Lecture 3

CORE ELEMENTS
SELECTION
&
FUNCTIONS

Last Week

- What a program is
- What a statement is
- What an expression is
- How to do repetitions/iterations
- Introduced the notion of Functions

Overview

- Branching with Selection Structure
  - nested if-elif-else
- Functions and Parameters
- Void and Non-Void Function

Today's Problem

- We would like to write a program taking a measurement in meters (resp. Feet, inches) as input and convert it to Feet, inches (resp. meters).
  - We need to find a way for the user to tell us which conversion he/she want to do
  - Depending on user SELECTION we need to do one operation or the other.
  - We haven't seen such a structure yet

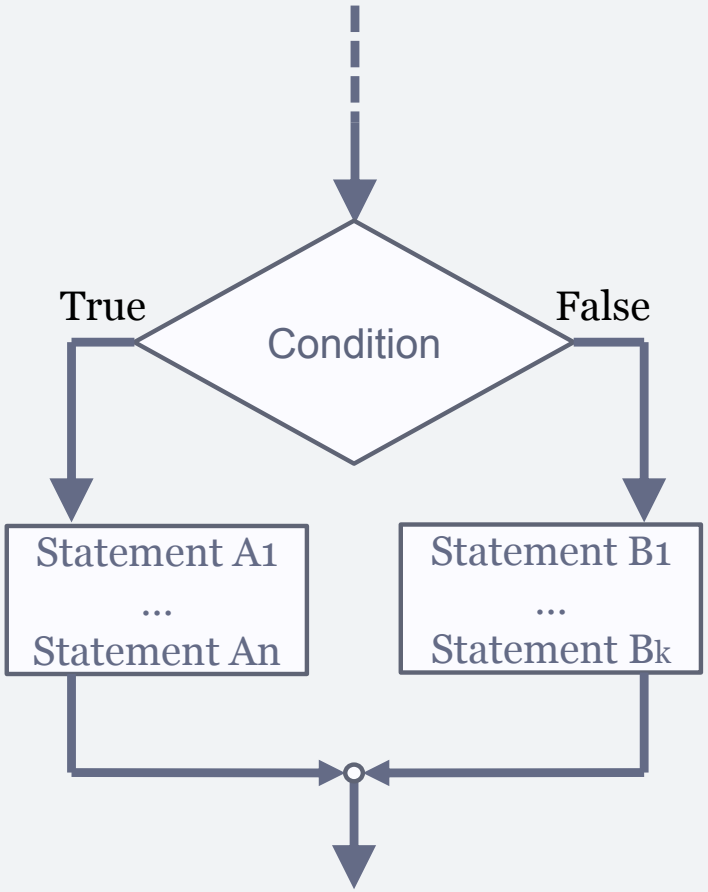
A Selection Structure

IF-ELIF-ELSE

Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

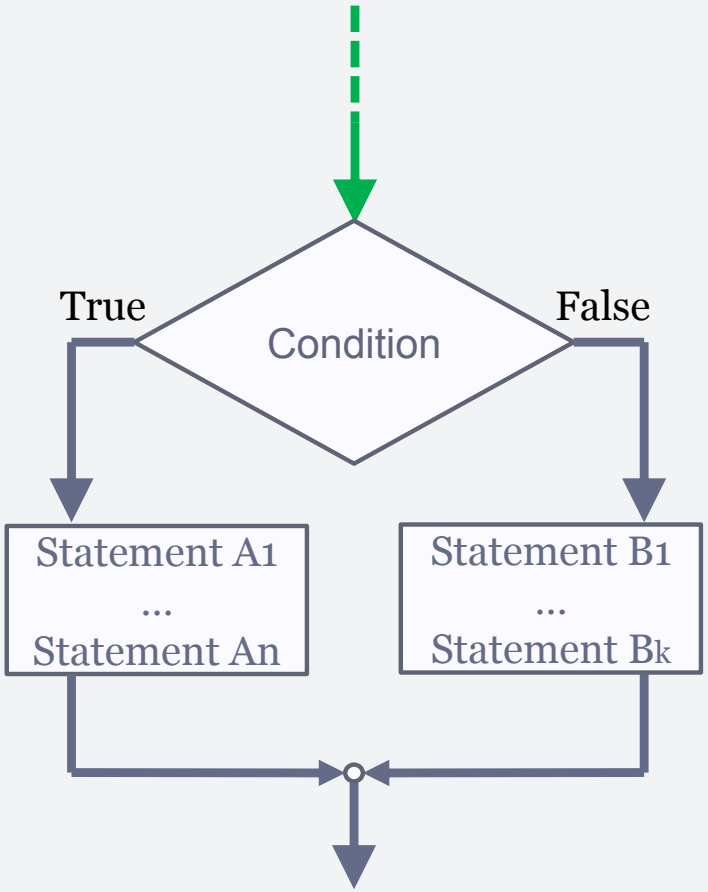
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

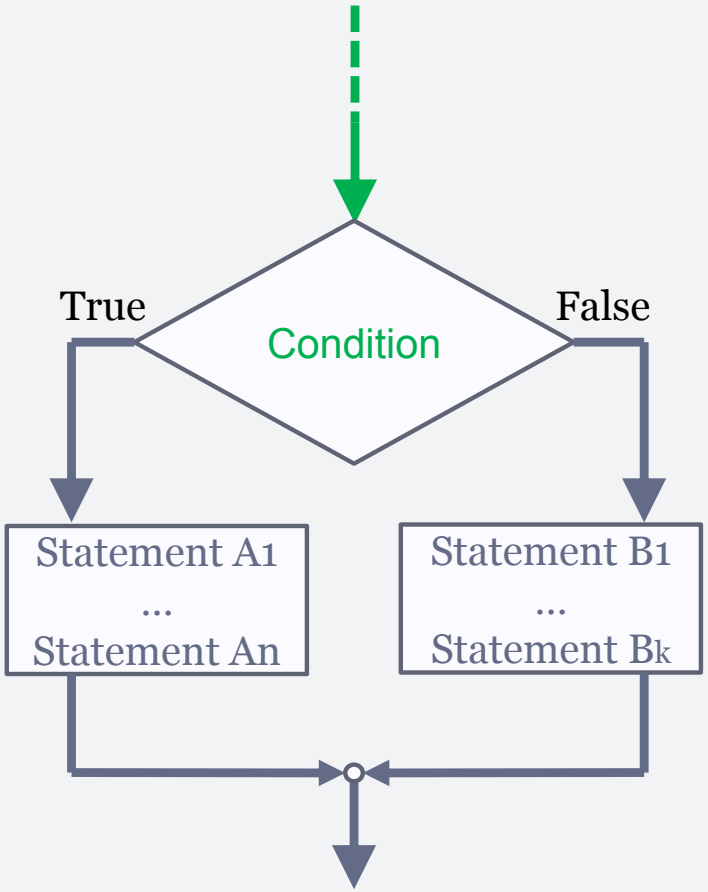
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

