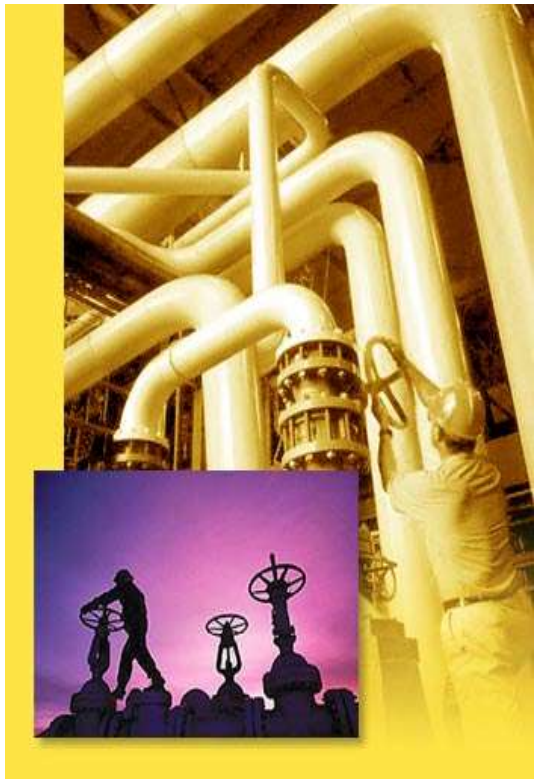
Chapter 2: Flow of Control

Stephen Huang

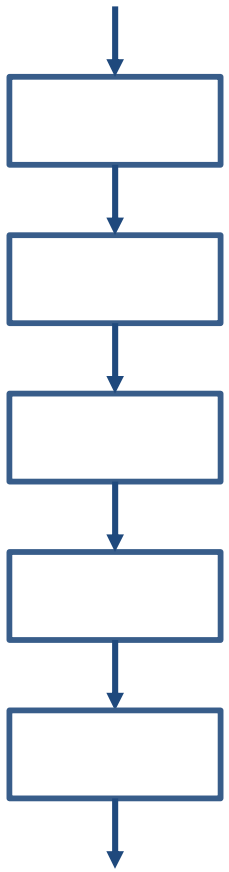
January 26, 2023

Contents

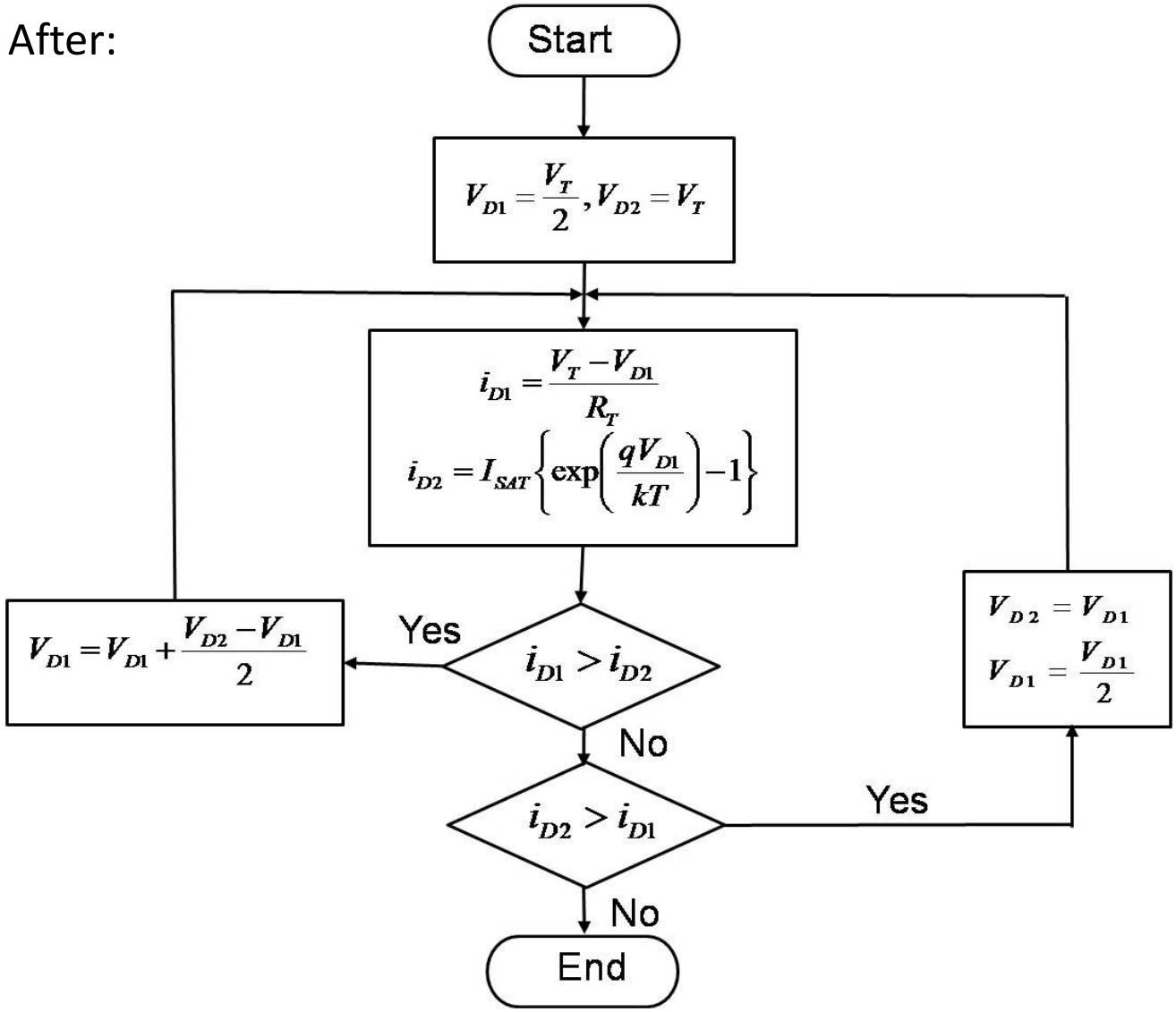


Complexity

Before:



After:

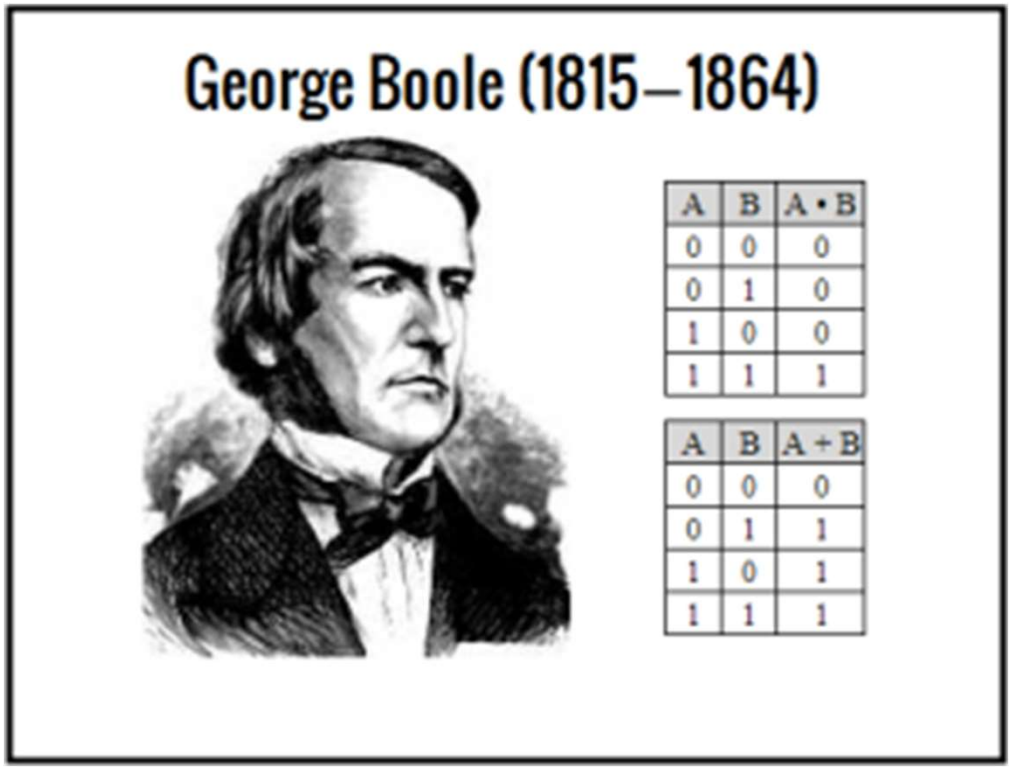


Simple IF

```
if <Boolean expression>:
    <statement>
    . . .
    <statement>
```

1. Boolean Expressions

- A variable of Boolean type (bool) is either "True" or "False".



Operators resulted in Boolean

Symbol	Meaning
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Logical Operators

- A Boolean (logical) expression is an expression that is either true or false.
- Expressions can be more complicated by connecting smaller (sub-)expressions with logical operators:
  - `and`
  - `or`
  - `not`

Logical Operators

- Just as with arithmetic expressions, Boolean expressions use brackets ( ) and operator precedence to specify the order in which their sub-parts are evaluated.
  - **`not`**
  - **`and`**
  - **`or`**

Logical Operators

p	q	p == q	p != q	p and q	p or q	not p
False	False	True	False	False	False	True
False	True	False	True	False	True	True
True	False	False	True	False	True	False
True	True	True	False	True	True	False

