Multi-Paradigm Programming: Imperative/Procedural Programming

Dominic Carr

Atlantic Technological University

dominic.carr@atu.ie

What We Will Cover

1. Goals of this Session

2. Imperative Programming
   - Procedural Programming
     - Python Example
     - Procedural Programming Example
     - Imperative Programming Example

Goals of this Session

Goals

- To understand....
  - What is Imperative programming?
  - What is Procedural programming?
  - How a Procedural program is structured

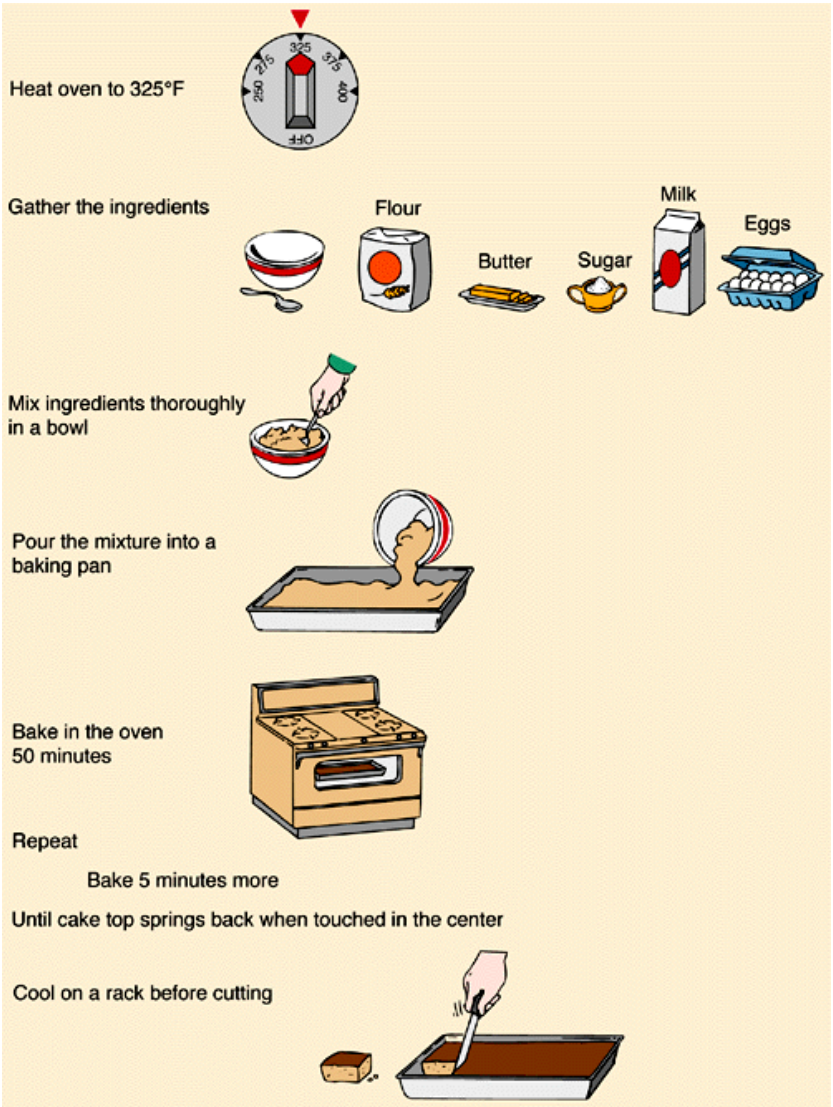
Imperative Programming

Imperative I

Imperative Programming

- Programming with an explicit sequence of commands that update state
- Says **how** to do something.
- Consider baking a cake
  - An imperative program says how to do something in the correct sequence it should be done
    - therefore order of execution (the order in which each statement is executed) is important.
    - Obviously you cannot add candles if you didn't bake the cake right?
    - You cannot eat no cake!

Imperative II



Imperative III

Very General Structure

First **do this** and next **do that**, then **repeat this 10 times**, then **finish**.

Imperative IV

Procedural Programming

- Is simply Imperative programming with procedure calls
  - AKA methods or functions or sub-routines
- To **avoid repetition** and to **provide structure** early programmers realized they should *group instructions together* into re-usable elements called procedures which can then be called when needed during program execution
- Imperative did provide a means of non-linear execution a "go-to" but it was messy.
  - This was one of the first advances in *programming maintainability*.
  - The idea was first implemented in ALGOL in the 1950's
- Mostly when someone say they are doing Imperative Programming they are really doing Procedural Programming

Imperative V

Listing 1: Very Simple Function in Python

```
def my_function():
  print("Hello from a function")

my_function()
```

Imperative VI

Listing 2: Slightly Less Simple Function in Python

```
def print_list(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

print_list(fruits)
```

Imperative VII

Listing 3: C Language Example (Procedural)

```
 # Print numbers from 1 to 10
#include <stdio.h>
int main() {
  int i;
  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
  return 0;
}
```

- Execution begins at **main()** the programs entry point
- Proceeds linearly
- Loop repeats until condition is false
- **printf()** is a function taking args, defined as part of standard C
- Program terminates returning exit code 0

Imperative VIII

```
#include <stdio.h>
int sum(int a, int b)
{
        if (a==b)
        {
                return (a+b)*3;
        } else {
                return (a+b);
        }
}
int main()
{
        int res = sum(1,2);
        printf("Result is %d\n", res);
        res = sum(3,3);
        printf("Result is %d\n", res);
}
```

Imperative IX

```
def sum(a, b):
        if (a==b):
                return (a+b)*3
        else:
                return (a+b)

res = sum(1,2)
print(res)
res = sum(3,3)
print(res)
```

Imperative X

Listing 4: Imperative with go-to

```
    result = []
    i = 0
start:
    numPeople = length(people)
    if i >= numPeople goto finished
    p = people[i]
    nameLength = length(p.name)
    if nameLength <= 5 goto nextOne
    upperName = toUpper(p.name)
    addToList(result, upperName)
nextOne:
    i = i + 1
    goto start
finished:
    return sort(result)
```

Imperative XI

- Imperative Programs often compile to binary executables that run more efficiently since all CPU instructions are themselves imperative statements
- Imperative approaches and “lower level” languages are often used where efficiency and code-footprint are important such as in embedded systems.
  - “Low Level” refers to the closeness to the machine with regard to the level of abstraction
  - An approach like declarative programming would be more “high level”

Imperative XII

- Procedural Languages
  - C
  - Python
  - ALGOL
  - JavaScript
  - PHP

Guidelines for creating Methods/Functions/Procedures I

- DRY (Don't repeat Yourself)
  - Group similar logic in a single method
  - Do not duplicate them all over your code
    - maintenance nightmare
- SRP (Single Responsibility Principle)
  - Your method should do one "thing"
  - Be responsible for one aspect of the program
- Smaller is better
  - Readable
  - Maintainable
- When a method is long...
  - Likely complex too!
  - break it down into smaller functions
- Good method names act as "self documenting"
  - Every language has ways of explicitly documenting when needed

Guidelines for creating Methods/Functions/Procedures II

The End