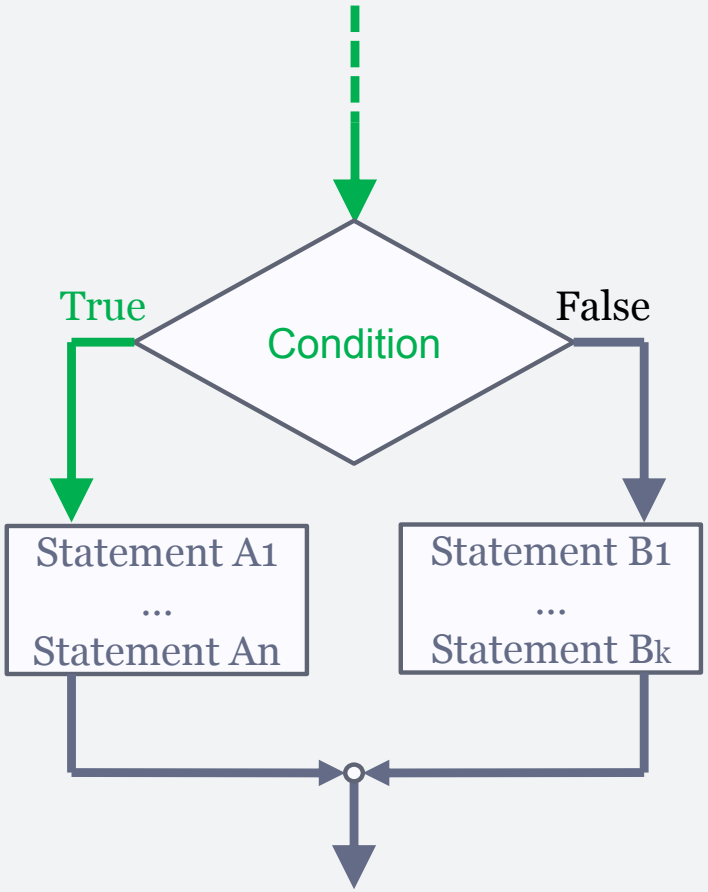
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

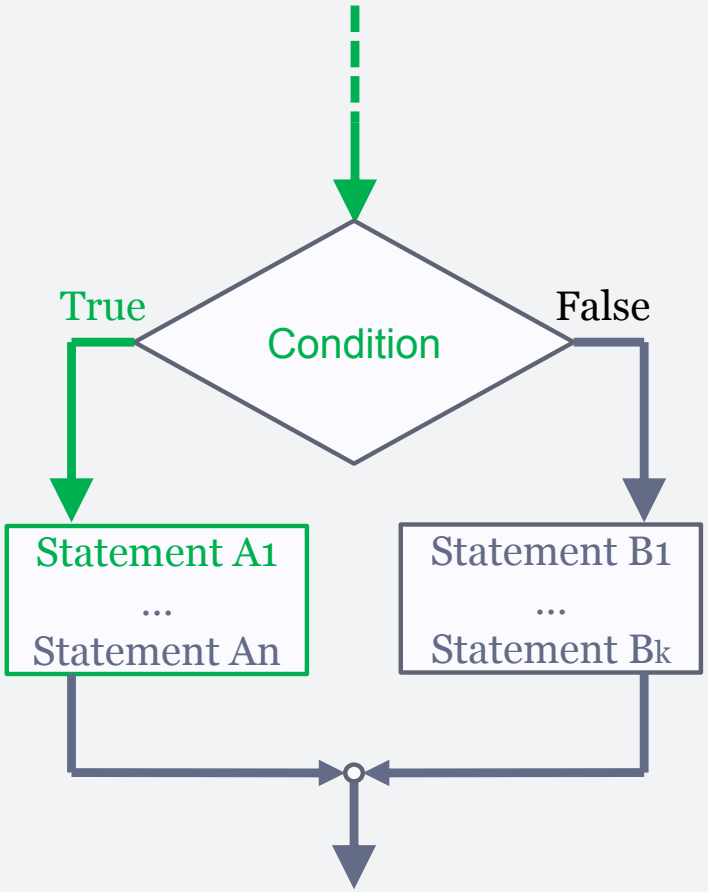
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

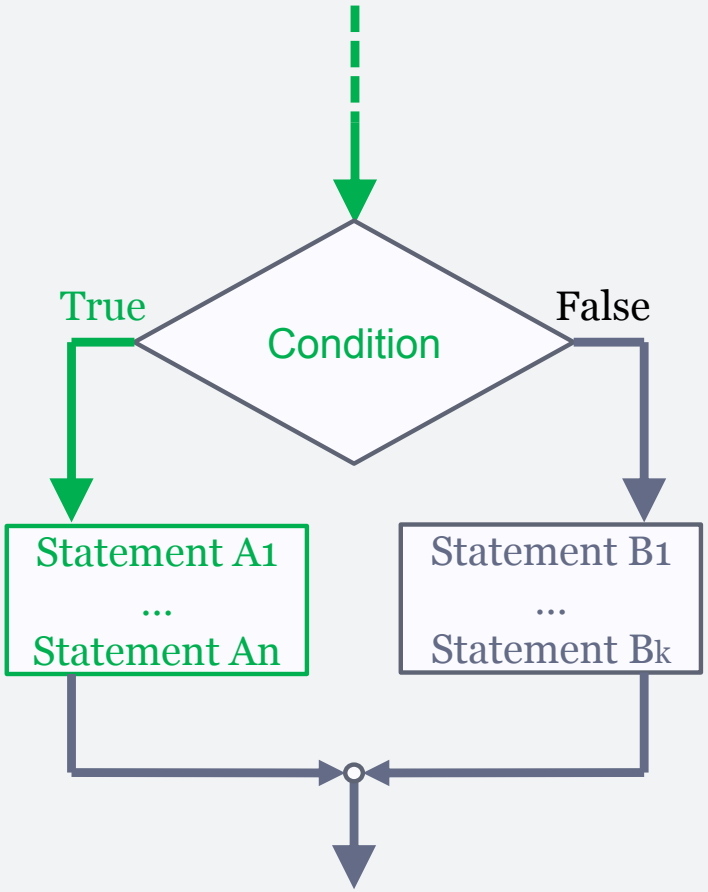
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

