BIOS6642

Conditional Execution

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Boolean expressions**
- Unary and binary selection
- Nested and chained conditionals
- Conditional expression

Boolean expressions

- A Boolean expression is an expression that is either true or false.
- The simplest Boolean expressions in Python might be *True* and *False*.
- Note that they are not strings.

```
>>> True
True
>>> False
False
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Boolean expressions

- In Python, an expression comparing numeric expressions for equality or inequality is also a Boolean expression.
- Relational operators (e.g., <, ==, >, etc.) are used to compare two expressions.

```
>>> 2 == 2
True
>>> 2 == 3
False
>>> 2 < 3
True
```

Boolean expressions

- Relational operators

Expression	Meaning
x == y	True if $x = y$ (mathematical equality, not assignment); otherwise, false
x < y	True if $x < y$; otherwise, false
x <= y	True if $x \leq y$; otherwise, false
x > y	True if $x > y$; otherwise, false
x >= y	True if $x \geq y$; otherwise, false
x != y	True if $x \neq y$; otherwise, false

Boolean expressions

- Simple Boolean expressions, each involving one relational operator, can be combined into more complex Boolean expressions using the logical operators *and*, *or*, and *not*.
- The following expressions are true:

```
>>> 2 == 2 and 3 > 2
True
>>> 2 == 3 or 3 > 2
True
>>> not 2 > 3
True
```

Boolean expressions

- e_1 and e_2 are Boolean expressions:

e_1	e_2	e_1 and e_2	e_1 or e_2	not e_1
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Boolean expressions

- Precedence of some Python operators. Higher precedence operators appear above lower precedence operators.

Arity	Operators	Associativity
binary	**	
unary	+, -	
binary	*, /, //, %	left
binary	+, -	left
binary	>, <, >=, <=, ==, !=	left
unary	not	
binary	and	left
binary	or	left

Boolean expressions

- Consider the following program

```
#  All but 0, 1, 2, ..., 10
if value < 0 and value > 10:
    print(value)
```

Boolean expressions

- Consider the following program
  - Want to print OK if x is 1, 2, or 3

```
if x == 1 or 2 or 3:
    print("OK")
```

Boolean expressions

- The *and* operator first tests the expression to its left. If it finds the expression to be false, it does not bother to check the right expression. This approach is called short-circuit evaluation.
- The *or* operator also uses short-circuit evaluation.
- What if x is 0 for the following two expressions?

