HO CHI MINH CITY UNIVERSITY OF TRANSPORT
FACULTY OF INFORMATION TECHNOLOGY
SOFTWARE ENGINEERING DEPARTMENT

CHAPTER 7
CONTROL FLOW STATEMENTS

CONTENTS

1. Control Flow
2. Control Flow Statements
3. Conditional Statements
4. Looping Statements
5. Break Statement
6. Continue Statement

1. Control Flow

- A program's **control flow** is the order in which the program's code executes.
- All of the programs that we have examined to this point have a simple flow of control:
  - The statements are executed one after the other in exactly the order given.

  →**sequential execution**.
- What if you wanted to change the flow of how it works?
  - For example, you want the program to take some decisions depending on different situations, such as printing 'Good Morning' or 'Good Evening' depending on the time of the day?

2. Control Flow Statements

- Most programs have a complicated structure.
- Statements may/may not be executed depending on certain conditions, or groups of statements are executed multiple times.
- Control flow statements control the way the computer executes programs.
- There are three control flow statements in Python:
  - **if** statement
  - **for** statement
  - **while** statment

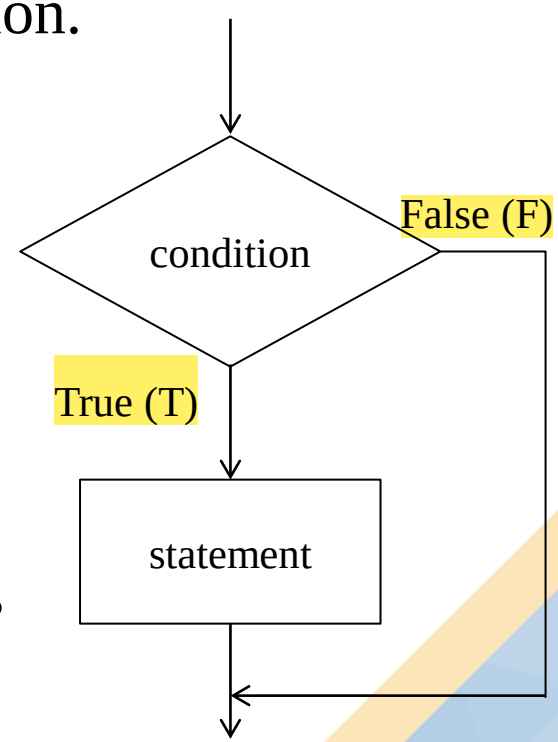
3. Conditional Statements

- Conditional statements are used to alter the flow of control in the program.
- They allow for decision making based on condition.

1. if Statement

- Syntax

```
if <condition>:
    <statement>
```



- <condition>: an expression evaluated.
- <statement> is a statement or block of statements which *must be indented*.
- There is syntax for branching execution

Conditional Statements

In [10]:
```
import random
num = random.randint(1,10)
guess = eval(input('Enter your guess:'))
if guess == num:
    print('You got it!!!')
```

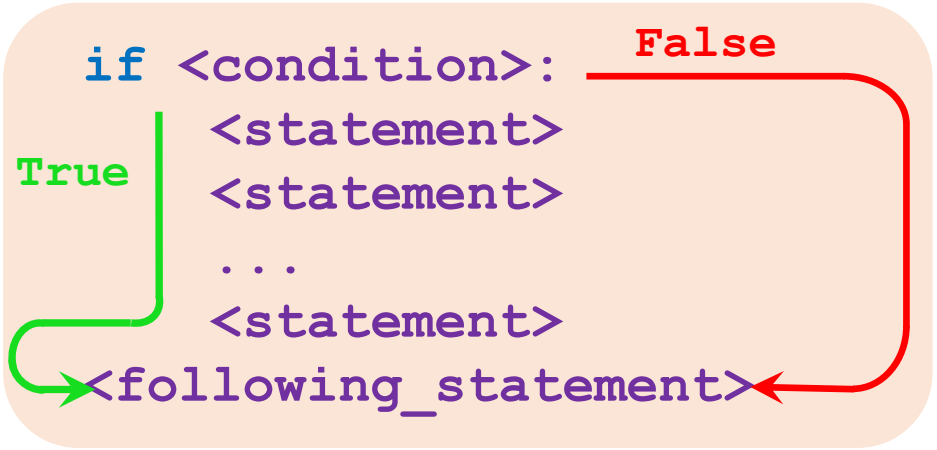
```
Enter your guess:6
You got it!!!
```

