session2_summary.md

Programming Training

Program	Programming Concepts - Python
Session	Sunday, 1 July 2018, 14:00 - 18:00

Branching

Branching is a conditional statement, basically we impose some logical tests and execute some logic depends on whether the logical tests are satisfied.

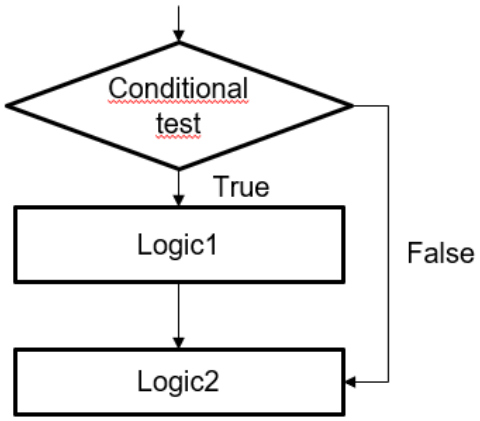
if

The most basic form of branching is a single `if`. We do some logical test, and if it passes, we perform some logic, else, do nothing.

In pseudocode, we have:

```
if <conditional_test_is_true>:
    logic1
logic2
```

In flowchart, we have:

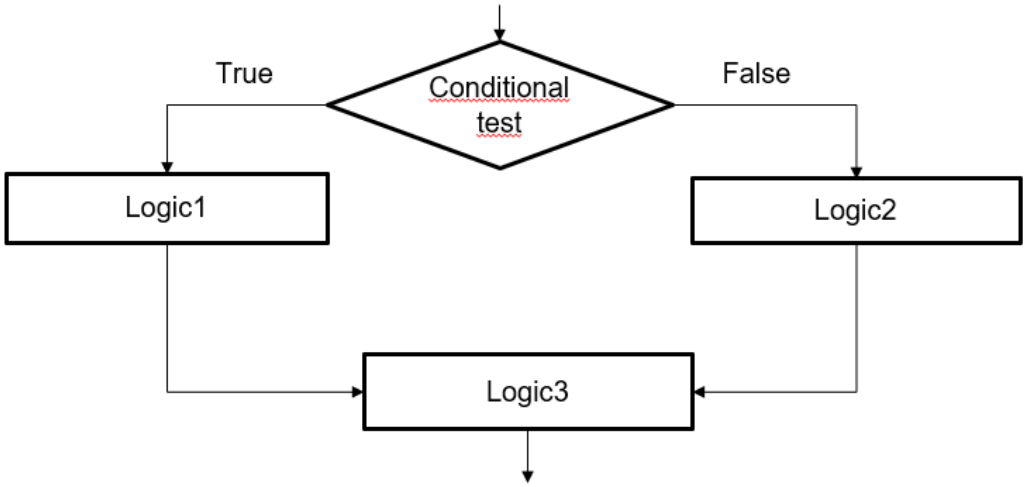


if - else

The next form is `if-else`. The only difference here is we still do some other logic if the conditional check doesn't pass.

```
If <conditional_check_is_true>:
    logic1
else:
    logic2
logic3
```

Also can be depicted with:

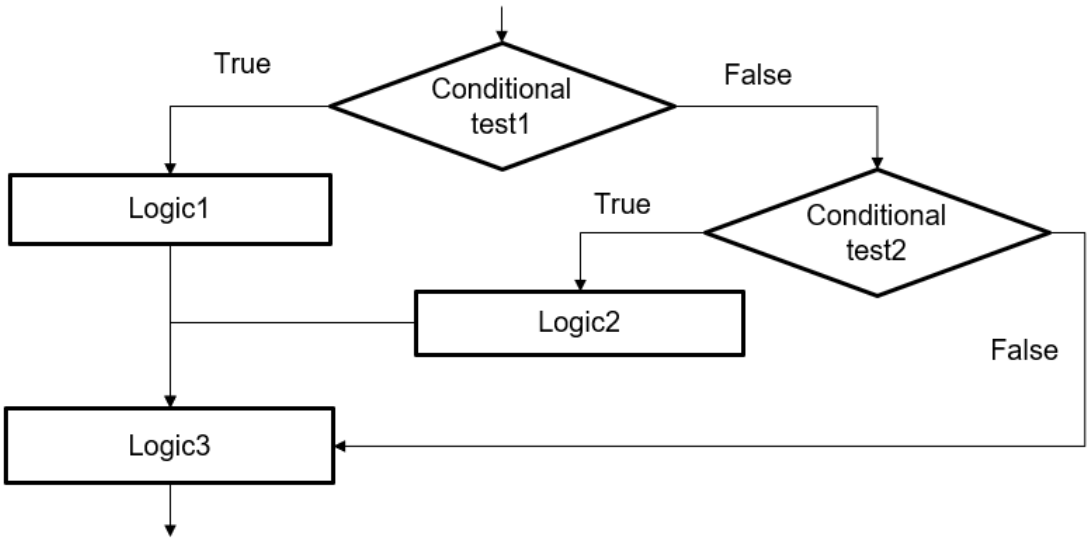


If - else if

This basically means we have more than one conditional check. If the first check fails, we check for the second condition.

```
If <condition1_is_true>:
    logic1
else if<condition2_is_true>:
    logic2
logic3
```

In flowchart:



We can extend the conditional check as many levels as we want, and we also can combined it with `else` clause at the end of all conditional checks.

```
If <condition1_is_true>:
    Do some logic here
else if <condition2_is_true>:
    Do other logic here
else if <condition3_is_true>:
    Another logic here
else:
    Execute this block when all checks fail
```

Having another if clause inside an if clause is also possible, for example, we can have some gigantic `if` block as follow:

```
If <condition1>:
    Do logic1
    if <condition2>:
```

```
        Do logic2a
    else:
        Do logic2b
    if <condition3>:
        Do logic3
else if <condition4>:
    if <logic5>:
        Do logic5
    if <logic6>:
        if <logic7>:
            Do logic7
        Do logic6
```

At the most basic level, `branching` might seem very simple, but remember it is one of the building blocks in logic and programming, having full understanding of it is very important. It is simple, yet powerful. When you have a very complex logic to code, the if block might become complex very quickly, as shown in previous pseudocode. Having a pseudocode before jumping into the actual coding might come in handy. Spending some time at the beginning to get the clarity of what you want to achieve will save you time when you actually do the coding. Remember, coding is just the container, computational thinking is the bread and butter of programmer.

For Python syntax, please refer to:

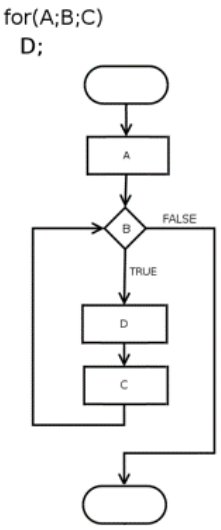
Python Documentation

Loop

Loop or repetition, as the name suggest, is basically repeating some process multiple times. In general, there are 4 components of a loop:

1. Some value(s) or variable(s) initializations
2. Conditional check or termination condition
3. Value(s) update
4. Loop Body

We can capture the general flow as follow:



A is value(s) initialization where we can initialized all variables that we need during the loop, it can be loop counter, variables that we want to update, etc.

B is conditional check or termination condition. At the beginning of every loop, this conditional check will be performed, and the loop will be terminated when the check evaluates as False.

If the conditional check passes, we will execute the loop body (block D in flow chart). Here is where we put the logical statements that we want to perform.

At the end of every loop, usually there will be some values getting updated. Most of the time it is loop counter, but we can update any other values too, depending on what we want to achieve. The rule of thumb is to make sure at some point, the conditional check will fail and we exit the rule. One common pitfall is forgetting to update the loop counter or putting wrong update logic, hence resulting in `infinite loop`.

Similar with `branching`, `loop` is also one of the basic building blocks that you have to fully understand. Again, having pseudocode might help to avoid logical flaw.

For python syntax, please refer to:

Python Documentation

Data Types

Data type is an important concept in programming that often overlooked by programmers, especially in weakly-typed languages such as Python. Weakly-typed language means the language allows a variable to hold different data types, which means we do not need to specify the data type during variable declaration.

```
a = 5 // a is integer here
a = "Hello, world!" // a is string here
a = [1, 2, 3, "string here", True] // a is a list here
```

The above code snippet is allowed in Python, but is not allowed in strongly-typed languages such as C/C++, Java and Golang. They require a variable to hold only one type of data until that variable is out of its scope.

At a glance, it's true that weakly-typed language gives us more flexibility. But if we are not careful, we might run into nasty bugs because of data types confusion. For now, just bear in mind that a variable in Python might change from one data type into another, and different data types might give you different results when some operations are performed on it. As you get more exposure into programming, you will start appreciating the importance of data types.

Practice

For practices, please refer to Practices.