Examples

```
x, y, z = 5, 2, 0
print(x>9, y>0)              False True
print(x>9 and y>0)           False
print(x>9 or y>0)            True
print(not x>9)               True
print(x=5)                   Error
Print(x==5)                  True
```

Boolean Functions

- Many functions return a Boolean value.
- For example, bool(x) returns the Boolean value of a specified object. Cast.
- The object will always return True unless:
  - The object is empty, like [], (), {}
  - The object is False
  - The object is 0
  - The object is None

Boolean functions (String)

- `isinstance(item, dataType)`
- `str1.isdigit()`
- `str1.isalpha()`
- `str1.isalnum()`
- `str1.islower()`
- `str1.isupper()`
- `str1.isspace()`
- `str1.startwith(str2)`
- `str1.endwith(str2)`

Examples

```
str1 = "0123"
str2 = "Tier 1"
str3 = "UH"
print(str1.isdigit())    True
print(str1.isalpha())    False
print(str2.isupper())    False
print(str2.isalnum())    False
print(str3.isalpha())    True
print(str3.isalnum())    True
```

Example

```
if isinstance(x, str):
    print(x, type(x), id(x), len(x))
else:
    print(x, type(x), id(x))
```

Other Values as Boolean

- Following C/C++ tradition, Python treats
  - Number 0 (integer or float) as False
  - Any other number as True.
- For strings,
  - An empty string is False (length = 0)
  - Everything else is True
- If you treat a Boolean as a number,
  - True is 1
  - False is 0

Examples

```
print(bool(1))          True
print(bool(0))          False
print(bool(-1))         True
print(bool(99))         True
print(bool(9.99))       True
print(bool(0.0))        False
print(bool("UH"))       True
print(bool("0"))        True
print(bool(""))         False
```

De Morgan's Law

- not (cond1 and cond2) =

  not (cond1) or not (cond2)
- not (cond1 or cond2) =

  not (cond1) and not (cond2)

Chained Comparison

```
if (x >= 10) and (x <= 20):
    print (x, "is inside 10 and 20." )
else:
    print (x, "is outside 10 and 20.")
```

```
if (10 <= x <= 20):
    print (x, "is inside 10 and 20." )
else:
    print (x, "is outside 10 and 20.")
```

This is better

‘in’ as a Boolean Operator

- We will see ‘in’ as a keyword later in for-loops. They are different!
- Syntax: `<value> in <a collection of values>`
  - `5 in [1, 3, 5, 7, 9]`
  - `6 not in [1, 3, 5, 7, 9]`
- It is a membership test.
- For string, the membership is interpreted as a substring.
  - `‘love’ in ‘I love Python’`

2. IF statement

- **if expression:**
  statement(s)
  else:
  statement(s)

Optional

IF statement

- The if statement chooses between two alternatives based on a test expression. There are two versions of the if statement:

```
if expression:
        statement1(s)
```

```
if expression:
        statement1(s)
else:
        statement2(s)
```

IF Only

- In the first form, execution proceeds as follows.
  - First, the test is evaluated.
  - If the test evaluates to True, the statement(s) is executed, and execution proceeds to the next instruction.
  - If the test evaluates False, the execution skips the statement(s) and proceeds to the next instruction.

