Python: Control Flow Statements

Introduction to Computer Programming
Bachelor in Data Science

Roberto Guidi

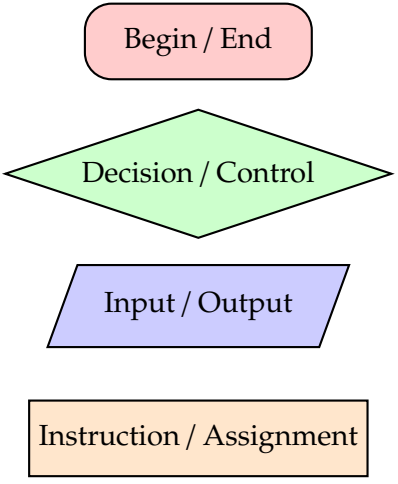
roberto.guidi@supsi.ch

Fall 2021

SUPSI University of Applied Sciences and Arts of Southern Switzerland

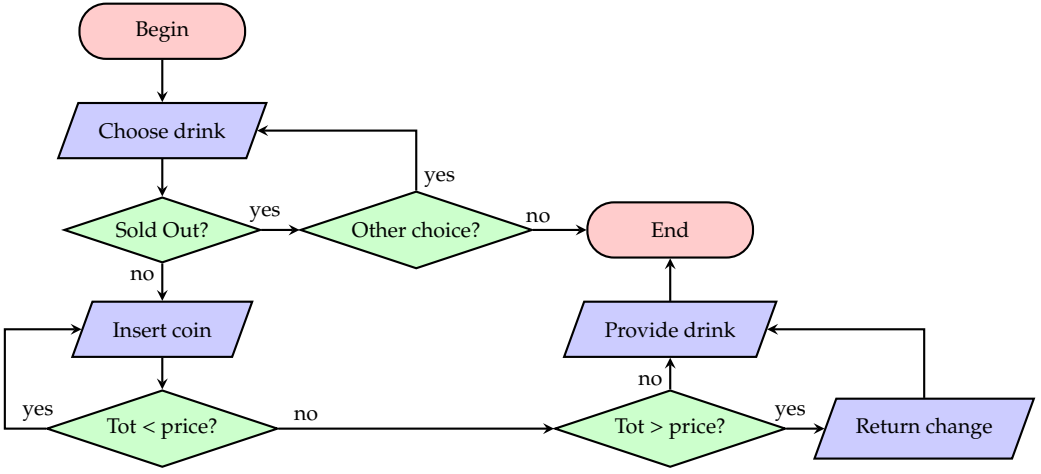
Content created in collaboration with: L. Grossi, F. Landoni

Flow chart diagrams



Refresher

Flow chart diagrams: example



Refresher

Statements

Statements (colloquially "instructions") are the units that are executed in a Python program.

Usually, each statement is a complete line of code that finishes at the end of the line.

In Python there are also ways to make statements span on multiple lines by using:

- the \ character
- implicit line continuation using parentheses (), brackets [] or braces { }

A statement in Python is also defined as logical line

Statements

Statements can be of different kinds, the most significant are:

- Expression statements: used to compute and write a value, or to call a procedure
- Assignment statements: instructions to bind identifiers (names) to values
- Compound statements: they contain other statements and they affect or control the execution of those statements. Flow control statements such as `if`, `while`, `for`, ... are an example.

Statements: example

```
# example of a program with various statement
if __name__ == '__main__':

    value = 0
    for j in range(0, 30):
        if j == 9:
            value += 1
            print("10")
        elif j == 19:
            value += 1
            print("20")
    print("Final value: " + str(value))
```

Program: example

In a program we need to:

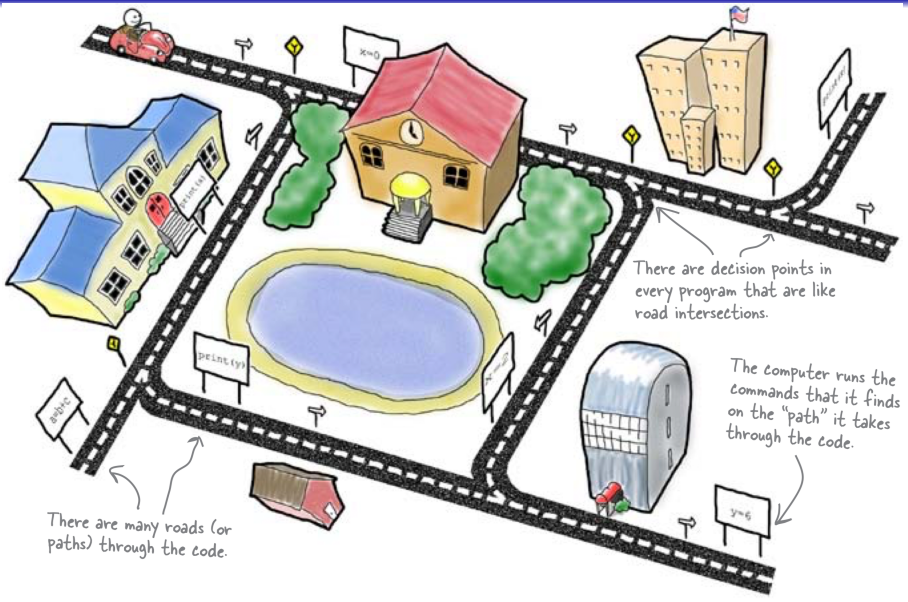
- declare and assign values to the variables that we are going to use,
- write the statements that specify the operations to execute on variables.

```
if __name__ == '__main__':
    name = input("What is your name?: ")
    print("Hi " + name + " welcome to the course")

    num1 = input("Insert an integer number: ")
    num2 = input("Insert another integer number: ")
    num1 = int(num1)
    num2 = int(num2)
    num_sum = num1 + num2
    num_diff = num1 - num2
    num_multi = num1 * num2

    print(str(num1) + " + " + str(num2) + " = " + str(num_sum))
    print(str(num1) + " - " + str(num2) + " = " + str(num_diff)) # comment
    print(str(num1) + " * " + str(num2) + " = " + str(num_multi))
```

Control Flow Statements



— *Head first Programming, O'Reilly*

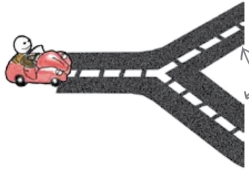
Control Flow Statements

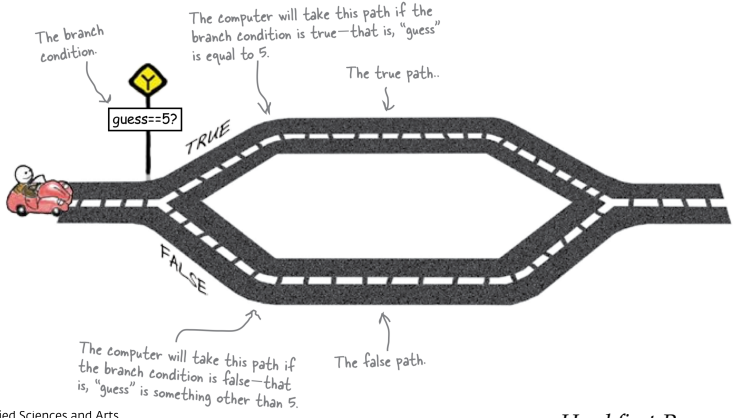
Control Flow Statements are divided in:

- Selection Statements (branches): they are used to direct the flow of execution in different parts of the program.
- Loop Statements (cycles): they are used to represent in a compact form the repetitive execution of particular instruction sequences.

Using these control flow statements it is possible to write quite complex programs.

