follow an `if` statement.
  - Full pronouncement is "`else if`".
- Allows multiple execution option rather than only `if` and `else` clause.
  - Provides another condition to be checked when the previous `if` result to `False.`
  - If one conditional expression evaluates to `True`, it gets executed and the rest ignored.
  - No guaranteed execution of particular expression.
- Characteristics
  - The `elif` keyword.
  - A conditional expression that evaluates to `True` or `False.`
  - A colon
  - Starting on the next line an indented block of code.
- Example
  - elif.py

elif.py

```
# elif statement example
print('who are you?')
name = input()
print('What is your age?')
age = int(input())
if name == 'Alice':
        print('Hi, Alice.')
elif age < 12:
        print('You are not Alice, Kiddo.')
elif age > 2000:
        print('Unlike you, Alice is not a Vampire.')
elif age > 100:
        print('You are not Alice, grannie.')
```

elif2.py

- `elif` statement with guaranteed execution of particular expression.
  - By adding an `else` statement

```
# elif statement example with guaranteed execution
print('who are you?')
name = input()
print('What is your age?')
age = int(input())
if name == 'Alice':
        print('Hi, Alice.')
elif age < 12:
        print('You are not Alice, Kiddo.')
else:
        print('You are neither Alice nor a little kid.')
```

Match - Case Statement

- Only available from Python 3.10 version.
- Used to execute statements based on specific pattern values. Often used in place of an if-elif-else statement when there are large number of conditions, especially patterns to match.
- Pattern value can be a number, string, an expression, iterable, class instance, etc.
- Subject value can be any data type
- a _: is used to accept any value not matched with list of values

```
match subject:
    case <pattern1>:
            action 1
    case <pattern2>:
            action 2
    case _:
            action for a no match
```

- One can combine several literals in a single pattern using |
    - Case <pattern1> | <pattern2> | <pattern3>:
        action

Match.py

```python
responseCode = 400

match responseCode:
    case 400:
        print("Error code not recognised.")

    case 401 | 402:
        print("Not found by server.")

    case 500:
        print("Server error.")

    case _:
        print("No valid match")
```

```python
responseCode = 200

match responseCode:
    case 200:
        print("healthy site and reachable")

    case 300:
        print("Seems off-track with access")

    case 400:
        print("Definitly off track")

    case _:
        print("Invalid code. Try again")
```

```python
responseCode = 300

match int(str(responseCode)[0]):
    case 2:
        print("healthy site and reachable")

    case 3:
        print("Seems off-track with access")

    case 4:
        print("Definitly off track")

    case 5:
        print("Not reachable site")
```

while Loop Statement

- Provides the ability to execute a code over and over again
  - As long as the conditional expression is true.
- Characteristics
  - The `while` keyword
  - A conditional expression that evaluates to `True` or `False`
  - A colon
  - Starting on the next line, an indented block of code (called the while clause).
- Similar to `if` statement
  - But the difference is that at the end of a `while` clause, execution jumps back to the beginning of the `while` loop.

while.py

```
# example while loop

count = 0
while count < 5:
        print('Hello, World.')
        count = count + 1
```

Annoying while Loop

- In while loops
  - Poor constructed conditional expression can have annoying consequences.
  - Example:
    - while2.py

```
# example annoying while loop

name = '’
while name != 'your name':
    print('Please type your name.')
    name = input()
print('Thank you!')
```

break Statements

- A shortcut from getting out of a while loop
- When execution reaches a break statement
  - It immediately exits the `while` loop
- Characteristics
  - The `break` keyword
- Example
  - break.py

break.py

```
# example break statement

while True:
        print('Please type your name.')
        name = input()
        if name == 'your name':
                break
print('Thank you!')
```

continue Statement

- Provides the possibility of skipping a loop cycle
  - When continue statement is reached, it terminates the cycle immediately and jump back to the beginning of loop to continue execution.
- Same behaviour as when the execution reaches end of a loop cycle
- Characteristics
  - The `continue` keyword
- Example
  - continue.py

continue.py

```
# example continue statement
while True:
	print('Who are you?')
	name = input()
	if name != 'Vincent':
		continue
	print('Hello, Vincent. What is the password? (It is a fish.)')
	password = input()
	if password == 'swordfish':
		break
print('Access granted.')
```

for Loop Statement

- Provides the ability to execute a block of code
  - Only a certain number of times
  - Key difference to while loop
- Characteristics
  - The `for` keyword
  - A variable name
  - The `in` keyword
  - A call to the `range()` function with up to three integer values passed to it
  - A colon
  - Starting on the next line, an indented block of code (called the for clause)

for.py

```
# example for loop statement

print('My name is')
for i in range(5):
	print('Vincent Five Times (' + str(i) + ')')
```

for and while Loops

- One can use `while` loop instead of `for` loop
- Difference is that `for` loop is more concise
- Example
  - forEquivalent.py

```
# example for loop statement using while

print('My name is')
i = 0
while i < 5:
        print('Vincent Five Times (' + str(i) + ')')
        i = i + 1
```

Starting, Stopping and Stepping Arguments to range()

- The range() function can accept up to three comma separated values
  - First value represents the starting point
  - Second value represents the stopping point
  - Third value represents the stepping
- Example:

```
for i in range(0, 6, 2):
    Print(i)
```

- Output:

```
0
2
4
```

Importing Module

- Python can call a set of built-in functions
  - print(), input(), str(), etc.
  - Default knowledge
- There are standard library modules
  - Contains a set of related functions
  - Can be embedded in your programs
  - Extends basic Python
  - Linked knowledge
- Example
  - Math module: has mathematical related functions
  - Random module: has random number related functions

Import Statement

- Before the functions in a module can be used
  - Module must be imported using an `import` statement
- Characteristics
  - The `import` keyword
  - The name of the module
  - Optionally, more module names separated by commas
- Example:
  - import.py

import.py

```
# import statement example

import random
for i in range(5):
        print(random.randint(1, 10))
```

Ending Program Early

- Usually program terminates when
  - Execution reaches the bottom of instruction
  - Error situation
- Programs can be stopped at any stage by
  - Calling the `sys.exit()` function
  - A function contained in `sys` module
- Example
  - exit.py

exit.py

```
# exit function example
import sys
while True:
        print('Type exit to exit.')
        response = input()
        if response == 'exit':
                sys.exit()
        print('You typed '+ response + '.')
```

Ollscoil Teicneolaíochta na Mumhan
Munster Technological University