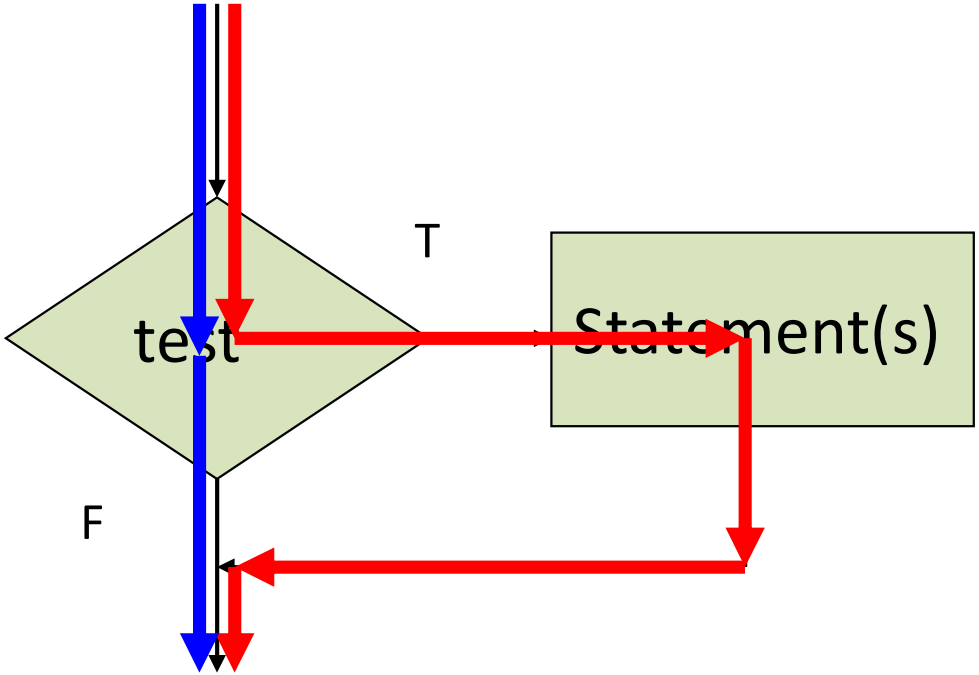
If-Else Statement

- For the second case,
  - First, the test is evaluated.
  - If the test evaluates to true, the statement1 is executed, and execution proceeds to the next instruction past the whole if-else, i.e., past statement2.
  - If the test evaluates to false, the execution skips the statement1, executes the statement2, and proceeds to the next instruction past the if-else.

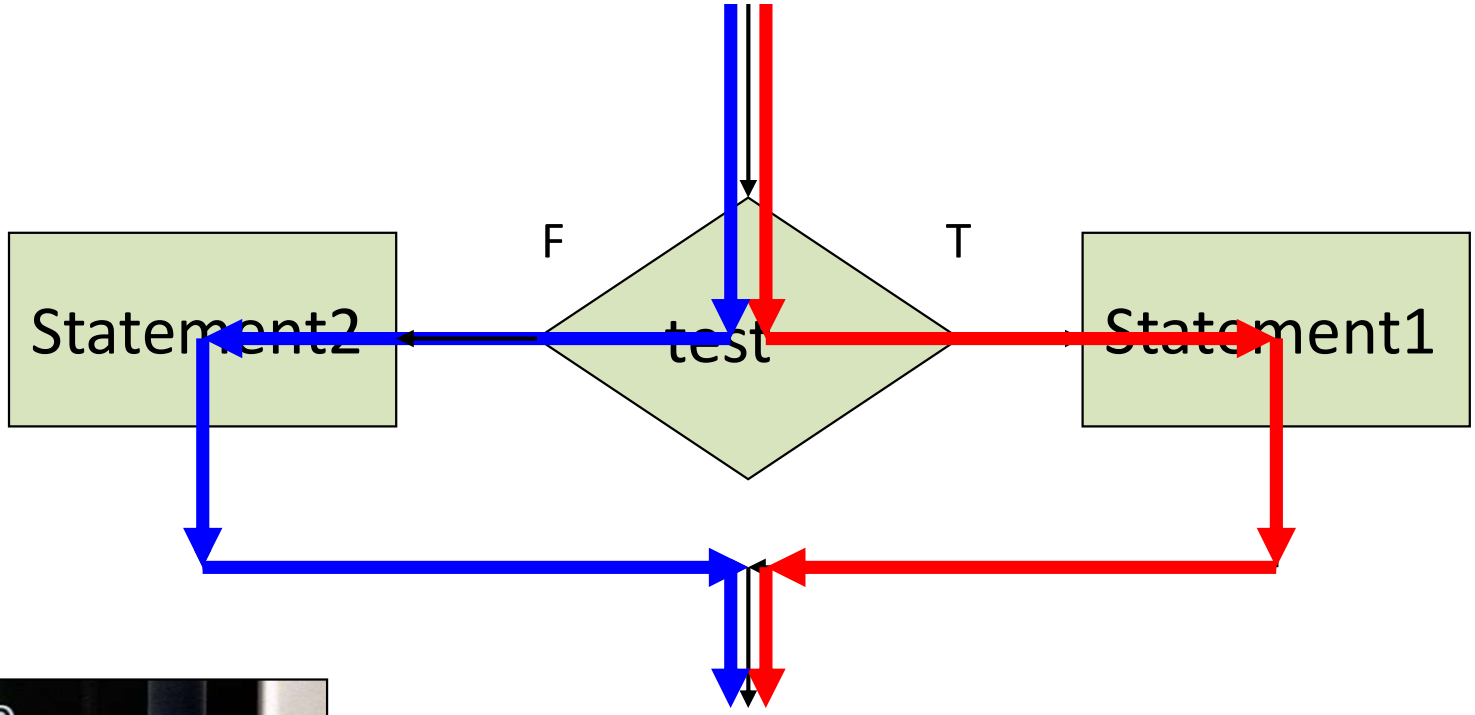
IF-ELSE statement

- Note that in both cases, execution proceeds to the next instruction after executing or skipping the statement(s).
- IF statement can be nested (IF inside IF)

IF



IF-ELSE


F
T
Statement2
test
Statement1


ONLY
EXCEPT
BUSES, TAXIS,
TRUCKS
& BIKES
LEFT LANE
BUS &
TAXI
ONLY
AT ALL
TIMES

ELIF (Else If)

```
if <expression>:
    <statement>
elif <expression>:
    <statement>
elif <expression>:
    <statement>
else:
    <statement>
```

Test Expression

- The test expression can be formed using relational operators.
- Especially note that the test for equality uses the symbol == and not =. The character = is used for the assignment operator. Thus, you should read == as "equal to" and read = as "assigned" or "set to."
- Using = in a test is one of the most common errors in writing programs.
- Fortunately, the IDE does catch this error.