Selection Statements

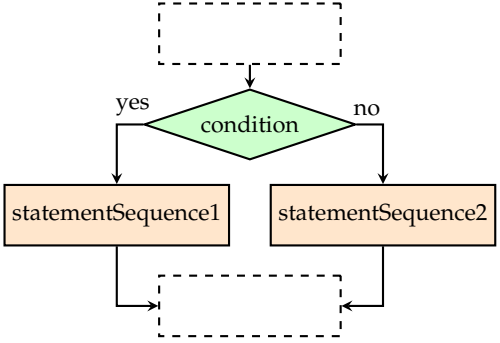




Selection statement in Python

```
if condition1:
  statementSequence1
elif condition2:
  statementSequence2
elif condition3:
  statementSequence3
else:
  statementSequence4
```

The `if . . . else` statement




```
if condition:
  statementSequence1
else:
  statementSequence2
```

The `if . . . else` statement

The `if . . . else` statement allows to execute portions of the code conditionally.

The `if . . . else` statement evaluates expressions that return `boolean` type values.

The `else` is optional.

The `if` selection statement: example without `else`

```
if __name__ == '__main__':

    number = input("Insert a number smaller than 1000: ")
    x = int(number)

    if x > 1000:
        print("Error, the inserted number is greater than 1000")
        x = 1000

    print("The number is: " + str(x))
```

Output 1:

```
Insert a number smaller than 1000:
1002
Error, the inserted number is
greater than 1000
```

Output 2:

```
Insert a number smaller than 1000:
42
The number is: 42
```

The if . . . else selection statement: example with else

```
if __name__ == '__main__':

    print("Insert two numbers: ")
    number1 = input()
    number2 = input()
    a = int(number1)
    b = int(number2)

    if a % b == 0:
        print(str(a) + " is divisible by " + str(b))
    else:
        print(str(a) + " is not divisible by " + str(b))
```

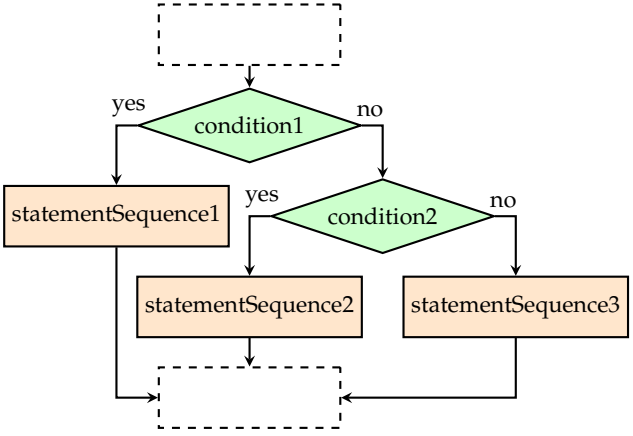
Output 1:

```
Insert two numbers:
12
3
12 is divisible by 3
```

Output 2:

```
Insert two numbers:
13
5
13 is not divisible by 5
```

The if . . . else . . . if . . . else statement




```
if condition1:
  statementSequence1
else:
  if condition2:
    statementSequence2
  else:
    statementSequence3
```

The if . . . else . . . if . . . else statement

```
if condition1:
  statementSequence1
else:
  if condition2:
    statementSequence2
  else:
    statementSequence3
```

```
if condition1:
  statementSequence1
elif condition2:
  statementSequence2
else:
  statementSequence3
```

These two manners to write the conditional statement are exactly equivalent!

The if . . . elif . . . else statement: example

```
temp = int(input("Insert the temperature: "))

if temp < 18:
  print("It's cold!")
elif temp < 25:
  print("It's fine.")
else:
  print("It's hot!")
```

Output 1:

```
Insert the temperature: 22
It's fine.
```

Output 2:

```
Insert the temperature: 30
It's hot!
```

Are these the same thing?

```
x = 10
y = 15
z = 20
if x < y:
  print("x smaller than y")
elif x < z:
  print("x smaller than z")
elif x > z:
  print("x greater than z")
```

Output:

```
x smaller than y
```

```
x = 10
y = 15
z = 20
if x < y:
  print("x smaller than y")
if x < z:
  print("x smaller than z")
if x > z:
  print("x greater than z")
```

