Flow Control
in Python

Massive Information & Knowledge Engineering

204113 Computer & Programming 4CPE

Arnon Rungsawang
Dept. of computer engineering
Kasetsart University

https://mike.cpe.ku.ac.th/204113

Version 2014

What is control flow?

- Control flow is the **order** in which individual statements, instructions, or blocks of code are executed or evaluated at runtime.
- The control flow of a Python program is regulated by conditional statements, loops, and function calls.

Sequential Program

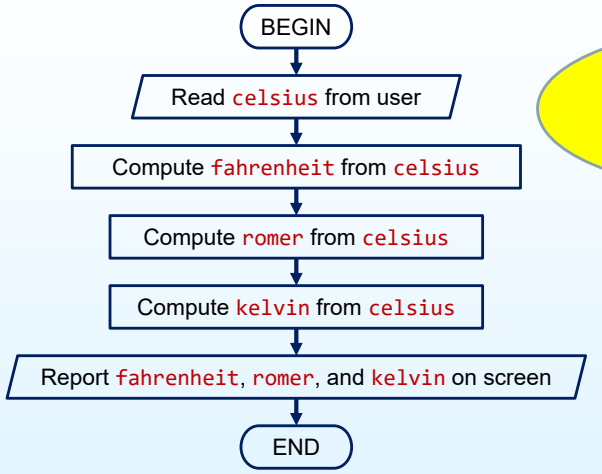
Task: Temperature conversion

- Relationship between temperature in degrees Celsius and degrees Fahrenheit is:

$$\frac{C}{5} = \frac{R}{4} = \frac{F-32}{9} = \frac{K-273.15}{5}$$

  - where *C* , *R*, *F*, and *K* are temperature values in °C, °R, °F, and Kelvin, respectively.
- Write a program that
  - Reads a temperature value in degrees Celsius
  - Then outputs the corresponding temperature values in degrees Fahrenheit, Romer, and Kelvin.

Temperature conversion



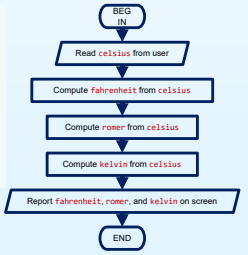
This diagram is called a ***Flowchart***

We will see it soon!

Temperature conversion (2)

```
celsius_str = input('Enter temp in degrees Celsius: ')
celsius = float(celsius_str)
fahrenheit = ((celsius*9)/5)+32
romer = (celsius*4)/5
kelvin = celsius+273.15
print(f'{celsius:.2f} degrees Celsius is equal to:')
print(f'{fahrenheit:.2f} degrees Fahrenheit')
print(f'{romer:.2f} degrees Romer')
print(f'{kelvin:.2f} degrees Kelvin')
```

```
Enter temp in degrees Celsius: 16
16.00 degrees Celsius is equal to:
60.80 degrees Fahrenheit
12.80 degrees Romer
289.15 degrees Kelvin
```



`if..else` statement

Python `if..else` statements

- Like in other programming language, in Python, the `if..else` statement help to create decision-making programs.
- We use the if statement to run a block code only when a certain condition is met.
- For example, assigning grades **(A, B, C)** based on marks obtained by a student.
  1. if the percentage is above **90**, assign grade **A**
  2. if the percentage is above **75**, assign grade **B**
  3. if the percentage is above **65**, assign grade **C**
- In Python, there are three forms of the `if..else` statement.
  1. `if` statement
  2. `if..else` statement
  3. `if..elif..else` statement

if statement

- The `if` statement evaluates condition.
  - If condition is evaluated to True, the code inside the body of `if` is executed.
  - Otherwise, the code inside the body of `if` is ignored or skipped.

```
# Condition is True
N = 10
if N > 0:
    print("N > 0")

# code after if
```

```
# Condition is False
N = -5
if N > 0:
    print("N > 0")

# code after if
```


if..else statement

- An `if` statement can have an optional `else` clause.
  - If condition is evaluated to True, the code inside the body of `if` is executed, and the code inside the body of `else` is skipped.
  - Otherwise, the code inside the body of `if` is ignored or skipped, but the code inside the body of `else` is executed.

```
# Condition is True
N = 10
if N > 0:
    print("N > 0")
else:
    print("Bla bla")

# code after if..else
```

```
# Condition is False
N = -5
if N > 0:
    print("N > 0")
else:
    print("Bla bla")

# code after if..else
```


