浙江理工大学
ZHEJIANG SCI-TECH UNIVERSITY

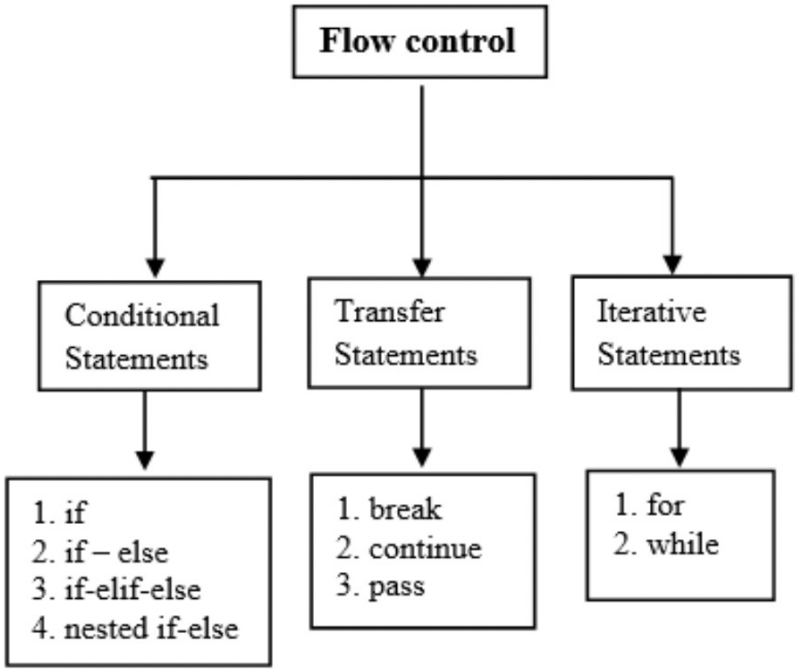
5. Control Flow (1)

《Python programming》 / Lecturer : Zhiyi Luo (罗志一)

School of Computer Science and Technology
计算机科学与技术学院

Control Flow Statements

- Conditional statements
- Iterative statements
- Transfer statements



Change the normal flow of execution

Conditional Statements

- In Python, condition statements act depending on whether a given condition is true or false. You can execute different blocks of codes depending on the outcome of a condition.
  - if statement
  - if-else
  - if-elif-else
  - nested if-else

If Statement

- If statement is the simplest form. It takes a condition and evaluates to either True or False.
- If the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and the controller moves to the next line.

```
if condition:
    statement 1
    statement 2
    statement n
```

syntax

If Statement

```
number = 6
if number > 5:
    # Calculate square
    print(number * number)
print('Next lines of code')
```

If-else Statement

- The if-else checks the condition and executes the if block of code when the condition is True, and if the condition is False, it will execute the else block or code.
- If the condition is True, then the True block of code will be executed, and if the condition is False, then the block of code is skipped, and the controller moves to the next line.

```
if condition:
    statement 1
else:
    statement 2
```

syntax

If-else Statement

```
password = input('Enter password ')

if password == "PYnative@#29":
    print("Correct password")
else:
    print("Incorrect Password")
```

Chain multiple if statement

- The if-elif-else condition statement has an elif blocks to chain multiple conditions one after another.
- With the help of if-elif-else we can make a tricky decision. The elif statement checks multiple conditions one by one and if the condition fulfills, then executes that code.

```
if condition-1:
     statement 1
elif condition-2:
     statement 2
elif condition-3:
     statement 3
     ...
else:
     statement
```

If-else Statement

```
def user_check(choice):
    if choice == 1:
        print("Admin")
    elif choice == 2:
        print("Editor")
    elif choice == 3:
        print("Guest")
    else:
        print("Wrong entry")

user_check(1)
user_check(2)
user_check(3)
user_check(4)
```

Nested if-else statement

- The nested if-else statement is an if statement inside another if-else statement. It is allowed to put any number of if statements in another if statement.Indentation is the only way to differentiate the level of nesting. The nested if-else is useful when we want to make a series of decisions.

syntax

```
if conditon_outer:
    if condition_inner:
        statement of inner if
    else:
        statement of inner else:
    statement ot outer if
else:
    Outer else
statement outside if block
```

Nested if-else statement

```
num1 = int(input('Enter first number '))
num2 = int(input('Enter second number '))

if num1 >= num2:
    if num1 == num2:
        print(num1, 'and', num2, 'are equal')
    else:
        print(num1, 'is greater than', num2)
else:
    print(num1, 'is smaller than', num2)
```

Single statement suites

- Whenever we write a block of code with multiple if statements, indentation plays an important role. But sometimes, there is a situation where the block contains only a single line statement.
- Instead of writing a block after the colon, we can write a statement immediately after the colon.

```
number = 56
if number > 0: print("positive")
else: print("negative")
```