```
>>> (x != 0) and (y/x > 1)
```

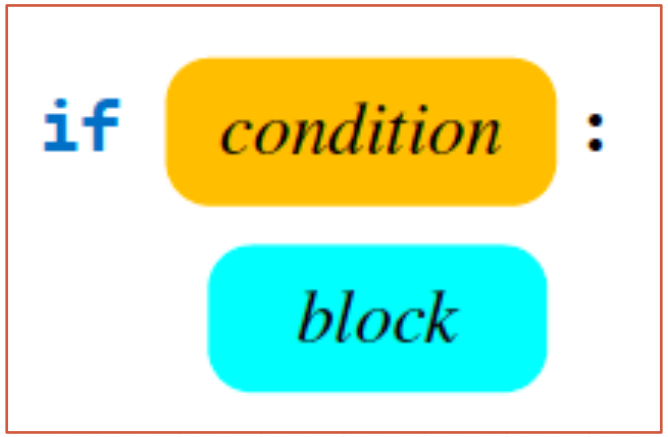
```
>>> (y/x > 1) and (x != 0)
```

Outline

- Boolean expressions
- **Unary and binary selection**
- Nested and chained conditionals
- Conditional expression

Unary and binary selections

- Unary selection (*if* statement)



- The reserved word *if* begins a *if* statement.
- The condition is a Boolean expression that determines whether or not the body will be executed. A colon must follow the condition.
- The block is a block of one or more statements to be executed if the condition is true. The statements within the block must all be **indented the same number of spaces from the left** (usually 4 spaces for indentation).

Unary and binary selections

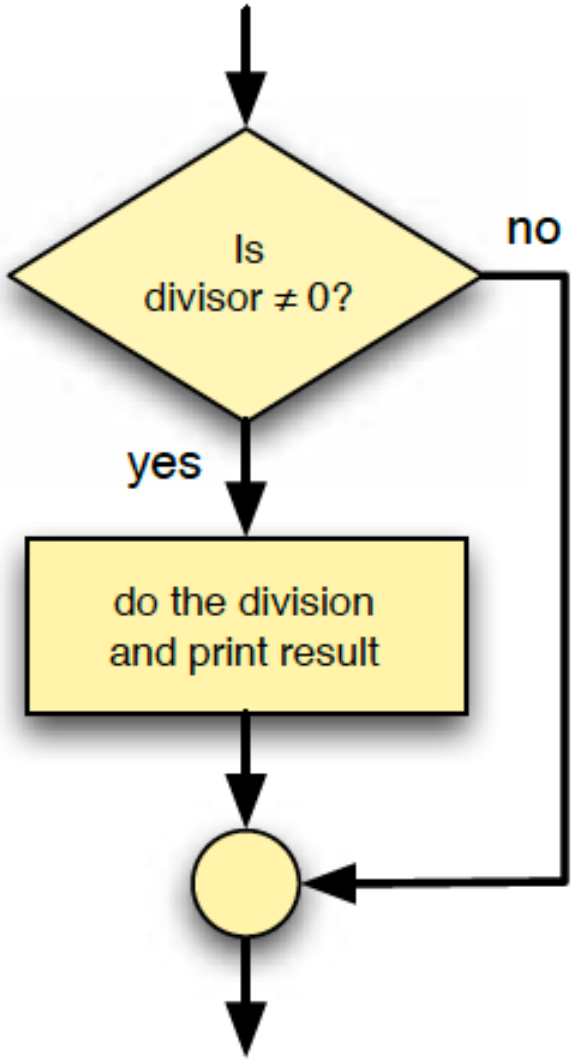
- Unary selection (*if* statement)

Codes

```
print('Please enter two numbers to divide.')
dividend = int(input('Please enter the first number to divide: '))
divisor = int(input('Please enter the second number to divide: '))
#  If possible, divide them and report the result
if divisor != 0:
    print(dividend, '/', divisor, "=", dividend/divisor)
```

Output

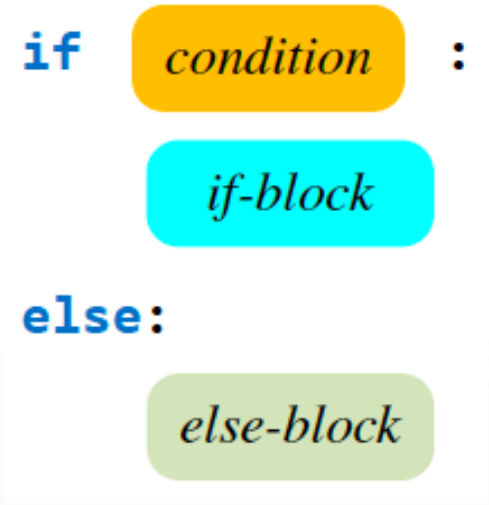
```
Please enter two numbers to divide.
Please enter the first number to divide: 32
Please enter the second number to divide: 8
32 / 8 = 4.0
```



Flowchart

Unary and binary selections

- Binary selection (*if-else* statement)



- The *else* block that is executed only if the Boolean condition is false.

Unary and binary selections

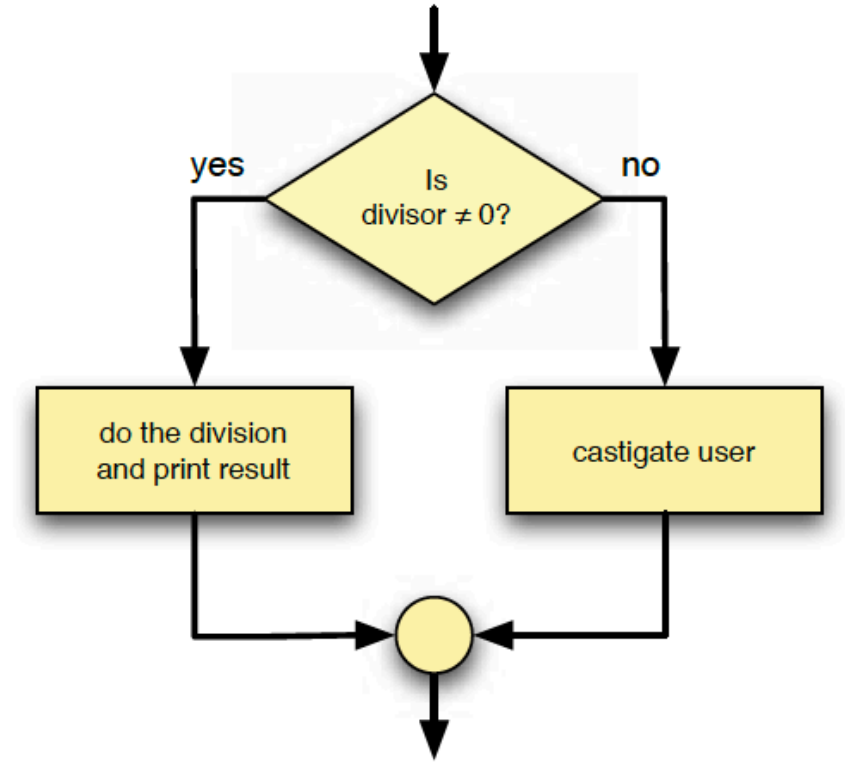
- Binary selection (*if-else* statement)

Codes