if..elif..else statement

- The `if..else` statement is used to execute a block of code among two alternatives.
- However, if we need to make a choice between more than two alternatives, then we use the `if..elif..else` statement.

```
# 1st condition
N = 15
if N > 9 :
    print("> 9")
elif N < 0 :
    print("Negative")
else :
    print("0 to 9")

# code after…
```

```
# 2nd condition
N = -5
if N > 9 :
    print("> 9")
elif N < 0 :
    print("Negative")
else :
    print("0 to 9")

# code after…
```

```
# 3rd condition
N = 5
if N > 9 :
    print("> 9")
elif N < 0 :
    print("Negative")
else :
    print("0 to 9")

# code after…
```


Nested if statements

- We can also use an `if` statement inside of an `if` statement.

```
1  number = 5
2
3  # outer if statement
4  if (number >= 0):
5      # inner if statement
6      if number == 0:
7        print('Number is 0')
8
9      # inner else statement
10     else:
11         print('Number is positive')
12
13 # outer else statement
14 else:
15     print('Number is negative')
16
17 # Output: Number is positive
```

Nested `if` statements – possible scenarios

Each of these yellow boxes is actually **a single statement**.

```
pass
```

```
x = y+1
```

```
print('hi')
```

```
if x > y:
    m = x
```

```
if x > y:
    m = x
    z = x+y
```

```
if x > y:
    m = x
    z = x+y
else:
    m = y
```

```
if x > y:
    m = x
    z = x+y
else:
    m = y
    z = x-y
```

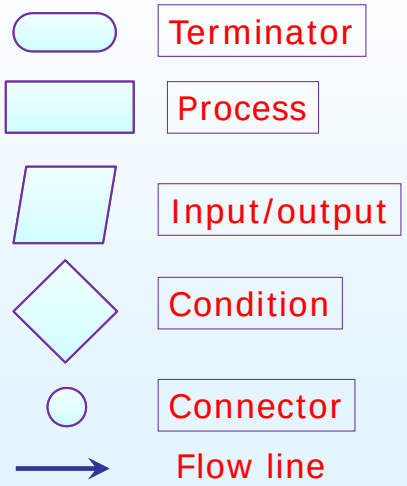
so each can be put here:

```
if condition :
    statement1
    statement2
else:
    statement3
    statement4
    statement5
```

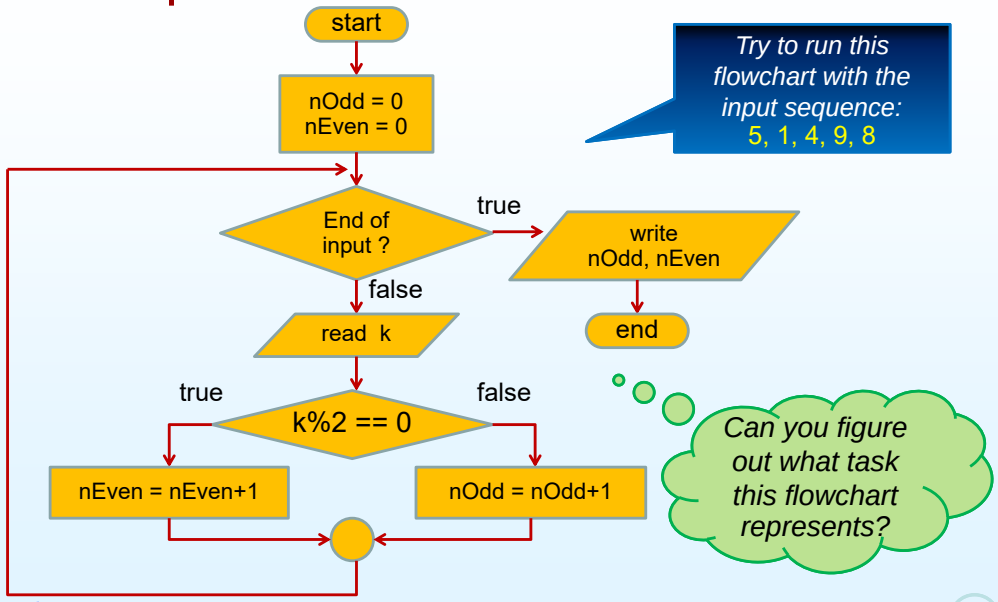
or here:

Flowcharts: Graphical representation of control flow

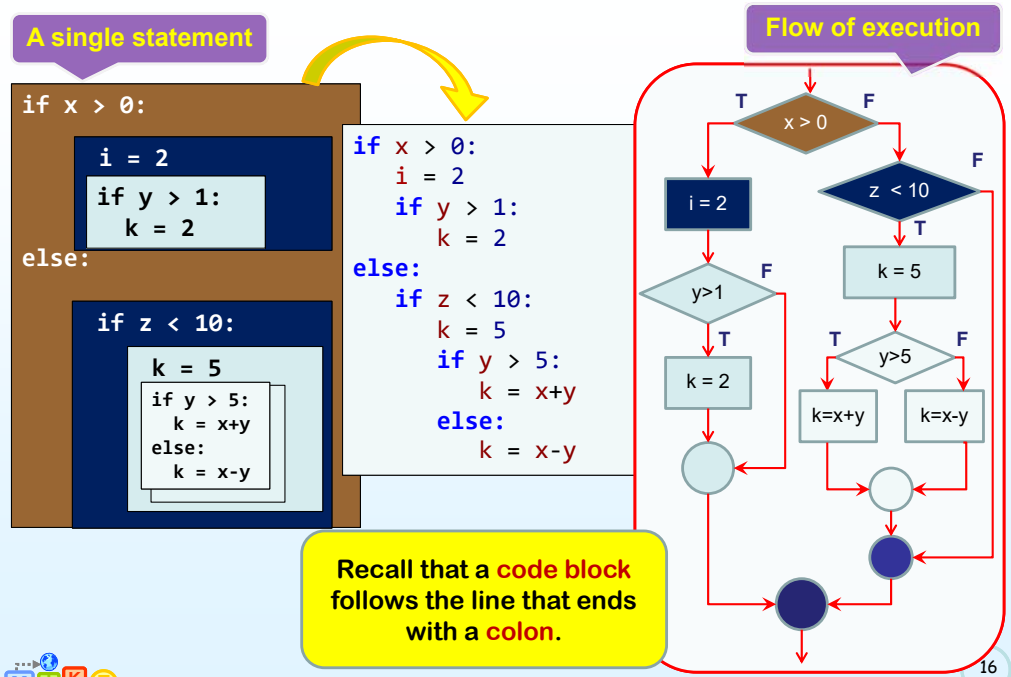
Basic flowchart symbols:



Sample flowchart

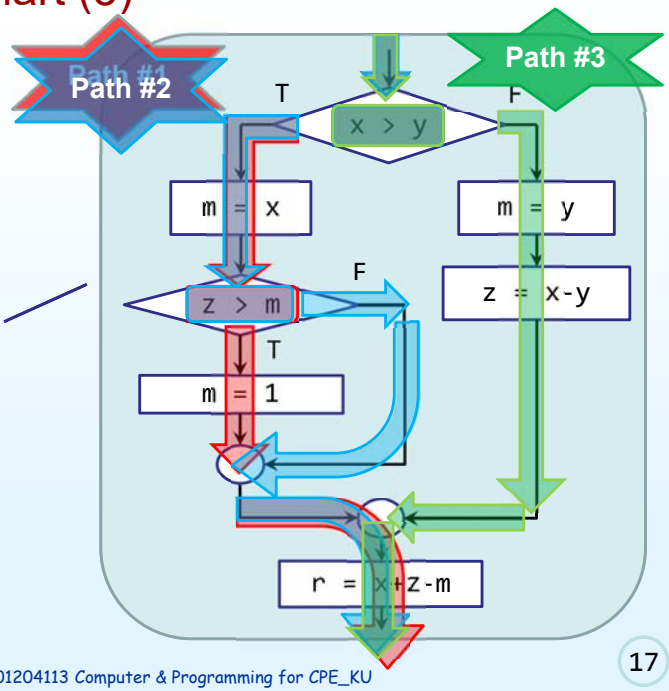


Sample flowchart (2)