Output:

```
x smaller than y
x smaller than z
```

Example: Conditional Statement with and operator

```
if __name__ == '__main__':

  day = int(input("Insert day, month and year: "))
  month = int(input())
  year = int(input())

  # calculate the next day
  # ...

  # manage the last day of the year scenario
  if day == 31 and month == 12:
    day = 1
    month = 1
    year += 1
```

Example: Conditional Statement with or operator

```
if __name__ == '__main__':

  years = int(input("Insert the duration: "))
  value = float(input("Insert the value: "))

  # Always executed
  print("The value of the investment after ", end='')
  if years == 0 or years > 1:
    print(str(years) + " years")
  else:
    # Manage the case with year = 1
    print("1 year", end='')

  # Always executed
  print(" is of " + str(value) + " CHF.")
```

Ternary operator

In Python, to express conditions, it also exists an inline if, also known in other languages as ternary operator.

```
a if condition else b
```

The inline if returns the first value (a) if the condition is true. Otherwise it returns the second value (b).

It allows to write, in a compact manner, expressions like: "if . . . than . . . else"

Ternary Operator

```
next_value = n / 2 if n % 2 == 0 else 3 * n + 1
```

It is equal to:

```
if n % 2 == 0:
  next_value = n / 2
else:
  next_value = 3 * n + 1
```

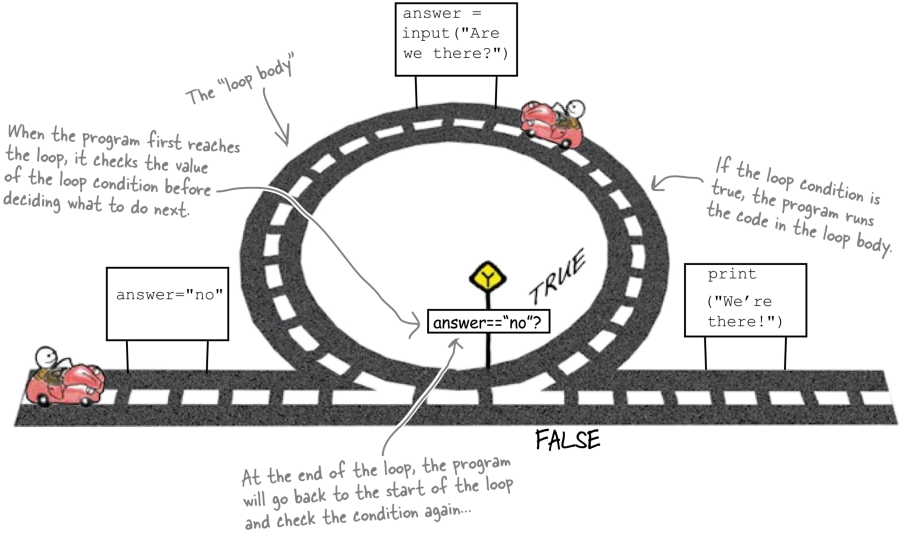
Ternary Operator: example

```
if __name__ == '__main__':

    value = float(input("Insert the value: "))

    # Calculate absolute value
    abs_val = -value if value < 0 else value
    print("The absolute value of" + str(value) + " is " + str(abs_val))
```

The Loop Statements



— *Head first Programming, O'Reilly*

Loop Statements in Python

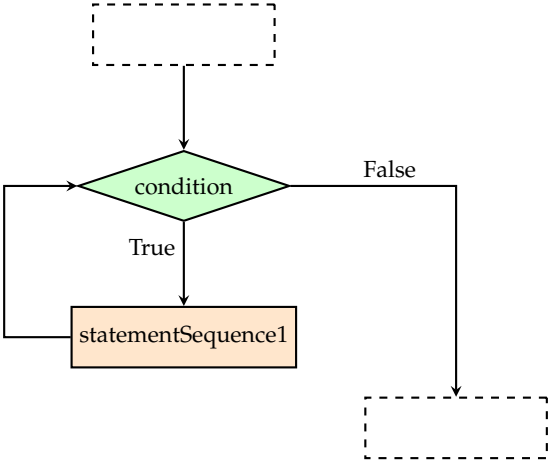
```
while condition:
  statementSequence
```

Executed zero or more times.

```
for i in range(5):
  statementSequence2
```

Executed a fixed number of times.