Loop

- Loop repeats a specific block of code a fixed number of times. It reduces the repetition of lines of code, thus reducing the complexity of the code. Using for loops and while loops we can automate and repeat tasks in an efficient manner.
  - for loop
  - while loop

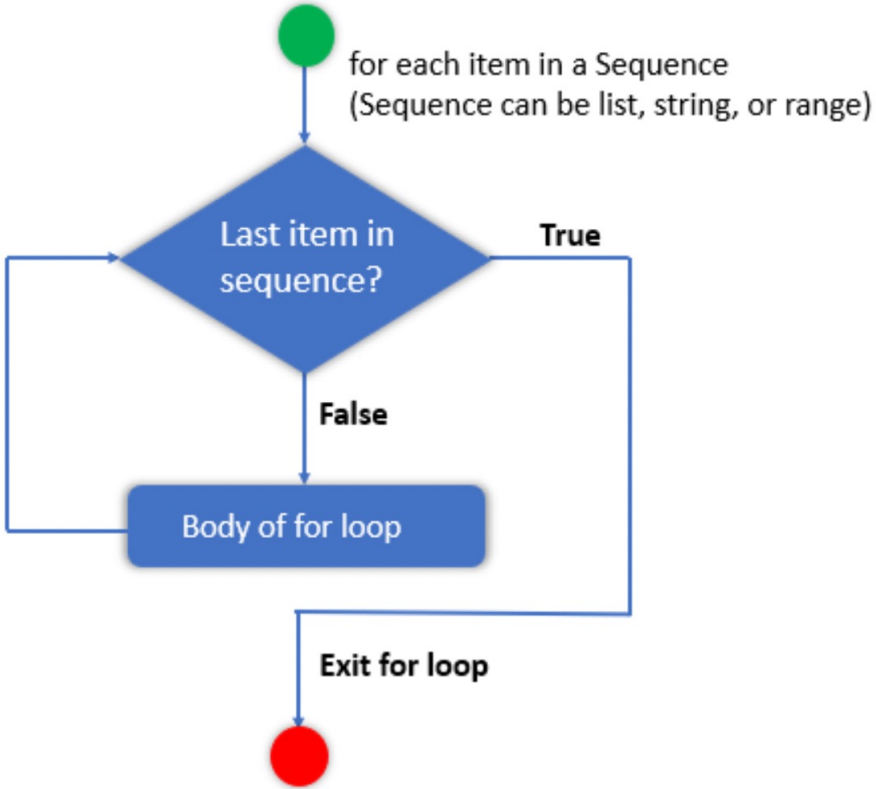
For loop

- Using for loop, we can iterate any sequence or iterable variable. The sequence can be string, list, dictionary, set or tuple.
- When we know how many times we wanted to run a loop, then we use count-controlled loops such as for loops.

syntax

```
for element in sequence:
    body of for loop
```

For loop


for each item in a Sequence
(Sequence can be list, string, or range)
Last item in sequence?
True
False
Body of for loop
Exit for loop

For loop

- Example: Print all even and odd numbers in a list
  - First, for loop statement iterates all the elements in a list
  - Next, the if statement checks whether the current number is even or not.

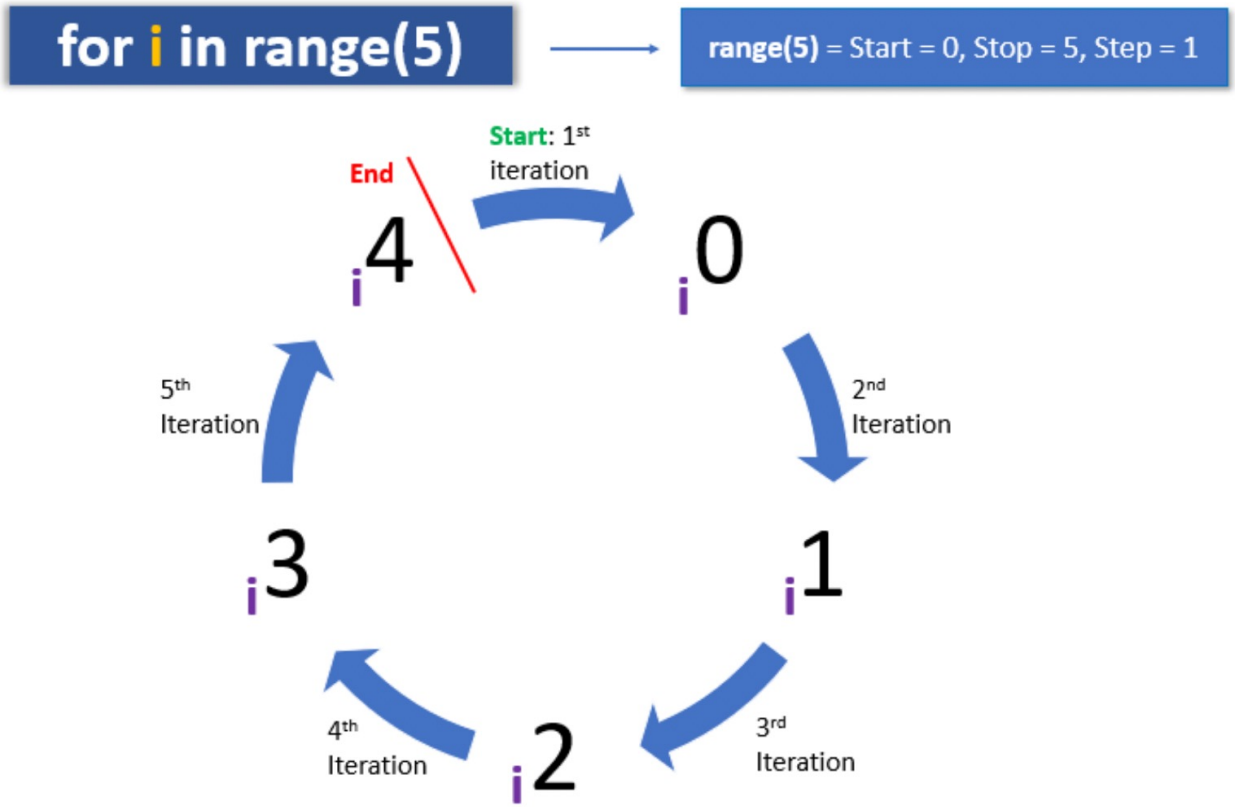
```
numbers = [1,2,3,4,5,6,7,8,9,10]

for i in numbers:
    if i % 2 == 0:
        print('Even Number:', i)
    else:
        print('Odd Number:', i)
```

