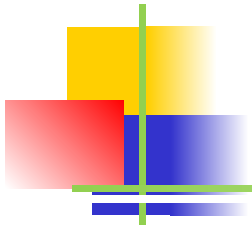
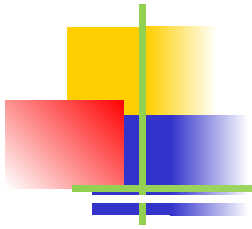


Fundamentos de Programação

António J. R. Neves
João Rodrigues

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro



Summary

- Iteration
- The **`while`** statement
- The **`for`** statement
- The **`range`** function
- The **`break`** statement
- Other stuff: **`continue`** statement and **`else`** clause

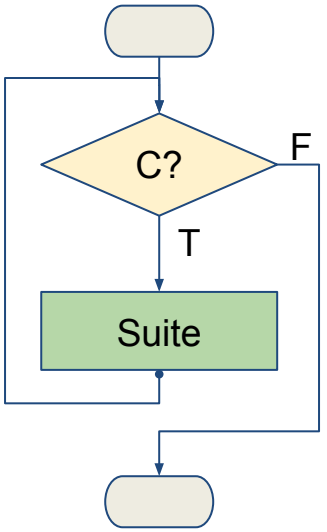
The `while` statement

- The **`while`** statement tells Python to **repeatedly execute** some target statements for as long as a given condition is true.

Syntax	Example

`...`
`while` `condition:`
`    statements`
`...`
	`n = 3`
	`while` `n > 0:`
	`    print(n)`
	`    n = n-1`
	`print("Go!")`

Play



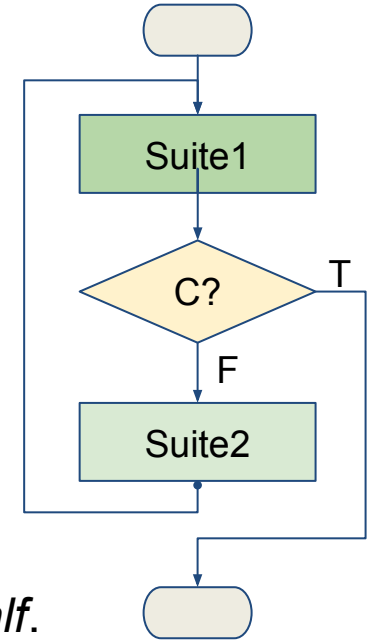
- If the `condition` is true, the `statements` are executed.
- Then, the condition is re-evaluated, and if still true, the statements are repeated.
- When the condition becomes false, execution skips to the line immediately following the block of indented statements.
- The `condition` should be a Boolean expression.
  - Other types of expressions are implicitly converted to `bool`, so any null or empty value means false.

The **`break`** statement

- The *body* of the loop should change the value of one or more variables so that eventually the condition becomes false and the loop terminates. Otherwise, the loop will repeat forever, which is called an *infinite loop*.
- Quite often the best place to decide if the loop should stop is halfway through the body. In that case you can use the **`break`** statement to jump out of the loop.

```
while True:
    line = input('Enter text? ')
    if line == 'done':
        break
    print(line)
print('The end')
```

Play▶️

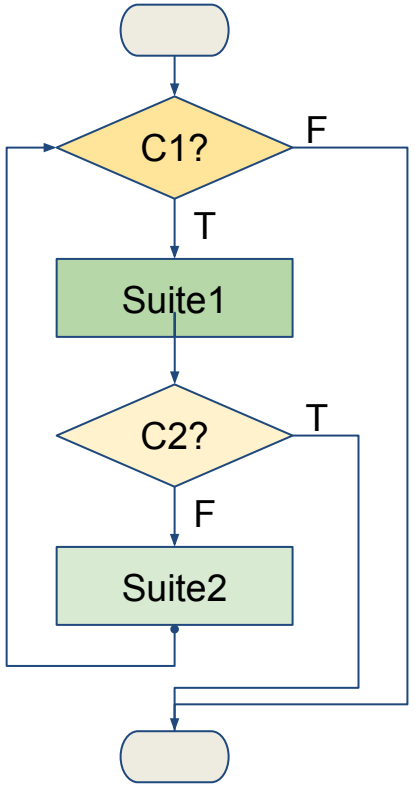


- A loop with this pattern is sometimes called a *loop-and-a-half*.

Multi-exit loops

- Sometimes there are several conditions to terminate the loop and multiple places to test them along the body of the loop.
- Use multiple if-break statements to achieve that.

```
while C1:
    Suite1
    if C2: break
    Suite2
    if C3: break
    Suite3
...
```



The `for` statement

- Another loop mechanism is the **`for`** statement.
- It repeats statements once for each item in a *collection* of items, such as a list, a string or a tuple.

Syntax	Example

`...`
`for` `var` **`in`** `collection:`
`    statements`
`...`
	`for` `n` **`in`** `[3, 1, 9]:`
	`    print(n)`
	`print("End")`

Play

- The `collection` is an expression and it is evaluated first.
- Then, the first item in the collection is assigned to the iterating variable `var`, and the statements block is executed once.
- Next, the second item is assigned to `var`, the statements are executed again, and so on, until the entire collection is exhausted.

The `range` function

- The built-in function `range` generates a sequence of numbers in arithmetic progression.

```
list(range(4))  →  [0, 1, 2, 3]
```

- The `range` function is often used in **`for`** loops.

```
for n in range(0, 4):
    print(n)
```

- It may be called with 1, 2 or 3 arguments, as follows:
  - `range(stop)`
  - `range(start, stop)`
  - `range(start, stop, step)`
- All arguments must be **integers**.
- All arguments can be positive or negative.
- Generates integers up/down to, but **not including**, `stop`.

Loop control statements

- Two special statements --- **`break`** and **`continue`** --- can be used inside a loop body to change the normal flow of execution.
- A **`break`** statement terminates the loop execution and jumps to the statement immediately following the loop.
- The **`continue`** statement skips to the next iteration of the enclosing loop body, without executing the remaining statements in the current iteration.
- Both statements can occur only within **`for`** or **`while`** loops.

The `else` clause

- WARNING: This feature is unusual, confusing, and seldom used. Avoid it. We describe it here for the sake of completeness.
- The iteration statements may have an optional **`else`** clause.

```
count = 0
#count = 1        # uncomment to cause break
while count < 5:
    print(count, "is less than 5")
    if count==3: break
    count += 2
else:
    print(count, "is not less than 5")
print("END")
```

Play ▶️

- Statements in the **`else`** clause are executed once, when the condition evaluates to false.
- They are not executed only if a **`break`** terminates the loop.