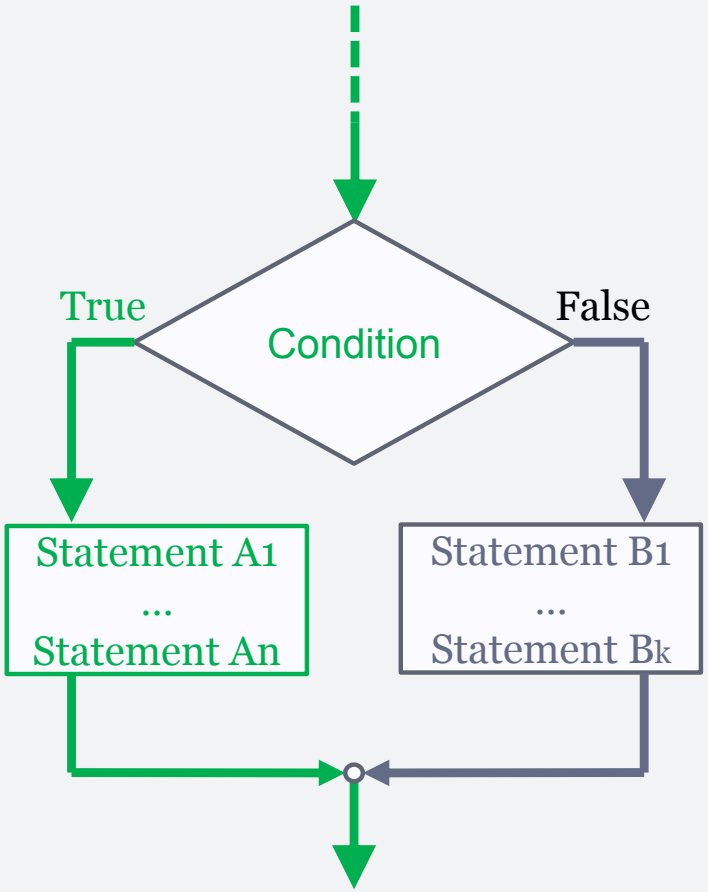
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

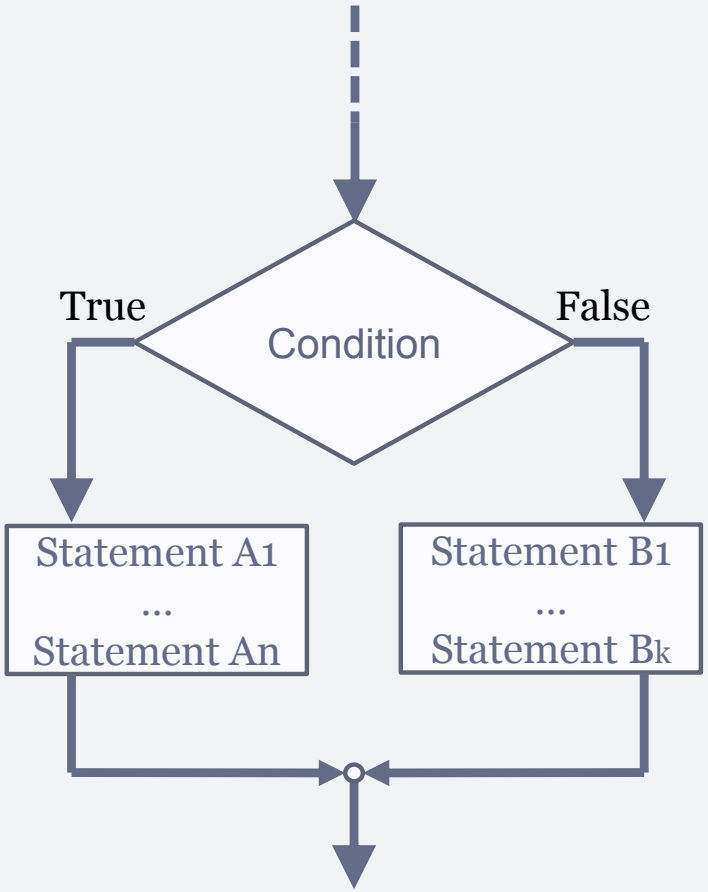
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

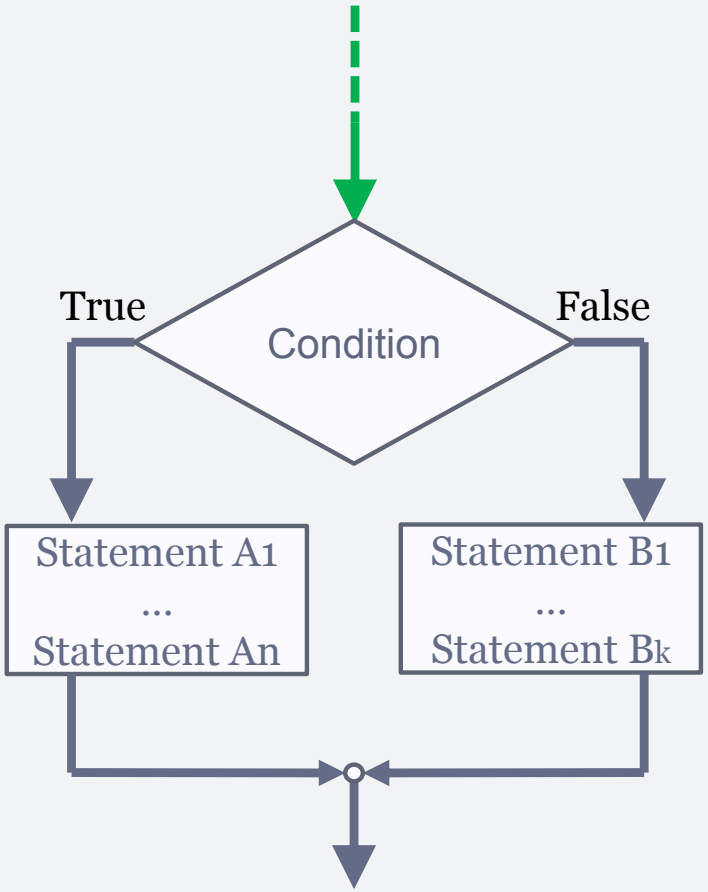
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

