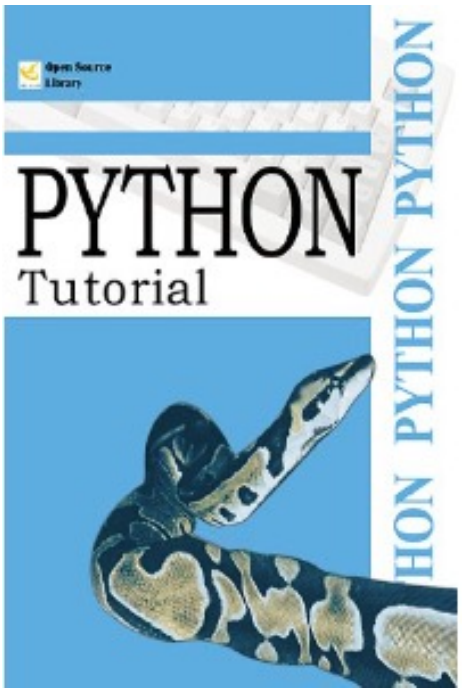
Control Loops



Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Understand Control Loops

Source

- https://docs.python.org/3.10/tutorial/appetite.html
- https://python-textbok.readthedocs.io/en/1.0/index.html
- Python Tutorial - Tapa blanda. GuidoVan Rossum (2012)

Outline

- **Introduction**
- Conditional Statements
- Looping Statements
- Control Flow Statements
- Other Control Statements

Introduction

- Control statements are used to control the flow of execution in a program
- Types:
  - Conditional Statements (if, elif, else, match)
  - Looping Statements (for, while, for...else, while...else)
  - Control Flow Statements (break, continue, pass)
  - Other Control Statements (is, Conditional Operator)

Outline

- Introduction
- **Conditional Statements**
- Looping Statements
- Control Flow Statements
- Other Control Statements

if statement

- Decide some code has to be executed
- The result is a boolean
- Execute code if condition is satisfied

if statement

```
if condition:
    # Some code
```

Sign	Operator	Sign	Operator
==	Equal	and	Logical and
!=	Not equal	or	Logical or
>	Greater	not	Logical not
<	Less than		
>=	Greater or equal		
<=	Less than or equal		

if ... else statement

- Decide some code has to be executed if something is True
- Otherwise, we execute the other part of the code

if statement

```
if condition:
    # Some code
else:
    # Some other code
```

Operators

() (highest)
**
*, /, %
+, -
<, <=, >, >= ==, !=
is, is not
not
and
or (lowest)

Example

```
def check_number(number):
    if number > 0 and number % 2 == 0:
        result = "The number is positive and even."
    else:
        result = "The number is either negative, zero, or positive and odd."
    return result
# Example usage
number = 14
print(check_number(number))
```

Nested *if* statement

- Used when a decision depends on the result of an earlier decision
- Example: calculating the cost of sending a small parcel
  - R5 for the first 300g
  - R2 for every 100g thereafter, up to 1000g
- Important:
  - Maintain proper indentation
  - Indent inner if and else clauses one more level than outer clauses

Example

```
if weight <= 1000:

    if weight <= 300:

        cost = 5

    else:

        cost = 5 + 2 * round((weight - 300) / 100)

    print("Your parcel will cost R%d." % cost)

else:

    print("Maximum weight for small parcel exceeded.")

    print("Use large parcel service instead.")
```

elif and *if* ladders

- elif clause
  - else allows specifying actions when the condition is false
  - elif allows handling multiple alternatives
  - Example: Assigning grades based on marks
- if ... else ladder:
  - Each alternative is nested
  - Increase indentation
- if ... elif ... else ladder:
  - Alternatives are at the same indentation level
  - Easier to read and maintain

If elif else statement

```
if condition:

   # Some code

elif condition1:
    # Some code

else:
    # Some other code
```

Example

If ladder

```
if mark >= 80:

    grade = A

else:

    if mark >= 65:

        grade = B

    else:

        if mark >= 50:

            grade = C

        else:

            grade = D
```

elif

```
if mark >= 80:

    grade = A

elif mark >= 65:

    grade = B

elif mark >= 50:

    grade = C

else:

    grade = D
```

match statement

- Used for pattern matching, similar to switch-case statements in other languages
- Introduced in: Python 3.10
- Patterns can include literals, variable names, wildcards, and more complex structures
- Each case block is executed if the pattern matches the value of the variable
- The _ wildcard is used to catch all unmatched cases, similar to the default case in other languages

Match statement

```
match variable:
    case pattern1:
        # code to execute
    case pattern2:
        # code to execute
    case _:
        # default case
```

Example

if match clause

```
command = "start"

match command:

    case "start":

        print("Starting...")

    case "stop":

        print("Stopping...")

    case _:

        print("Unknown command")
```

Example (II)

if match clause

```
match punto:

   case (0, 0):

      print("Origen")

   case (x, 0):

      print(f"Axis X in {x}")

   case (0, y):

      print(f"Axis Y in {y}")

   case (x, y):

      print(f"Point in ({x}, {y})")
```

Outline

- Introduction
- Conditional Statements
- **Looping Statements**
- Control Flow Statements
- Other Control Statements

for statement

- The loop iterates over elements in an iterable (like a list, tuple, string, etc.)
- For each iteration, the variable is assigned the next value from the iterable
- The indented block of code is executed once for each item in the iterable

`for` statement

```
for variable in iterable:

    # code to execute for each item in iterable
```