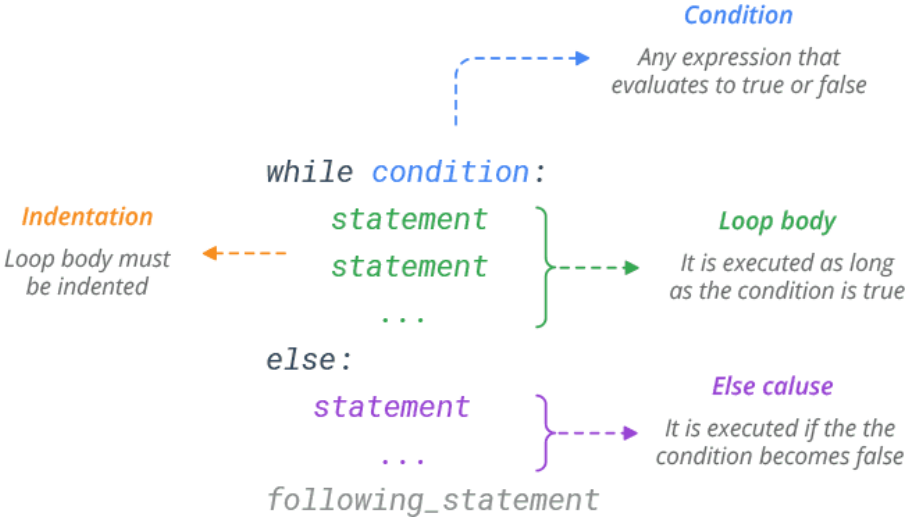
The `while` loop statement



General loop format:

```
while condition:
  statementSequence1
  if test: # optional
    break
  if test: # optional
    continue
else: # optional
  statementSequence2
```

The `while` loop statement



— *https://www.learnbyexample.org/python-while-loop/*

The `while` loop statement

The `while` loop statement allows to execute one or more instructions repeatedly.

The repetitive execution goes on until a certain condition remains true.

The `while` loop statement evaluates each time an expression that returns a `boolean` value.

If the condition is false at the loop begin, it can happen that the loop executes zero times.

The while loop statement: example

```
if __name__ == '__main__':

    limit = int(input("Set the max limit: "))

    x = 0
    while x <= limit:
        print(str(x) + " ", end="")
        x += 1

    print("\nEnd")
```

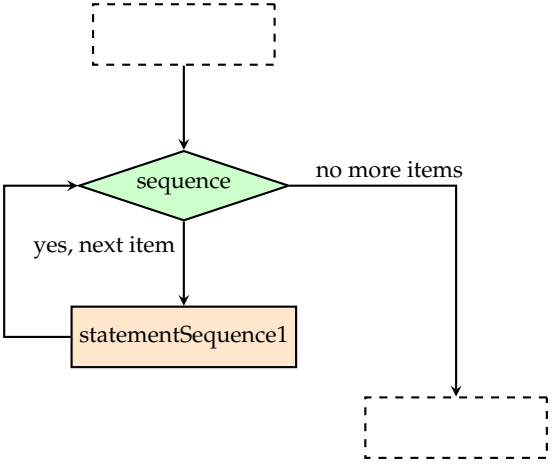
Output 1:

```
Set the max limit: 5
0 1 2 3 4 5
End
```