Example

```
num1 = int(input("Enter the first number: "))
num2 = int(input("Enter the second number: "))

if num1 < num2:
    print(num1, "is smaller.")
elif num1 > num2:
    print(num2, "is smaller.")
else:
    print("The two numbers are equal.")
```

Nested IF

```
if (a<b):
    if (c<b):
        print("b is the max")
    else:
        print("b is the median")

if (a<b):
    if (c<b):
        print("b is the max")
else:
    print("b is the median")
```

Nested IF

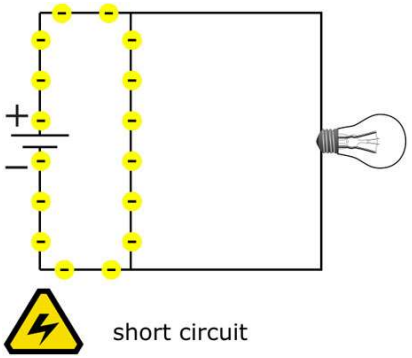
```
if (a<b):
    if (c<b):
        print("b is the max")
    else:
        print("b is the median")
else:
    pass

if (a<b):
    if (c<b):
        print("b is the max")
    else:
        pass
else:
    print("b is the median")
```

3. Short-Circuit Evaluation

- When Python is processing a logical expression, such as

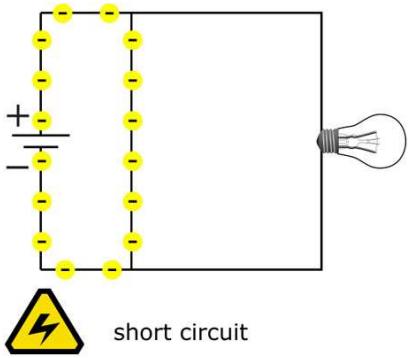
  Expr-1 and Expr-2

  it evaluates the expression from left to right:
  - Evaluate Expr-1 first and then
  - Evaluate Expr-2 if necessary.



Short-Circuit Evaluation

- If Expr-1 is False, the whole expression is False regardless of whether Expr-2 evaluates True or False.
- When Python detects that there is no need to evaluate the rest of a logical expression, it stops its evaluation and does not compute the rest of the expression.
- Saves computation time.



Short-Circuit

- Similarly, for

  Expr-1 or Expr-2

  There is no need to evaluate Expr-2 when Expr-1 has been evaluated to True.

- Caution: Sometimes, whether an expression is executed may have a side effect on the program's execution.

Examples

```
x, y = 6, 2
if x>=2 and x/y>2:
    print("1. true")
```

```
1. True
```

```
x, y = 1, 0
if x>=2 and x/y>2:
    print("2. true")
```

```
2. (none)
```

```
x, y = 6, 0
if x>=2 and x/y>2:
    print("3. true")
```

```
3. Error
```

Additional Remarks

- Side effects. A Boolean expression returns a value: the last evaluated value.
  - Short-circuit example
- There is a simpler way to do

```
if x**2>y:
    result = True
else:
    result = False
```