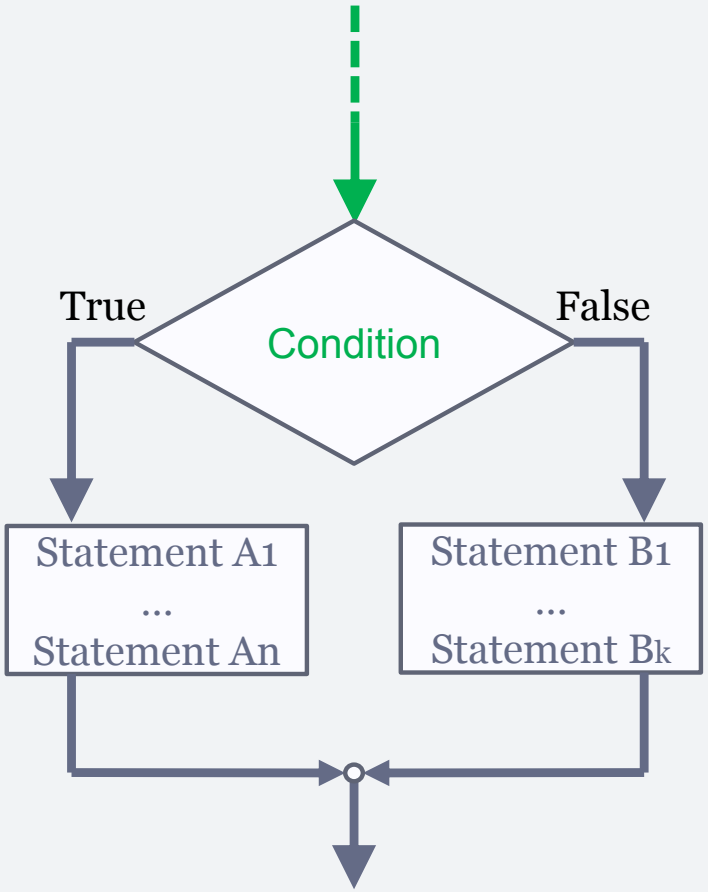
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

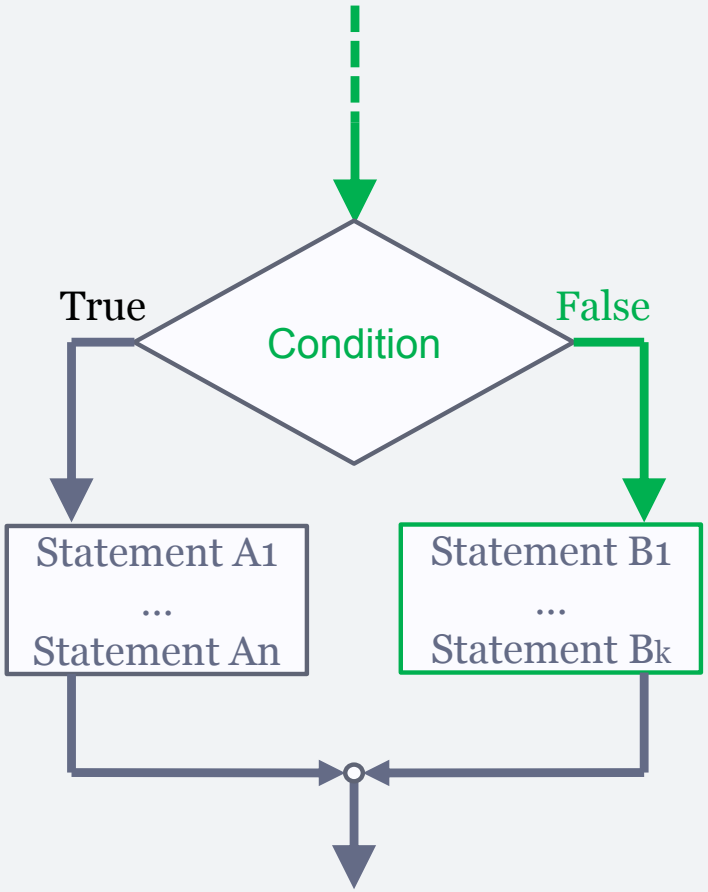
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

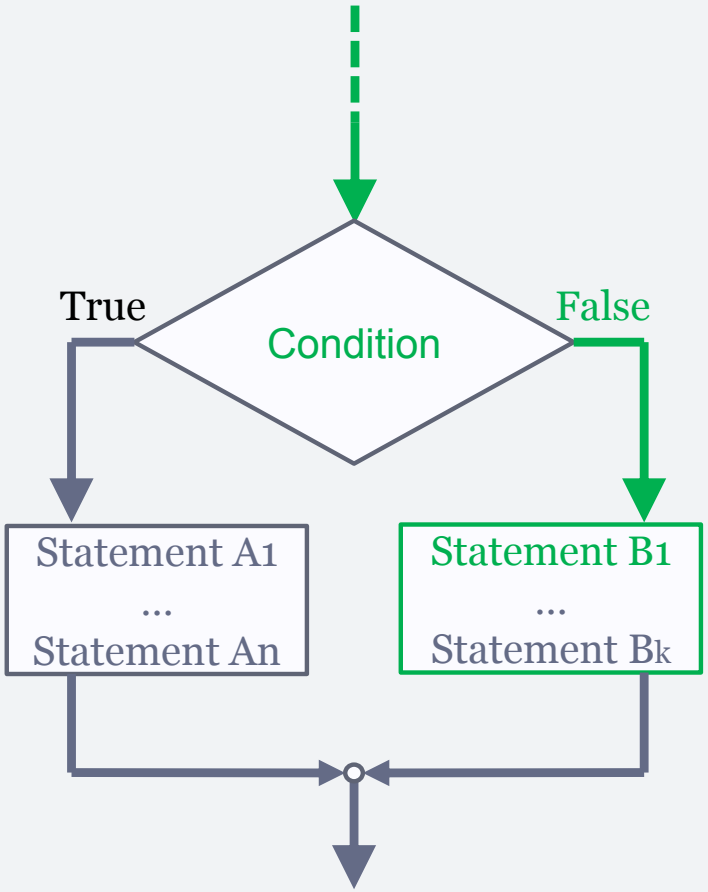
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