Output 2:

```
Set the max limit: -5
End
```

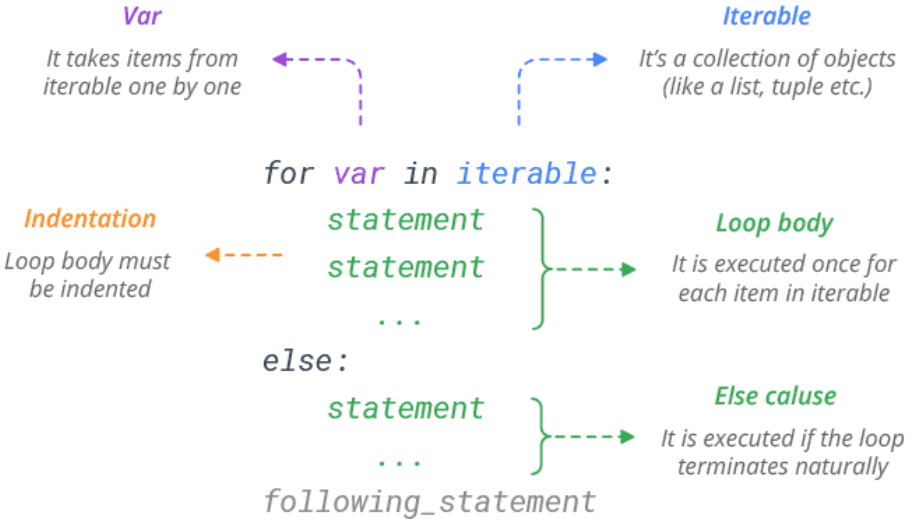
The for statement



General loop format:

```
for item in sequence:
  statementsSequence1
  if test: # optional
    break
  if test: # optional
    continue
else: # optional
  statementsSequence2
```

The for statement



— *https://www.learnbyexample.org/python-for-loop/*

The for statement

The `for` statement gives us a compact solution to iterate in a range of values.

In each `for` statement we specify:

- A variable we want to use to assign the current sequence item
- A sequence of values that we want to iterate on
- A body containing the statements that we want to execute at each iteration

Optionally, we can also specify an `else` statement, which body will be executed when we exit the loop naturally. We will enter more in detail about exiting a loop later.

The execution of the loop goes on until the sequence has values available.

The for statement: example

```
if __name__ == '__main__':
    years = int(input("Number of years: "))
    balance = float(input("Initial balance: "))
    tax = float(input("Interest tax: "))

    for i in range(years):
        balance += balance * tax
        print("Year: " + str(i+1) + " -> Balance: " + str(balance))
```

Output:

```
Number of years: 2
Initial balance: 1200
Interest tax: .02
Year: 1 -> Balance: 1224.0
Year: 2 -> Balance: 1248.48
```

The `for` statement: loop with a counter

One of the typical usage of the `for` statement is that of repeating an operation based on a counter.

As seen before, in the loop we need to provide a sequence of elements that will be iterated.

Let's assume that we want to print the values between a min (x) value and a max (y) value.

We need to provide to our loop a sequence that looks like this:

```
x, x+1, x+2, ..., y
```

The for statement: loop with a counter

In Python, for this purpose, we can exploit the method `range()`.

By calling the method `range(min, max)` passing our desired min (included) and max (not included) value, we obtain an iterable that we can use as sequence in the loop.

If we pass only one value, the min will be set to `0` by default.

Example with min and max:

```
for i in range(3, 7):
    print(str(i) + " ", end='')
```

Output:

```
3 4 5 6
```

Example with max only:

```
for i in range(4):
    print("Number: " + str(i))
```

Output:

```
0 1 2 3
```

The for statement: loop with a counter example

Calculation of the factorial:

```
if __name__ == '__main__':

    factorial = 1
    for i in range(1,10):
        factorial *= i
        print(str(i) + "! = " + str(factorial))
```

Count double letters occurrence:

```
if __name__ == '__main__':
    sentence = "Will it be a success?"
    cnt = 0
    for i in range(1, len(sentence)):
        if sentence[i] == sentence[i - 1]:
            cnt += 1
    print("Number of doubles = " + str(cnt))
```

The for statement: loop with a counter

When using `range()`, the increment (step) between one value and the next is 1 by default.

If needed, we can also pass a third parameter in order to specify the desired step.