Sample flowchart (3)

```
# in Python
if x > y:
    m = x
    if z > m:
        m = 1
else:
    m = y
    z = x-y
r = x+z-m
```



for loop

Python for loop

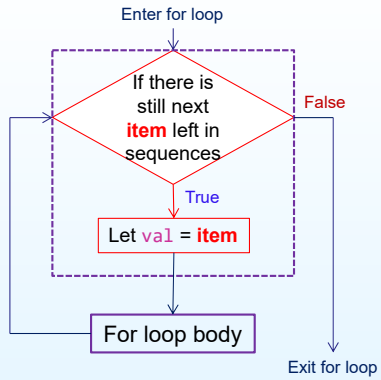
- In Python, a `for` loop is used to iterate over sequences such as lists, tuples, string, etc.

```
1  languages = ['Swift', 'Python', 'Go', 'JavaScript']
2
3  # run a loop for each item of the list
4  for language in languages:
5      print(language)
```

- In the above example, we first create a list called `languages`.
- Initially, the value of `language` is set to the first element of the list, i.e., swift, so the print inside the loop is executed.
- `language` is then updated with the next element of the list, and the print statement is executed again. This way the loop runs until the last element of the list is accessed.

for loop syntax and flowchart

```
for val in sequences:
    # for loop body
    statement(s)
```



- `val` accesses each item of sequence on each iteration.
- The loop continues until we reach the last item of the sequence.

Python range()

- The range() function return a sequence of numbers between the given range.
- Note that range() returns an immutable sequence of numbers that can be easily converted to lists, tuples, sets etc.

- Syntax:

```
range(start, stop, step)
```

  - The start and step parameters are optional.

range() with stop argument

- If we pass a single argument to range(), it means we are passing the stop argument.
- In this case, range() returns a sequence of numbers starting from 0 up to the number (but not including that number).

```
# numbers from 0 to 3 (4 is not included)
numbers = range(4)
print(list(numbers))    # [0, 1, 2, 3]

# if 0 or negative number is passed,
# we get an empty sequence
numbers = range(-4)
print(list(numbers))    # []
```


range() with start, stop, step arguments

- The step argument specifies the incrementation between two numbers, i.e., start (inclusive) up/down to stop (exclusive).
- Note that the default value of start is 0, and the default value of step is 1. That is why range(0,5,1) is equivalent to range(5).

```
# numbers from 2 to 10 with increment 3
numbers = range(2, 10, 3)
print(list(numbers))    # [2, 5, 8]

# numbers from 4 to -1 with increment of -1
numbers = range(4, -1, -1)
print(list(numbers))    # [4, 3, 2, 1, 0]

# numbers from 1 to 4 with increment of 1
# range(0, 5, 1) is equivalent to range(5)
numbers = range(0, 5, 1)
print(list(numbers))    # [0, 1, 2, 3, 4]
```


range() in for loop

```
s = 'Python'
lens = len(s)
for c in s:
    print(' '*lens, c)
    lens = lens -1

lens = len(s)
for i in range(len(s)):
    print(' '*(lens-i), s[i])
```

Using a `for` loop without accessing items

```
1 languages = ['Swift', 'Python', 'Go']
2
3 for language in languages:
4     print('Hello')
5     print('Hi')
```

- The loop runs three times as our list has three items. In each iteration, the loop body prints `Hello` and `Hi`. The items of the list are not used with the loop.

```
1 languages = ['Swift', 'Python', 'Go']
2
3 for _ in languages:
4     print('Hello')
5     print('Hi')
```

- If we do not intend to use items of a sequence within the loop, we can use _ symbol to denote that the elements of a sequence will not be used within the loop body.

`for` loop with `else`

- A `for` loop can have an optional `else` block.
- The `else` part is executed when the loop is exhausted (after the loop iterates through every item of a sequence).
- Note that the `else` block will **not** execute if the for loop is stopped by a `break` statement.