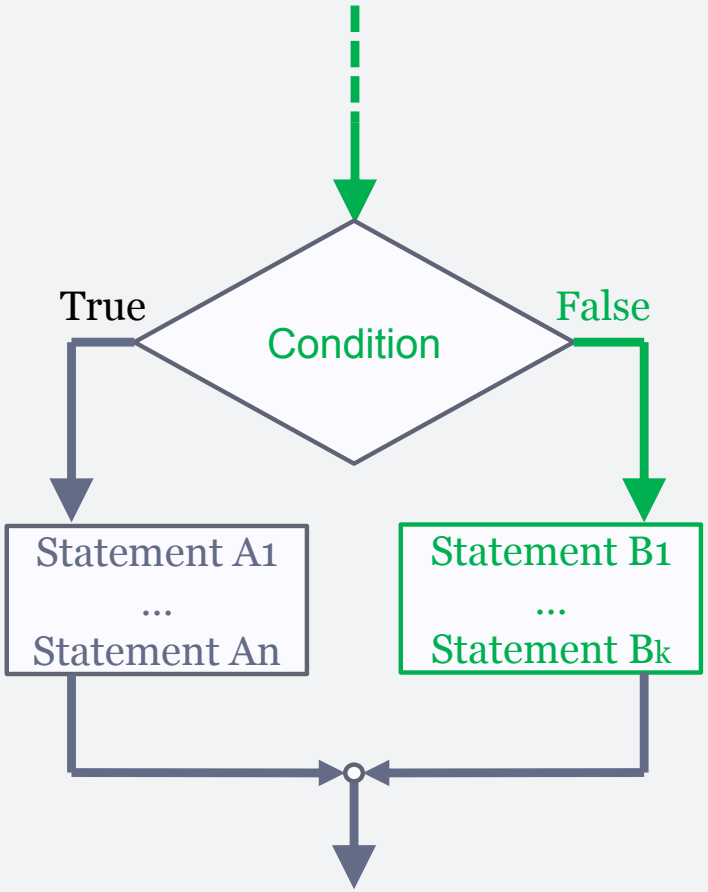
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

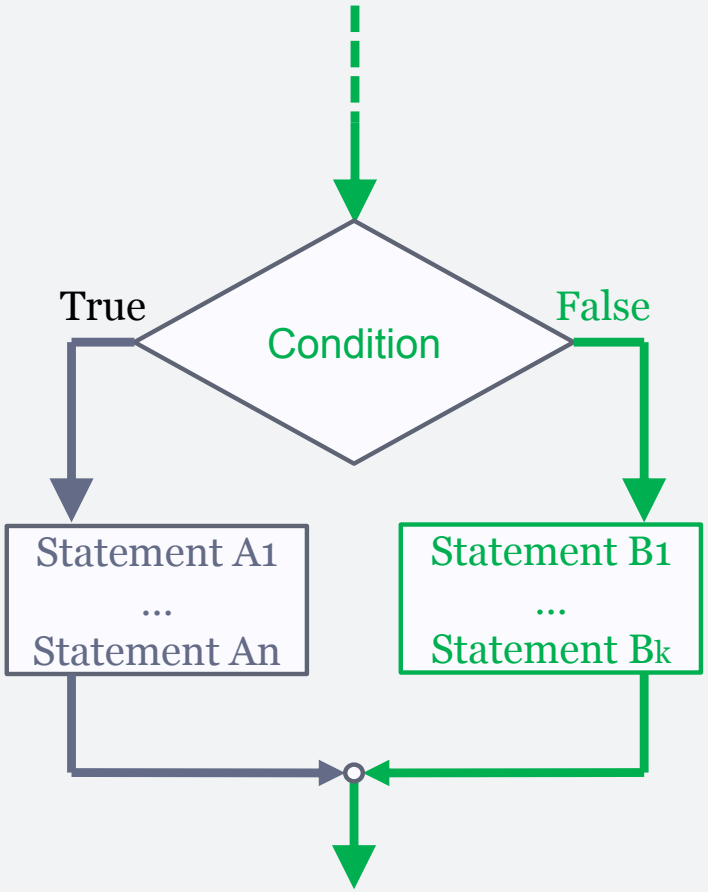
- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Motivation

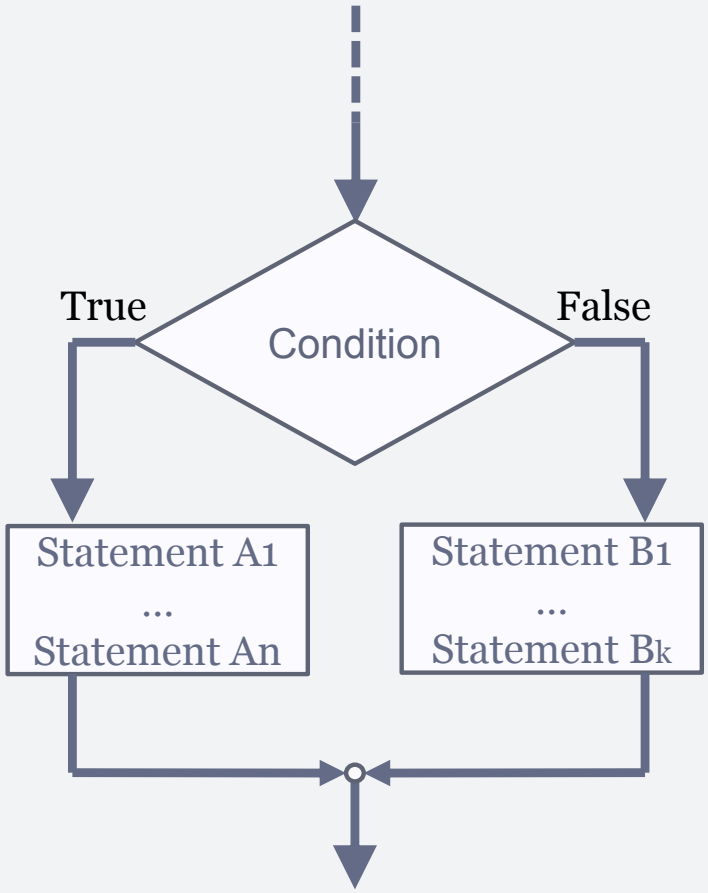
- Straight line programs: sequence of commands executed one after the other one, and only once.
  - Cannot do many interesting things if any.