Example with step equal to 2:

```
for i in range(3, 10, 2):
  print(str(i) + " ", end='')
```

Output:

```
3 5 7 9
```

Nested Control Flow Statements

If necessary, it is possible to nest more than one control flow statements one inside the other.

```
k = 0
for i in range(40):
    for j in range(20):
        k += i - j
    for j in range(20):
        k -= j + 1
```

```
for i in range(40):
    if i < 10:
        # ...
    elif i < 20:
        # ...
```

Nested Control Flow Statements: example

We want to show the multiplication tables from one to twelve.

1	2	3	4	5	6	7	8	9	10	11	12
2	4	6	8	10	12	14	16	18	20	22	24
3	6	9	12	15	18	21	24	27	30	33	36
4	8	12	16	20	24	28	32	36	40	44	48
5	10	15	20	25	30	35	40	45	50	55	60
6	12	18	24	30	36	42	48	54	60	66	72
7	14	21	28	35	42	49	56	63	70	77	84
8	16	24	32	40	48	56	64	72	80	88	96
9	18	27	36	45	54	63	72	81	90	99	108
10	20	30	40	50	60	70	80	90	100	110	120
11	22	33	44	55	66	77	88	99	110	121	132
12	24	36	48	60	72	84	96	108	120	132	144

Nested Control Flow Statements: example

Possible solution:

```
if __name__ == '__main__':
    # Navigate all the rows
    for row in range(1, 13):
        # Navigate al the columns of each row
        for col in range(1, 13):
            # Show the current result
            print(str(row * col) + "\t", end='');
        # start a new line
        print()
```

Other loop related statements

As already mentioned, there are some other optional statements that can be used in relation with loops.

- `break`: Jumps out of the closest enclosing loop (past the entire loop statement)
- `continue`: Jumps to the top of the closest enclosing loop (to the loop's header line)
- `else`: Runs if and only if the loop is exited normally (i.e., without hitting a break)
- `pass`: Does nothing at all: it's an empty statement placeholder

Other loop related statements: `break`

The `break` statement is used to jump out of the current loop.

It causes the immediate exit from the loop.

It can be used both in `while` and `for` loops

The code that follows a break statement inside the loop will not be executed.

Example, end program when user enter "exit":

```
if __name__ == '__main__':
    while True:
        name = input("Enter your name or type exit to end program: ")
        if name == "exit":
            print("bye bye")
            break
        age = input("Enter age: ")
        print("Hello " + name + " => " + age)
```

Other loop related statements: `continue`

The `continue` statement is used to jump at the beginning (top) of the current loop.

It can be used both in `while` and `for` loops.

The code that follows a continue statement inside the loop will not be executed.

Example, skip odd numbers:

```
if __name__ == '__main__':

    for i in range(100):
        # skip odd number
        if i % 2 != 0:
            continue
        print(str(i))
```

Other loop related statements: else

The else statement, combined with a loop statement, is used to specify statements that must be executed only when the loop exit naturally. The body of the else statement will not be executed when we exit the loop using a break statement.

If the loop exit condition is already false at the beginning of the loop, the else body will be executed.

Example, prime numbers:

```
if __name__ == '__main__':
    y = int(input("Insert a number: "))
    x = y // 2
    while x > 1:
        if y % x == 0:
            print(str(y) + " has factor " + str(x))
            break
        x -= 1
    else:
        print(str(y) + " is prime")
```

Other loop related statements: pass

The pass statement is used to as a placeholder where the syntax requires a statement, but there is no operation to carry on.

It basically does nothing at all, it is an empty statement.

We will see some use cases later during the course.

Summary

- Flow control statements
- Selection statement (`if ...elif ...else`
- Selection statement (`ternary operator`)
- Loop statement (`while`)
- Loop statement (`for`)
- The `range()` method
- Nested control flow statements
- Statement `continue`
- Statement `break`
- Statement `else`
- Statement `pass`

Bibliography

- Learning Python 5th edition, Oreilly - Mark Lutz: Chapters 12, 13
- Python Crash Course, no starch press - Eric Matthes: Chapters 5, 7
- Python Official Documentation: https://docs.python.org/3/tutorial/
- Python Wiki: https://wiki.python.org/
- Real Python: https://realpython.com
- LearnByExample: https://www.learnbyexample.org/python/