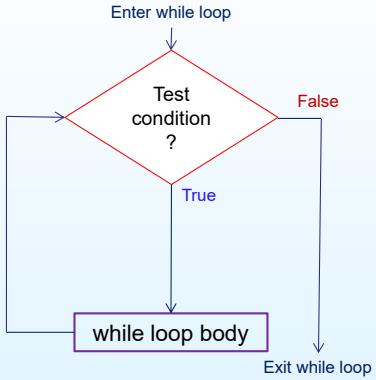
```
1 digits = [0, 1, 5]
2
3 for i in digits:
4     print(i)
5     # break
6 else:
7     print("No items left.")
```


`while` loop

Python `while` loop

- In Python, a `while` loop is used to run a block of code until a certain condition is met.

```
while condition:
    # while loop body
    statement
```



- First, the `while` loop evaluates the condition.
- If the condition evaluates to `True`, the code inside the while loop is executed.
- Condition is then evaluated again.
- This process continues until the condition is evaluated to `False`, the loop stops and exits.

Example of Python `while` loops

```python
# program to display numbers from 1 to 5

# initialize the variable
i = 1
n = 5

# while loop from i = 1 to 5
while i <= n:
    print(i)
    i = i + 1
```

```python
age = 32

# test condition is always True
while age > 18:
    print('You can vote')
```

```python
# program to calculate the sum of numbers
# until the user enters zero

total = 0

number = int(input('Enter a number: '))

# add numbers until number is zero
while number != 0:
    total += number    # total = total + number

    # take integer input again
    number = int(input('Enter a number: '))

print('total =', total)
```


`while` loop with `else`

- A `while` loop may have an optional `else` block.
- The `else` part is executed after the condition of the loop evaluates to `False`.
- Note that if the `break` statement has been executed, the `while` loop will be terminated, and the `else` part will be ignored.

```python
counter = 0

while counter < 3:
    print('Inside loop')
    counter = counter + 1
    # if counter == 2:
    #     break
else:
    print('Inside else')
```


Python `for` vs. `while` loops

```python
# this loop is iterated 4 times
for i in range(4):
    print(i)
```

- The `for` loop is usually used when the number of iterations is known.

```python
while condition:
    # run code until the
    # condition evaluates to False
```

- The `while` loop is usually used when the number of iterations is unknown. The loop will be terminated by a certain condition.

`break` statement

- The `break` statement is used to terminate the loop immediately when it is encountered.

```
for val in sequence:
  # code in if_body
  if condition:
    break
  # code in if_body
# code next to for..
```

```
while condition:
  # code in while_body
  if condition:
    break
  # code in while_body
# code next to while..
```

break statement (2)

```
# find first 5 multiples of 6

i = 1
while i <= 10:
    print('6 * ',(i), '=',6 * i)
    if i >= 5:
        break
    i = i + 1
```

```
# find first 5 multiples of 6

for i in range(1, 11):
    print('6 * ',(i), '=',6 * i)
    if i >= 5:
        break
```


continue statement

- The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

```
for val in sequence:
  # code in if_body
  if condition:
    continue
  # code in if_body
# code next to for..
```

```
while condition:
  # code in while_body
  if condition:
    continue
  # code in while_body
# code next to while..
```


continue statement (2)

```
# print odd numbers from 1 to 10
num = 0

while num < 10:
    num += 1
    if (num % 2) == 0:
        continue
    print(num)
```

```
# print odd numbers from 1 to 10
for num in range(10):
    if (num % 2) == 0:
        continue
    print(num)
```


pass statement

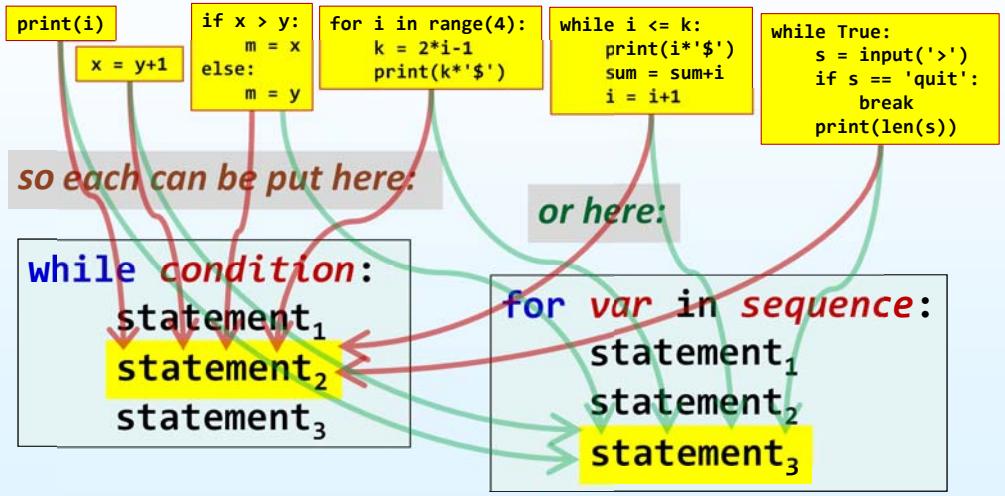
- pass is a null statement which can be used as a placeholder for future code.
- Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. In such case, we can use pass statement.
- When the pass statement is executed, it results in no operation (NOP).

```
n = 10

# use pass inside if statement
if n >= 10:
    # will implement later
    pass

print('Hello')
```

```
def function(args):
    pass

class Example:
    pass
```