In [8]:
```
days = int(input("How many days in a leap year? "))
if days == 366:
    print("Good job!")
print("Congrats!")
```

```
How many days in a leap year? 366
Good job!
Congrats!
```

3. Conditional Statements

- **Grouping Statements: Indentation and Blocks**
  - Syntax




```
x = eval(input('Enter a number: '))
if x > 0:
    print("Only printed when x is positive; x =", x)
    print("Also only printed when x is positive; x =", x)
print("Always printed, regardless of x's value; x =", x)
```

```
Enter a number: -1
Always printed, regardless of x's value; x = -1
```

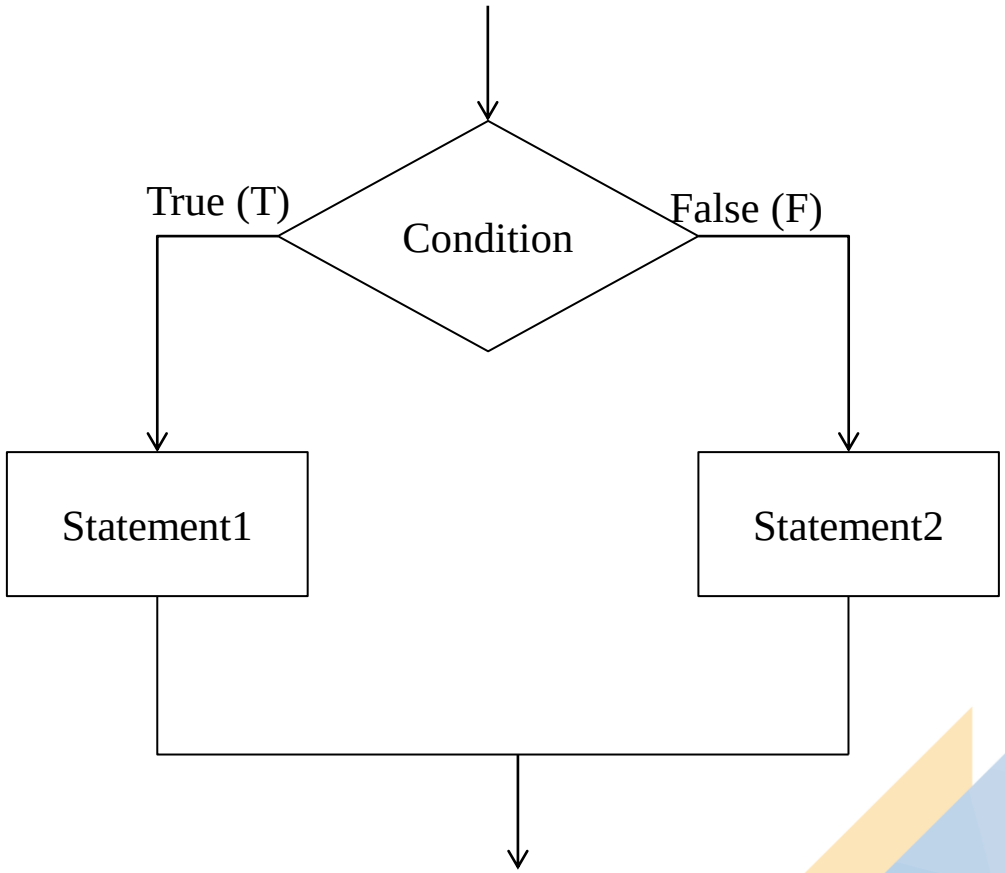
Conditional Statements

2. if…else Statement

- Syntax

```
if <condition>:
    <statement_1>
else:
    <statement_2>
```

- Evaluate a condition and take one path if it is true but specify an alternative path if it is not.