- Do this

```
result = x**2>y
```

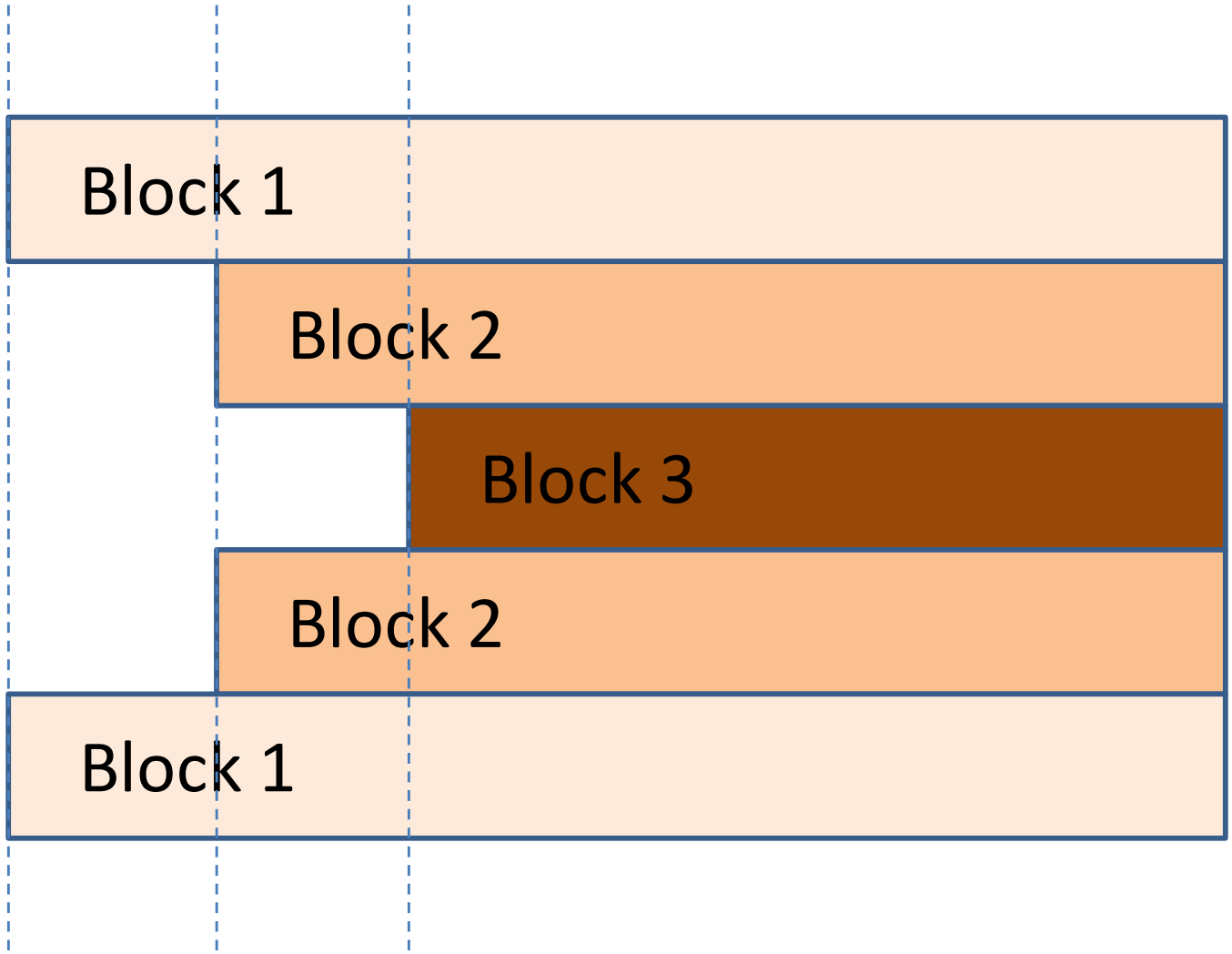
4. Indentation

- Leading whitespace at the beginning of a logical line is used to compute the line's indentation level, which in turn is used to determine the grouping of statements.
  - The total number of spaces preceding the first non-blank character determines the line's indentation.
  - Indentation cannot be split over multiple physical lines with backslashes; the whitespace up to the first backslash determines the indentation.