Example

for statement

```
lista = [2 , 4, 7]

for i in lista:

    print(i)  # This will print the list
```

for ... else statement

- Used to execute a block of code if the loop completes without encountering a break statement
- The else block is executed only if the *for* loop terminates normally

`for` statement

```
for variable in iterable:

    # code to execute for each item in iterable

     if some_condition:

        break

else:

    # code to execute if the loop completes without a break
```

range

- range() is a built-in Python function that generates a sequence of numbers
- Commonly used in for loops and to create lists of numbers
- Returns an object of type range, useful for large iterations without overloading memory
- Basic Syntax:
  - range(stop)
  - range(start, stop)
  - range(start, stop, step)
- Parameters:
  - start: initial value of the sequence (default is 0)
  - stop: final value of the sequence (not included in the generated sequence)
  - step: increment between each number in the sequence (default is 1)

Example

for statement (with range)

```
lista = [2 , 4, 7]

for i in range(len(lista)): # range(0,len(lista),1)

    print(lista[i])
```

Example

for .. else statement

```
for i in range(5):
    if i == 3:
        break
    print(i)
else:
    print("Loop completed without break")

# Prints 0, 1, 2 and not the else block because loop is
broken at i == 3
```

Range() function

```
range(5) # [0, 1, 2, 3, 4]

range(2, 8) # [2, 3, 4, 5, 6, 7]

range(1, 10, 2) # [1, 3, 5, 7, 9]
```

while statement

- The loop continues to run as long as the condition is true
- The indented block of code is executed repeatedly until the condition becomes false

`while` statement

```
while condition:

    # code to execute
```

Example

while statement

```
count = 0

while count < 5:

    print(count)  # This will print 0 to 4

    count += 1
```

while... else statement

- The else block is executed only if the while loop terminates normally (i.e., not via break)

while statement

```
while condition:

    # code to execute as long as the condition is true

    if some_condition:

        break

else:

    # code to execute if the loop completes without a break
```

Example

while…else statement

```
count = 0

while count < 5:

    if count == 4:

        break

    print(count)

    count += 1

else:

    print("Loop completed without break")

# Print 0, 1, 2, 3 and not the else block because the loop is broken at count == 4
```

Outline

- Introduction
- Conditional Statements
- Looping Statements
- **Control Flow Statements**
- Other Control Statements

break

- Can be used within both *for* and *while* loops
- Commonly used to terminate a loop based on some condition that occurs during iteration
- When used inside nested loops, break only exits the innermost loop

Example

break statement

```
for i in range(10):

    if i == 5:

        break

    print(i)  # This will print 0 to 4
```

break statement (inner loop)

```
for i in range(3):

    for j in range(3):

        if j == 1:

            break

        print(f"i: {i}, j: {j}")

# This will print:

# i: 0, j: 0 # i: 1, j: 0 # i: 2, j: 0
```

continue

- Can be used within both *for* and *while* loops
- Useful for skipping specific conditions without terminating the entire loop
- When used inside nested loops, continue only affects the loop in which it is called

Example

continue statement (inner loop)

```
for i in range(3):
    for j in range(3):
        if j == 1:
            continue
        print(f"i: {i}, j: {j}")
# This will print:
# i: 0, j: 0 # i: 0, j: 2 # i: 1, j: 0 # i: 1, j: 2 # i: 2, j: 0 # i: 2, j: 2
```

pass

- Used as a placeholder for future code. It does nothing when executed
- Can be used in loops, functions, classes, or conditional statements where syntactically some code is required, but you want to leave it empty for now
- Helpful for stubbing out code or for creating minimal structures that will be implemented later

Example

pass statement

```
for i in range(5):
    if i == 3:
        pass  # Do nothing
    print(i)  # print 0, 1, 2, 3, 4
```

pass statement

```
def my_function():
    pass  # Placeholder for future implementation


class MyClass:
    pass  # Placeholder for future implementation
```

Outline

- Introduction
- Conditional Statements
- Looping Statements
- Control Flow Statements
- **Other Control Statements**

is Operator

- Used to test object identity, not equality
- Checks if two references point to the same object in memory
- Commonly used to check if a variable is None
- Useful for ensuring that two variables point to the exact same object, not just equal values

Example

is operator

```
a = [1, 2, 3]

b = a

print(a is b)  # True, because b is
the same object as a

c = a[:]

print(a is c)  # False, because c is
a different object with the same
contents
```

is operator

```
a = [1, 2, 3]

b = [1, 2, 3]

print(a == b)  # True, because a and
b have the same values

print(a is b)  # False, because a and
b are different objects
```

Example

is not operator

```
a = [1, 2, 3]

b = a

c = a[:]   #Creates a copy of all the elements in a

print(a is not c)  # True, because c is a different object

print(a is not b)  # False, because b is the same object as a
```

Conditional operator

- A concise way to perform conditional assignments or evaluations
- Allows embedding an if-else condition in a single line
- Useful for simple conditional assignments
- Enhances code readability when used appropriately

Conditional Operator

```
value_if_true if condition else value_if_false
```

Example (I)

Conditional operator

```
age = 18

status = "Adult" if age >= 18 else "Minor"

print(status)  # This will print "Adult"
```

Nested Conditional operator

```
score = 85

grade = "A" if score >= 90 else ("B" if score >= 80 else "C")

print(grade)  # This will print "B"
```

Example (II)

Conditional operator

```
age = 18

status = "Adult" if age >= 18 else "Minor"

print(status)  # This will print "Adult"
```

Nested Conditional operator (how it would be with if-else?)

```
score = 85

grade = "A" if score >= 90 else ("B" if score >= 80 else "C")

print(grade)  # This will print "B"
```