Conditional Statements

In [6]:
```
import math
radius = eval(input("Enter radius of circle:"))
if radius < 0:
    print("Incorrect input")
else:
    area = radius * radius * math.pi
    print(f"Area is {area:.2f}")
```

```
Enter radius of circle:1.5
Area is 7.07
```

In [15]:
```
num = eval(input('Enter a integer number: '))
if (num % 2) == 0:
    print(f'{num} is even.')
else:
    print(f'{num} is odd.')
```

```
Enter a integer number: 5
5 is odd.
```

Conditional Statements

3. if…elif…else Statement

- There is syntax for branching execution based on several conditions.
- Use one or more **elif** clauses.
- The **else** clause is optional. If it is present, there can be only one, and it must be specified last

```
if <condition_1>:
    <statement_1>
elif <condition_2>:
    <statement_2>
elif <condition_3>:
    <statement_3>
    ...
else:
    <statement_n>
```

Conditional Statements

In [16]:
```
num = eval(input('Enter a integer number: '))
if num == 0:
    print(num,"is zero")
elif num > 0:
    print(num,"is positive")
else:
    print(num,"is negative")
```

```
Enter a integer number: 5
5 is positive
```

In [21]:
```
# Print number of days in a given month
month = eval(input("Enter a month: "))
if month==2:
    print("This month has 28 or 29 days.")
elif month==4 or month==6 or month== 9 or month==11:
    print("This month has 30 days.")
elif month==1 or month==3 or month==5 or month==7\
or month==8 or month==10 or month==12:
    print("This month has 31 days.")
else:
    print("Invalid input! Please enter month number between (1-12)")
```

```
Enter a month: 3
This month has 31 days.
```

Conditional Statements

4. Nested if Statement

```
if <condition_1>:
    if <condition_1.1>:
       <statement_1.1>
    elif <condition_1.2>:
      <statement_1.2>
    else:
        <statement_1.3>
elif <condition_2>:
    <statement_2>
...
else:
    <statement_n>
```

Conditional Statements

In [25]:
```
# find root of linear equation ax + b = 0
a = eval(input("Enter a = "))
b = eval(input("Enter b = "))
if a == 0:
    if b == 0:
        print("Infinitely many solutions")
    else:
        print("Equation has no solution")
else:
    print("Root is {:.2f}".format(-b/a))
```

```
Enter a = 2
Enter b = 3
Root is -1.50
```

Common Errors in Conditional Statements

Note:

- Most common errors in conditional statements are caused by *incorrect indentation.*

```
import math
radius = eval(input("Enter radius:"))
if radius >= 0:
    area = radius * radius * math.pi
print(f"Area is {area:.2f}")
```

```
Enter radius:-5

---------------------------------------
NameError
<ipython-input-1-e821993ff7c9> in <mod
      3 if radius >= 0:
      4     area = radius * radius * m
----> 5 print(f"Area is {area:.2f}")

NameError: name 'area' is not defined
```

```
import math
radius = eval(input("Enter radius:"))
if radius >= 0:
    area = radius * radius * math.pi
    print(f"Area is {area:.2f}")
```

```
Enter radius:-5
```

Common Errors in Conditional Statements

Note:

- Which **if** clause is matched by the **else** clause?

```
i = 1
j = 2
k = 3
if i > j:
    if i > k:
        print('A')
else:
    print('B')
```

```
B
```

```
i = 1
j = 2
k = 3
if i > j:
    if i > k:
        print('A')
    else:
        print('B')
```

Conditional Statements

5. if…else Statement In One Line

- Syntax

```
if <condition>: <statement>
```

- There can even be more than one <statement> on the same line, separated by semicolons:

```
if <condition>: <st_1>; <st_2>;…;<st_n>
```

Conditional Statements

6. Conditional expressions

- Python supports one additional decision-making entity called a conditional expression.
- Syntax

```
<expr_T> if <conditional_expr> else <expr_F>
```

- <conditional_expr> is evaluated first. If it is true, the expression evaluates to <expr_T>. If it is false, the expression evaluates to <expr_F>.