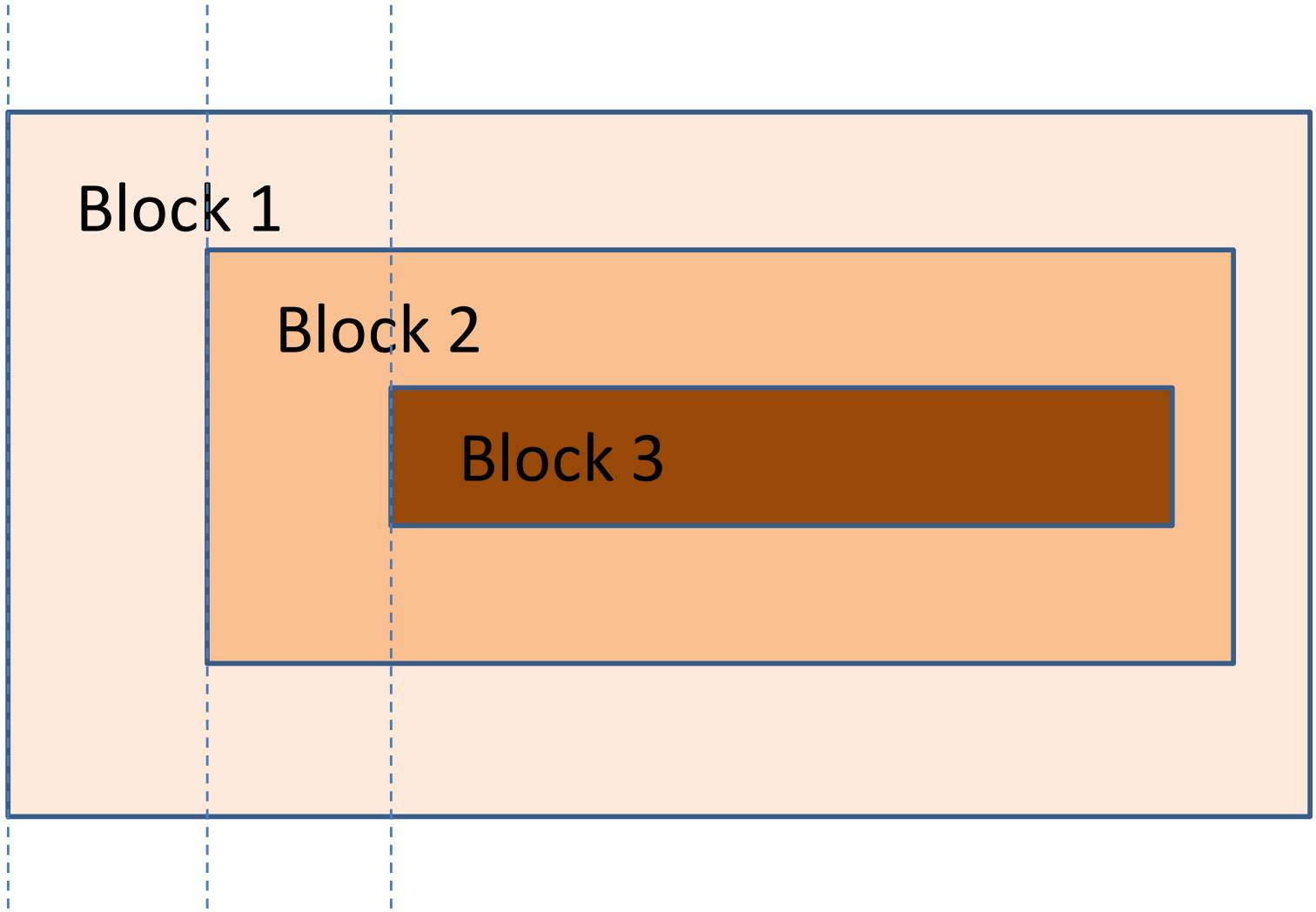
Indentation

- The indentation denotes python blocks; thus, indentation is uniform in Python programs.
- Indentation is meaningful to us as readers.

Indentation in Python



Indentation in Python



Indentation

- One of the most distinctive features of Python is its use of indentation to mark blocks of code.
- Indentation is a good practice but not necessary.
- The semicolon (;) is used as a “separator,” not as a “terminator.”
- To indicate a code block in Python, you must indent each block line by the same amount.

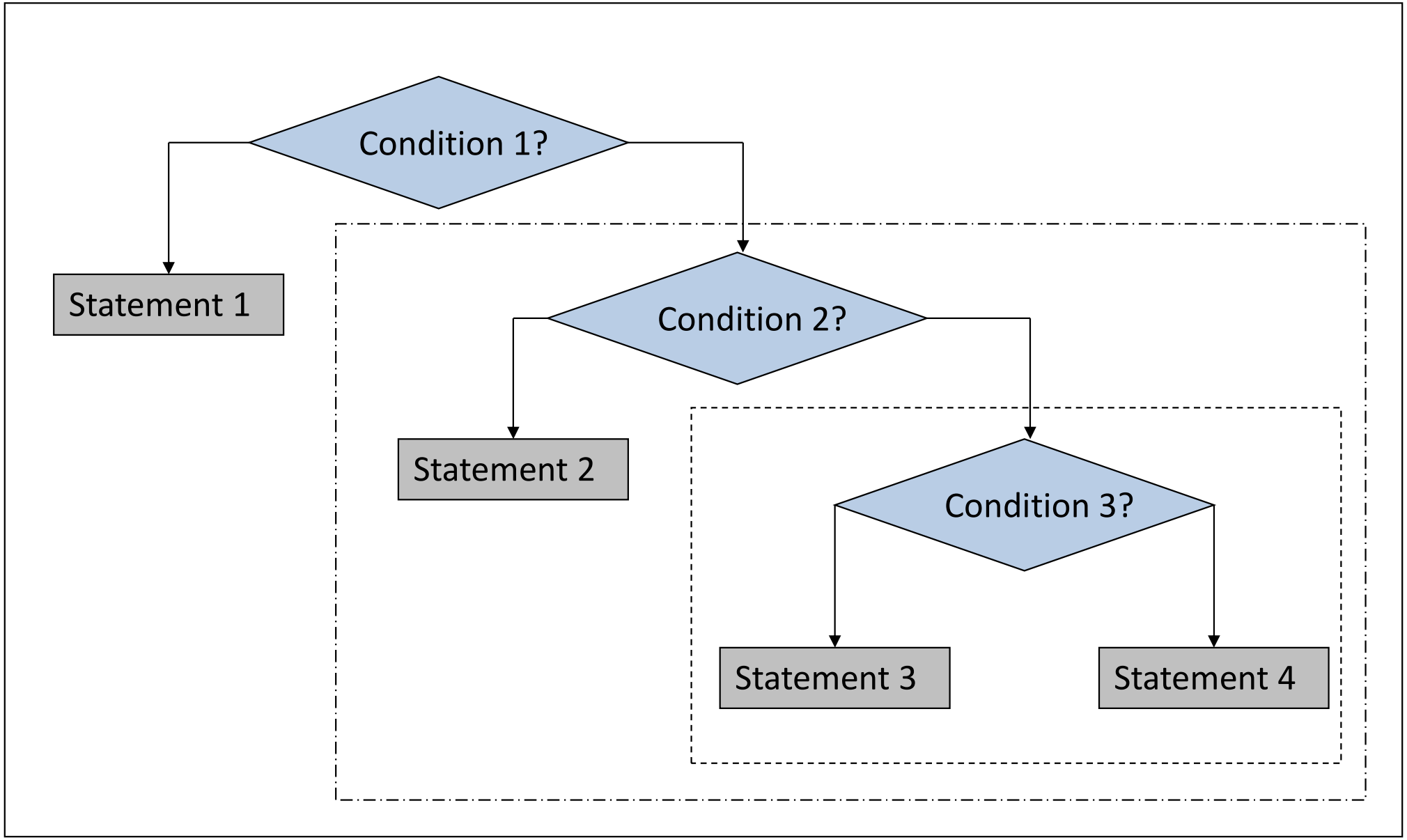
Don’t do this

```
status = int(input("Enter a number: "))

if status==1:
    print("Hello")
    print("world")
    print("!")

if status==1:
    print("Hello"); print("world"); print("!")

if status==1: print("Hello"); print("world"); print("!")
```