- Branching: A program can decide which statements to execute based on a condition.
  - If condition is True, execute A_1 to A_n
  - If condition is False, execute B_1 to B_k



Python Code

Schema

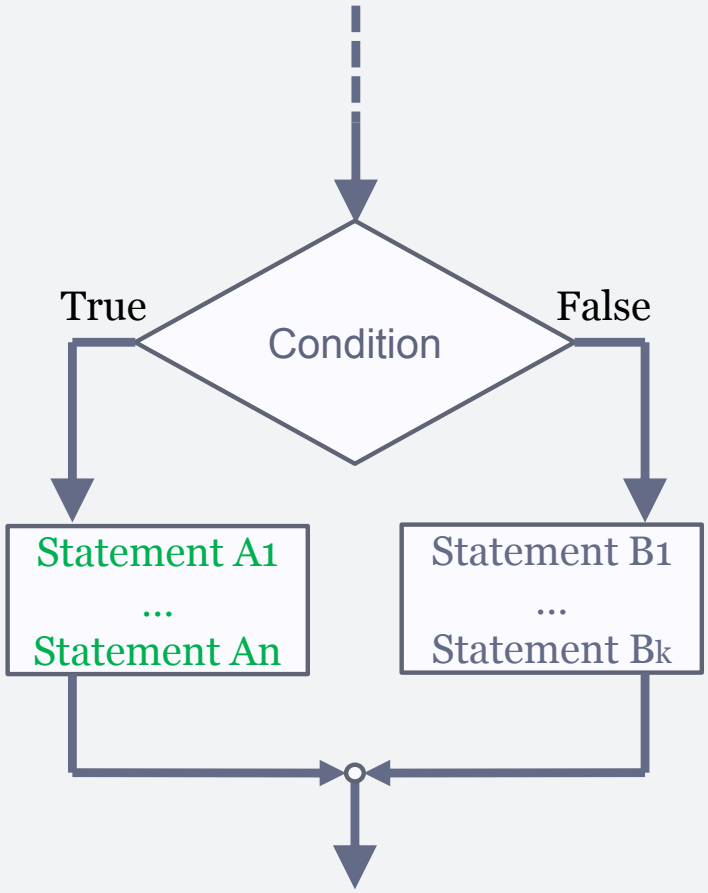


Python Code

```
...
...
if condition :
    Statement A1
    ...
    Statement An
else :
    Statement B1
    ...
    Statement Bk
...
```

Python Code

Schema

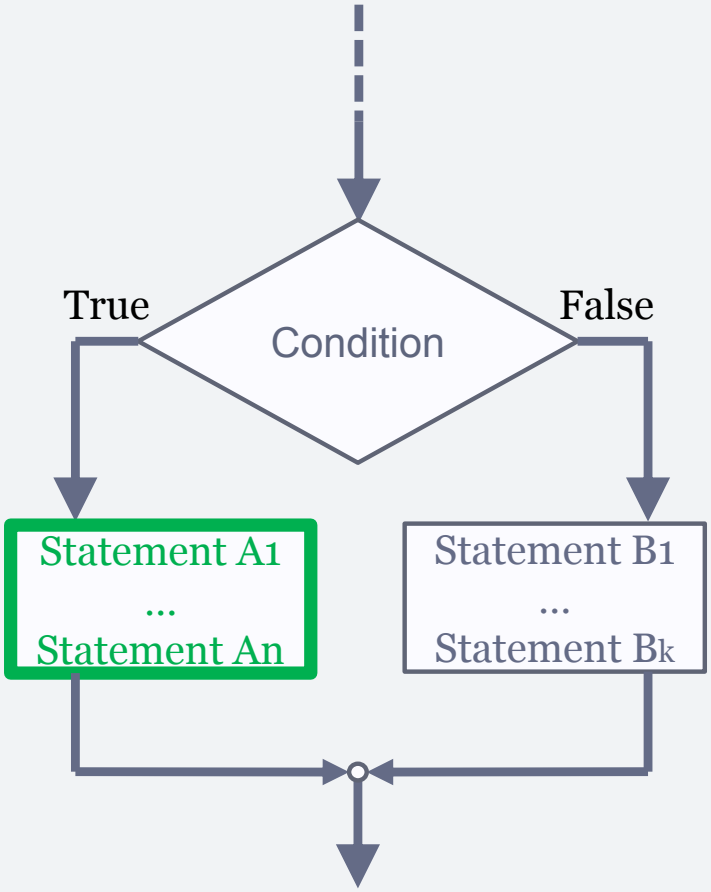


Python Code

```
...
...
if condition :
    Statement A1
    ...
    Statement An
else :
    Statement B1
    ...
    Statement Bk
...
```

Python Code

Schema

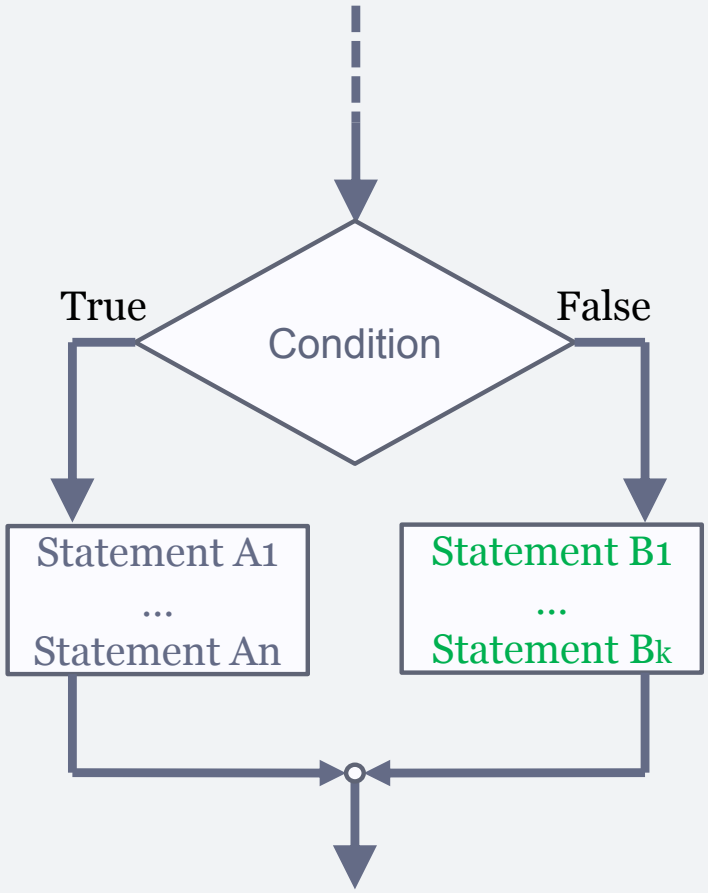


Python Code

```
...
...
if condition :
    Statement A1
    ...
    Statement An
else :
    Statement B1
    ...
    Statement Bk
...
```