Nested loop

Each of these yellow boxes is actually **a single statement**.



Sample Problem Solving using Control Flow

Ladder `if`

```
1  time = 19
2
3  if time == 9 :
4     print("On time")
5
6  if time > 9 and time <= 19:
7     print("10 minutes late")
8
9  if time > 19 and time <= 39:
10   print("30 minutes late")
11
12 if time > 39:
13   print("Zero marks")
```

```
1  time = 19
2
3  if time == 9 :
4     print("On time")
5  else:
6      if time > 9 and time <= 19:
7          print("10 minutes late")
8      else:
9          if time > 19 and time <= 39:
10             print("30 minutes late")
11         else:
12             if time > 39:
13                 print("Zero marks")
```


Ladder `if` (2)

```
1  time = 19
2
3  if time == 9 :
4     print("On time")
5  else:
6      if time > 9 and time <= 19:
7          print("10 minutes late")
8      else:
9          if time > 19 and time <= 39:
10             print("30 minutes late")
11         else:
12             if time > 39:
13                 print("Zero marks")
```

```
1  time = 19
2
3  if time == 9 :
4     print("On time")
5
6  elif time > 9 and time <= 19:
7     print("10 minutes late")
8
9  elif time > 19 and time <= 39:
10   print("30 minutes late")
11
12 else:
13   print("Zero marks")
```

Which number is bigger or the biggest?

```python
n1 = int(input('Enter 1st number '))
n2 = int(input('Enter 2nd number '))

if n1 >= n2:
  if n1 == n2:
    print(n1, 'and', n2, 'are equal')
  else:
    print(n1, 'is greater than', n2)
else:
  print(n1, 'is smaller than', n2)
```

```python
a, b, c = 20, 10, 15
if a > b:
   if a > c:
      print("a value is big")
   else:
      print("c value is big")
elif b > c:
    print("b value is big")
else:
     print("c is big")
```


Even and Odd numbers

```python
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found an odd number", num)
```


Student arrival time

```python
students_arrival_time = [9,25,39,45,9,75,84,2,18,13]

for student_counter in range(len(students_arrival_time)):

  if students_arrival_time[student_counter] == 9 :
    print("On time")

  elif students_arrival_time[student_counter] > 9 \
       and students_arrival_time[student_counter] <= 19:
    print("10 minutes late")

  elif students_arrival_time[student_counter] > 19 \
       and students_arrival_time[student_counter] <= 39:
    print("30 minutes late")

  else:
    print("Zero marks")
```

Challenge: Could you rewrite this code using while loop?

Fibonacci series

```python
n = 10
a, b = 0, 1
while a < n:
    print(a, end=' ')
    a, b = b, a+b
print()
```

```python
n = 10
a, b = 0, 1
while True:
    print(a, end=' ')
    a, b = b, a+b
    if not(a < n):
        break
print()
```

Prime numbers

```
for n in range(2, 10):
  for x in range(2, n):
    if n % x == 0:
      print(n, 'equals', x, '*', n//x)
      break
  else:
    # loop fell through without finding a factor
    print(n, 'is a prime number')
```


Granted access

```
while True:
  name = input('Who are you? ')
  if name != 'Joe':
    continue
  password = input('Password? (It is a fish.): ')
  if password == 'swordfish':
    break
print('Access granted.')
```


To be continue..

つづく