an alternative way

```
for i in range(1, 11):
```

For loop with range()



For loop practice

- Use for loop to generate a list of numbers from 9 to 50 divisible by 2.
- Print sum of all even numbers from 10 to 20.

Note: The loop runs till it reaches the last element in the sequence. If it reaches the last element in the sequence, it exits the loop. Otherwise, it keeps on executing the statements present under the loop’s body

Transfer statements

- Transfer statements (loop control statements) are used when you want to exit a loop or skip a part of the loop based on the given condition.
  - break
  - continue
  - pass

Break statement

- The break statement is used to terminate the loop. You can use the break statement whenever you want to stop the loop. Just you need to type the break inside the loop after the statement, after which you want to break the loop.
- When the break statement is encountered, Python stops the current loop, and the control flow is transferred to the following line of code immediately following the loop.

Break statement

- Example: break the loop if number a number is greater than 15

```
numbers = [1, 4, 7, 8, 15, 20, 35, 45, 55]

for i in numbers:
    if i > 15:
        # break the loop
        break
    else:
        print(i)
```

Break statement

- Note: If the break statement is used inside a nested loop (loop inside another loop), it will terminate the innermost loop.

Continue statement

- The continue skips the current iteration of a loop and immediately jumps to the next iteration.
- Use the continue statement when you want to jump to the next iteration of the loop immediately.
- The continue statement skips a block of code in the loop for the current iteration only. It doesn’t terminate the loop but continues in the next iteration ignoring the specified block of code.

Continue statement

- Example: Count the total number of ‘m’ in a given string.

Pass statement

- The pass statement is a null statement, i.e., nothing happens when the statement is executed. Primarily it is used in empty functions or classes. When the interpreter finds a pass statement in the program, it returns no operation.
- Sometimes there is a situation in programming where we need to define a syntactically empty block. We can define that block with the pass keyword.

For loop with else block

- Same as the if statement, Python allows use to use an else statement along with for loop.
- For-loop can have the else block, which will be executed when the loop terminates **normally**.
- Else block will be skipped when:
  - For loop terminate abruptly
  - The break statement is used to break the loop
- Defining the else part with for loop is optional.

For loop with else block

```
count = 0
for i in range(1, 6):
    count = count + 1
    if count > 2:
        break
    else:
        print(i)
else:
    print("Done")
```

Nested for loops

- Nested for loop is a for loop inside another for loop.
- A nested loop has one loop inside of another. It is mainly used with two-dimensional arrays. For example, printing numbers or star patterns. Here outer loop is nothing but a row, and the inner loop is columns.
- In nested loops, the inner loop finishes all of its iteration for each iteration of the outer loop. i.e., For each iteration of the outer loop inner loop restart and completes all its iterations, then the next iteration of the outer loop begins.

Nested for loops

syntax

```
# outer for loop
for element in sequence
    # inner for loop
    for element in sequence:
        body of inner for loop
    body of outer for loop
other statements
```

Nested for loops

- Example: Nested for loop to print the following pattern

```
*
* *
* * *
* * * *
* * * * *
```

```
rows = 5
# outer loop
for i in range(1, rows + 1):
    # inner loop
    for j in range(1, i + 1):
        print("*", end=" ")
    print('')
```

Nested for loops