Python Code

Schema

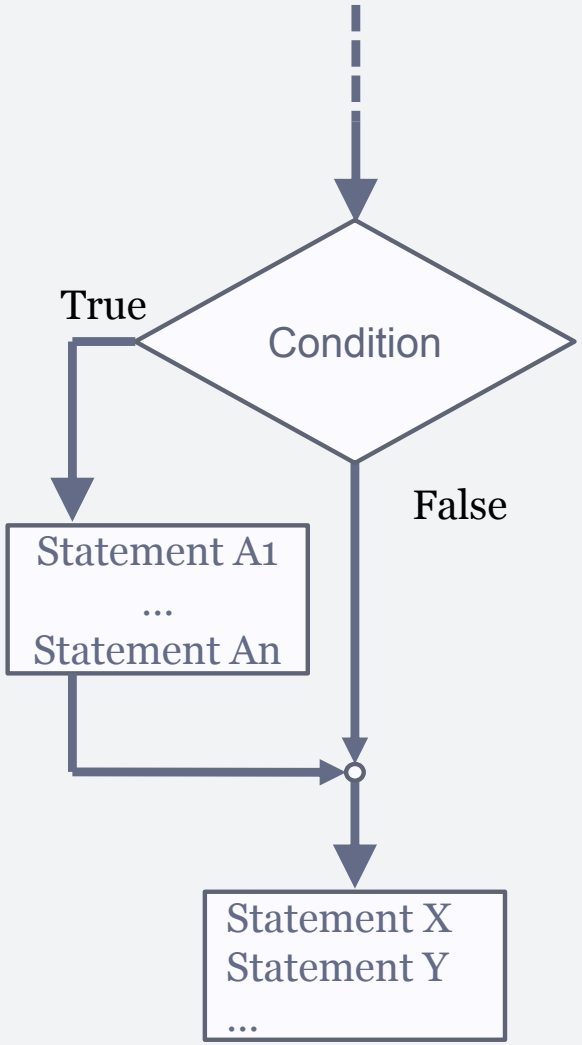


Python Code

```
...
...
if condition :
    Statement A1
    ...
    Statement An
else :
    Statement B1
    ...
    Statement Bk
...
```

Simple if-statement

Schema



Python Code

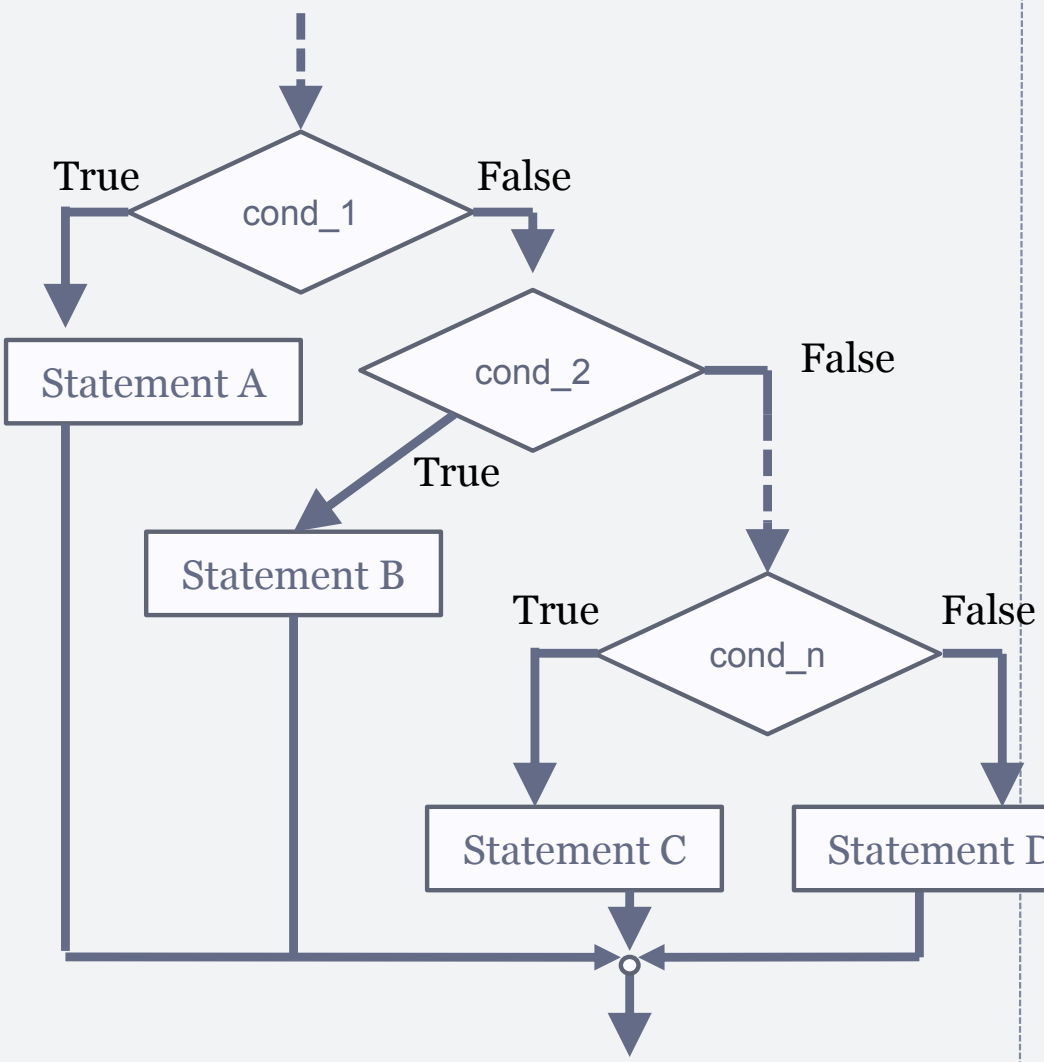
```
...
...
if condition :
    Statement A1
    ...
    Statement An

statement X
statement Y
...
```