Conditional Statements

In [30]:
```
grade = eval(input("Enter your grade: "))
if grade < 50:
    outcome = "failed"
else:
    outcome = "pass"
print(f'You {outcome} the quiz with a grade of {grade}')
```

```
Enter your grade: 51
You pass the quiz with a grade of 51
```

In [33]:
```
grade = eval(input("Enter your grade: "))
outcome = "failed" if grade < 50 else "pass"
print(f'You {outcome} the quiz with a grade of {grade}')
```

```
Enter your grade: 51
You pass the quiz with a grade of 51
```

Conditional Statements

pass Statement

- No limit on the number of statements under the two clauses of an if statement.But, have to be *at least one statement in each block.*
- Occasionally, it is useful to have a section with no statements (*empty block*).
  - a place keeper, scaffolding, for code you haven't written yet

→ use the **pass** statement

```
if <condition>:
    pass
else:
    pass
```

```
In [1]: if True:
            pass
        print('End!')

End!
```

4. Looping Statements

- Repeating similar tasks without making errors is something that computers do well and people do poorly.
- Repeated execution of a set of statements is called **iteration.**
- There are two types of iteration:
  - **Definite iteration**, in which the number of repetitions is specified explicitly in advance.
  - **Indefinite iteration**, in which the code block executes until some condition is met.

```
count = 0
while count < 3:
    print('Python programming is fun!')
    count += 1
```

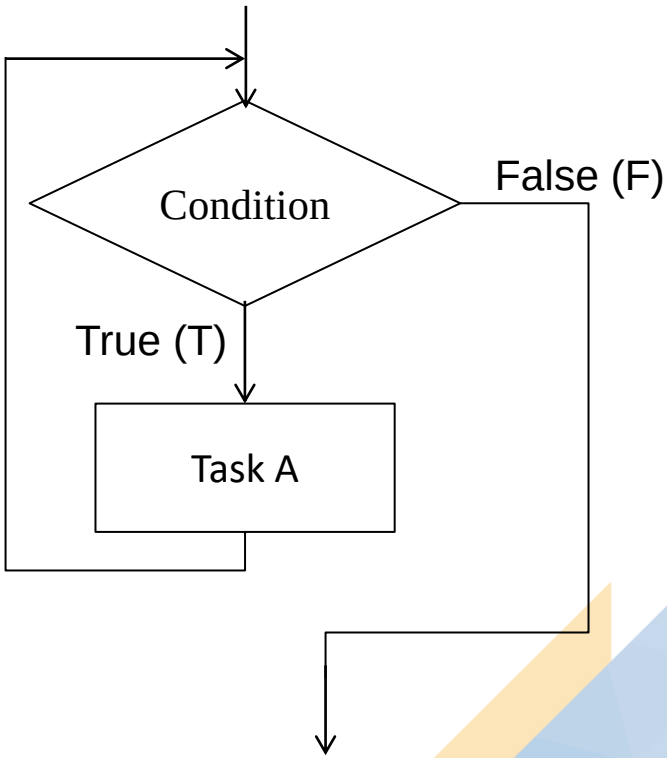
```
Python programming is fun!
Python programming is fun!
Python programming is fun!
```

4. Looping Statements

1. for Statement

- Used for **definite iteration** when you know exactly how many times the loop body needs to be executed.
- Syntax:

```
for <var> in <sequence>:
    <statement(s)>
```



- <sequence>: holds multiple items of data.
- <var>: a control variable that takes on the values of the next element in <sequence> each time through the loop
- <statement(s)>: is loop body, *must be indented*.