5. ELIF statement

- The IF statement can be nested. Any statement inside the if-block can be an if statement too.
- Each nested if has to be indented further.
- It is not practical to have more than 5 or 6 levels of indentation. Your program will be shifted to the right.
- "elif" can be viewed as a shorthand for "else if."

Nested if (cases)



Nested if in C/C++

```
if (test1)
    if (test2)
        statement1
    else
        statement2
```

```
if (test1)
    if (test2)
        statement1
else
    statement2
```

Indentation does not change the interpretation of the program.

The ELSE matches with the nearest unmatched IF

A Comparison

```
if status==1:
    print("One")
else:
    if status==2:
        print("Two")
    else:
        if status==3:
            print("Three")
print("That's all.")
```

```
if    status==1:
    print("One")
elif status==2:
    print("Two")
elif status==3:
    print("Three")
print("That's all.")
```

Example

```
status = int(input("Enter a number: "))
if status==1:
    print("One")
else:
    if status==2:
        print("Two")
    else:
        if status==3:
            print("Three")
        print("Back to 2.")
    print("Back to 1.")
print("That's all.")
```

6. Conditional Expression

- Python has a short-hand notation for if-statement that can be used directly within an expression.
- "Shorthand" form of if-else.
- The if-condition is typically very simple.
- You don't have to use this if you don't like it. But you should be able to understand it.

Conditional Expression

<expr1> `if` <conditional_expr> `else` <expr2>

- The conditional expression behaves like an expression syntactically. It can be used as part of a longer expression.
- It is also referred to as a conditional operator or ternary operator in various places in the Python documentation.
- In the following example, we can save a temporary variable if the result is used only once.

Equivalence

```
if food == "lamb":
    reply = "Yuck"
else:
    reply = "Yum"
```

```
reply = "Yuck" if food=="lamb" else "Yum"
```

Other Ways

<on_true> **`if`** <expression> **`else`** <on_false>

- (Boolean) Ternary operator.
- Compare to the ?: operator in other languages.

You can do this too

```
s = ('a' if x == 1 else
     'b' if x == 2 else
     'c' if x == 3 else
     'd'
    )
```

No \

No :

- This is cool!
- Probably easier to understand than nested if-else. After all, the statement's purpose is to give `s` a value.

Which one is better?

```
num = eval(input("enter a number: "))
```

```
if num>=0:
    abs = num
else:
    abs = -num
```

```
abs = num if num>=0 else -num
```

```
print ("absolute value: ", abs)
```

Which one is better?

```
age = eval(input("Enter age: "))
if age < 18:
    if age < 12:
        print("kid")
    else:
        print("teeager")
else:
    print("adult")
```

```
print("kid" if age<12 else
      "teenager" if age<18 else
      "adult")
```

Which one is better?

```
if boolExp == True:
```

```
if boolExp:
```


```
if boolExp == False:
```

```
if not boolExp:
```

```
if (a<x) and (x<b):
```

```
if (a<x<b):
```

Are they the same?

```
if num1>num2:
    xxx
if num2>num1:
    xxx
if num2==num1:
    xxx
```

```
if num1>num2:
    xxx
elif num2>num1:
    xxx
else:
    xxx
```

Which one is better?

```
if num1>num2:
    xxx
if num2>num1:
    xxx
if num2==num1:
    xxx
```

```
if num1>num2:
    xxx
elif num2>num1:
    xxx
else:
    xxx
```

Are they doing the same thing?