Note the change of indentation

if-elif-else statement

Schema

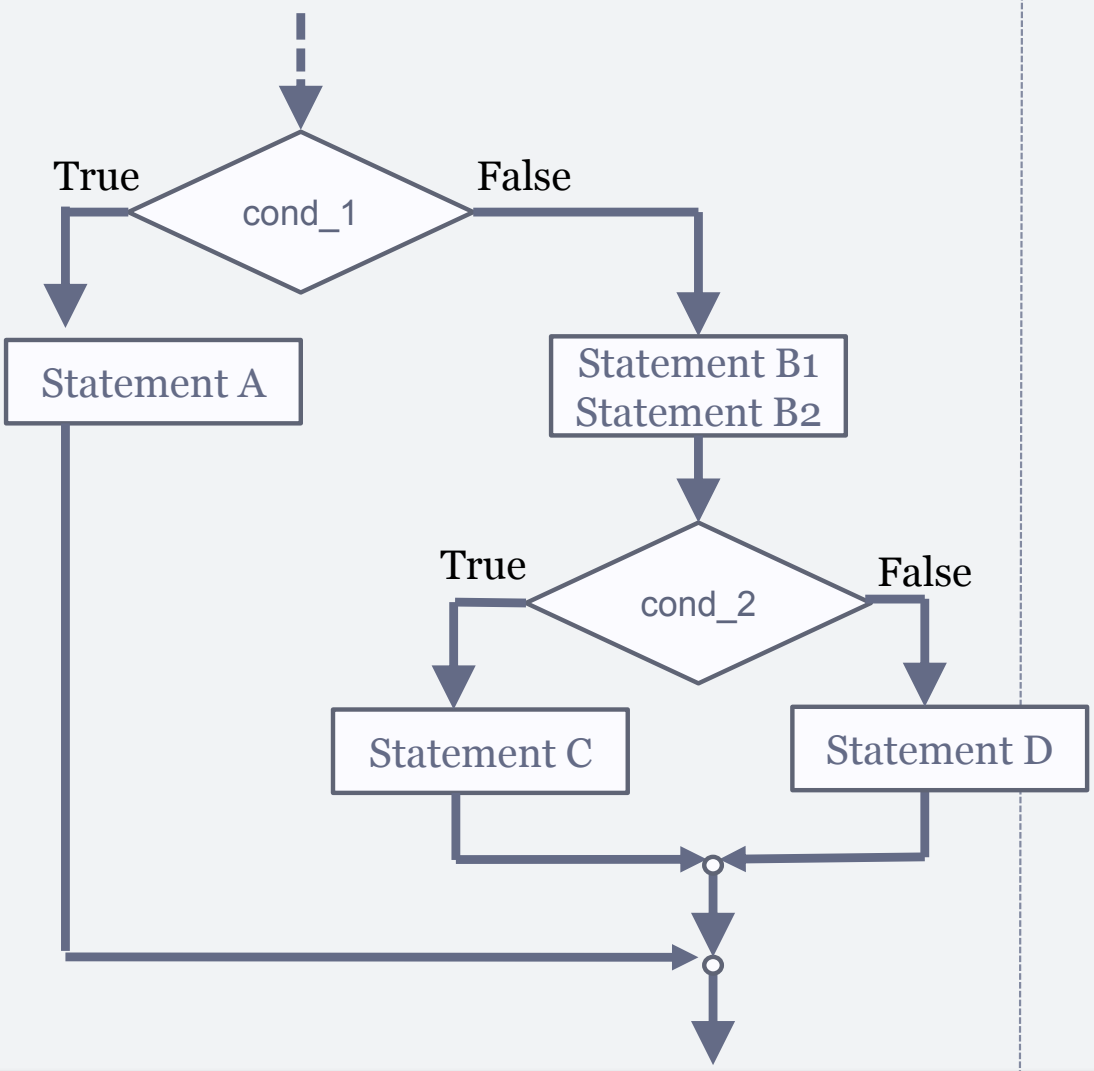


Python Code

```
...
...
if cond_1 :
    Statement A
elif cond_2 :
    Statement B
...
elif cond_n :
    Statement C
else :
    Statement D
...
```

Nested if-else statements

Schema



Python Code

```
...
...
if cond_1 :
    Statement A
else :
    Statement B1
    Statement B2
    if cond_2 :
        Statement C
    else :
        Statement D
...
```

Why using Functions?

- Advantages
  - code reuse
  - facilitate team work
  - modularisation
  - maintainability

- monolithic code
  - huge collection of statement
    - no modularisation
    - no code reuse (cut & paste is not code reuse!)
    - no parallel implementation

Implementation

- See code snippet 1

What is a Function?

- math function
  - f(x) = 2 * x +1
- kind of a sub-program
  - function definition
  - when use, we say that the definition is ‘called’ or ‘invoked’
- may or may not have argument
- may or may not return a value/object

Void and non-void function

- Void function doesn't return anything (meaningful)
  - help(...) function
  - None keyword <type 'NoneType'>, value to represent the "nothingness".

- non-void function returns something other than None
  - input(...) →value
  - raw_input(...) → string
  - len(...) → int

Declaring a Function

- template:

```
def <function_name> (<formal_parameters>):
    <body>
```

  - `<function_name>:` an identifier
  - `<formal_parameters>:` comma-separated identifiers
  - `<body>:` any number of indented statements
  - A non-void function must have a return statement as the last statement of the `<body>`.

Implementation

- See code snippet 2

- Key point 1: Separation of concerns
- Key point 2: A function should do one and only one thing (most of the time)
- Key point 3: Decide which information is needed by the function to do its computation… (parameters)
- Key point 4: Does the function need to return zero, one or more values.

non-void functions

- Must have a return statement
- Function with no parameter
  - (see head_tail())
- Function with one or more parameters
  - (see feet_to_meters(feet, inches) in snippet 3)
- Function returning multiple values
  - (see meters_to_feet(number_meters) in snippet 3)

Be Careful

Code

```
def f(x):
    y = 3 * x + 7
    return y
    print x, y
```

- print x, y never executed
- In some language this would generate a compile error

Void function

- Aimed at changing a state or display information
- No return statement
- Technically, all function return something
- A void function will return the **`None`** value even if there is no return statement in the body

Function Call

- When Python comes to a function call, it initiate a four-step process:

1. the calling program suspends execution at the point of call
2. the formal parameter of the function get assigned the value supplied by the actual parameters in the call
3. the body of the function is executed
4. control returns to the point just after where the function was called

Function Call

Code

```
def head_tail():
```

Interpreter

```
>>> heads = 0
>>> tails = 0
>>> for i in range(1, 10000):
        if (head_tail() == 'head'):
                heads += 1
        else :
                tails += 1
>>>
```

Todays' New Keywords

- Selection
  - If,
  - if-else,
  - if-elif-else,
- Functions
  - def,
  - return,
  - None.

Summary

- Sometime we need to execute some statements while ignoring others depending on some condition
  - Use the selection structure: if-elif-else
- When testing we must ensure that all branches are traversed by the test suit (difficult).
- When designing a function, we define what information is needed (parameters) and what should be returned after computation.
- Separation of concerns is key for reuse of a function