4. Looping Statements

In [1]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

In [5]:
```
for i in range(5,0,-1):
    print(i)
```

```
5
4
3
2
1
```

In [4]:
```
s = "Python"
for ch in s:
    print(ch,end=' ')
```

```
P y t h o n
```

In [2]:
```
animals = ['dog', 'cat', 'cow']
for a in animals:
    print(a)
```

```
dog
cat
cow
```

4. Looping Statements

In [8]:
```
#Calculate sum S = 1 + 2 + … + n
n = eval(input("n = "))
s = 0
for i in range(1,n+1):
    s = s + i
    i += 1
print('Sum is',s)
```

```
n = 5
Sum is 15
```

4. Looping Statements

2. while Statement

- Used for **definite and indefinite iteration**.
- Syntax:

```
while <condition>:
    <statement(s)>
```

- A **while** loop executes statements repeatedly as long as a <condition> remains true.
- <statement(s)>: is loop body, *must be indented*.

4. Looping Statements

```
#Calculate sum S = 1 + 2 + … + n
n = eval(input("n = "))
s = 0
i = 1
while i <= n:
    s = s + i
    i += 1
print('Sum is',s)
```

```
n = 5
Sum is 15
```

```
# Print all digits of integer number
n = eval(input("Enter n:"))
while n != 0:
    print(n%10)
    n = n//10
```

```
Enter n:123
3
2
1
```

In [1]:
```
num = eval(input("Enter a positive number:"))
while num <= 0:
    num = eval(input("Please enter another number:"))
print(f"{num} is a positive number.")
```

```
Enter a positive number:-5
Please enter another number:-3
Please enter another number:2
2 is a positive number.
```

4. Looping Statements

3. Nested Loops

- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered and started anew.

```
while <condition>:
    <statement(s)>
    while <condition>:
        <statement(s)>
```

```
for <var> in <sequence>:
    <statement(s)>
    for <var> in <sequence>:
        <statement(s)>
```

4. Looping Statements

In [1]:
```
# Calculate sum: S = 1! + 2! + ... + n!
n = eval(input("Enter n = "))
i = 1
s = 0
while i <= n:
    # Find factorial of i: i!
    j = 1
    gt = 1
    while j <= i:
        gt = gt*j
        j += 1
    s = s + gt
    i += 1
print("Result is",s)
```

```
Enter n = 5
Result is 153
```

In [55]:
```
# Multiplication Table
n = 6
print(" "*3,end='')
for i in range(1,n):
    print(f'{i:4d}',end='')
print() # Jump to new line
print("-"*4*n)
for i in range(1,n):
    print(f'{i:2d}|',end='')
    for j in range(1,n):
        print(f'{i*j:4d}',end='')
    print() # Jump to new line
```

```
      1   2   3   4   5
------------------------
 1|   1   2   3   4   5
 2|   2   4   6   8  10
 3|   3   6   9  12  15
 4|   4   8  12  16  20
 5|   5  10  15  20  25
```

5. Break Statement

- The **break** statement is used to
  - immediately terminate a loop
  - break out of a **for** or **while** loop before the loop is finished
- **Note**:
  - In nested loops, break statement terminate the execution of the nearest enclosing loop in which it appears.

```
while True:
    num = eval(input("Enter a number:"))
    if num > 0:
        break
```

```
Enter a number:-5
Enter a number:-2
Enter a number:1
```

```
name = input("Enter your name:")
for c in name:
    if c == ' ':
        break
    print(c,end='')
```

```
Enter your name:Tran Thi My Tien
Tran
```

6. Continue Statement

- The **continue** statement
  - immediately terminates the current iteration
  - Jumps to the next iteration

```
# Sum all the integers from 1 to 20 except 10 and 11
s = 0
for i in range(1,21):
    if i == 10 or i == 11:
        continue
    s = s + i
print("Sum is", s)
```

```
Sum is 189
```