```
#  Get two integers from the user
dividend = int(input('Please enter the number to divide: '))
divisor = int(input('Please enter dividend: '))
#  If possible, divide them and report the result
if divisor != 0:
    print(dividend, '/', divisor, "=", dividend/divisor)
else:
    print('Division by zero is not allowed')
```

Running

```
Please enter the number to divide: 32
Please enter dividend: 0
Division by zero is not allowed
```



Flowchart

Unary and binary selections

- The *pass* statement: do nothing

```
if x < 0:
    pass  # Do nothing
else:
    print(x)
```

- The following is not legal in Python:

```
if x < 0:
    # Do nothing    (This will not work!)
else:
    print(x)
```

Unary and binary selections

- Floating-point equality

```
d1 = 1.11 - 1.10
d2 = 2.11 - 2.10
print('d1 =', d1, ' d2 =', d2)
if d1 == d2:
    print('Same')
else:
    print('Different')
```

- What will the program print?

Unary and binary selections

- Floating-point equality

```
d1 = 1.11 - 1.10
d2 = 2.11 - 2.10
print('d1 =', d1, ' d2 =', d2)
diff = d1 - d2          # Compute difference
if diff < 0:            # Compute absolute value
    diff = -diff
if diff < 0.0000001:  # Are the values close enough?
    print('Same')
else:
    print('Different')
```

- What will the program print?

Outline

- Boolean expressions
- Unary and binary selection
- **Nested and chained conditionals**
- Conditional expression

Nested and chained conditionals

- Nested conditionals

```
value = int(input("Please enter an integer value in the range 0...10: ")
if value >= 0:          # First check
    if value <= 10:     # Second check
        print(value, "is in range")
    else:
        print(value, "is too large")
else:
    print(value, "is too small")
print("Done")
```

Nested and chained conditionals

The codes will print one of eight messages depending on the user's input.

```
value = int(input("Please enter an integer in the range 0...5: "))
if value < 0:
    print("Too small")
else:
    if value == 0:
        print("zero")
    else:
        if value == 1:
            print("one")
        else:
            if value == 2:
                print("two")
            else:
                if value == 3:
                    print("three")
                else:
                    if value == 4:
                        print("four")
                    else:
                        if value == 5:
                            print("five")
                        else:
                            print("Too large")
print("Done")
```

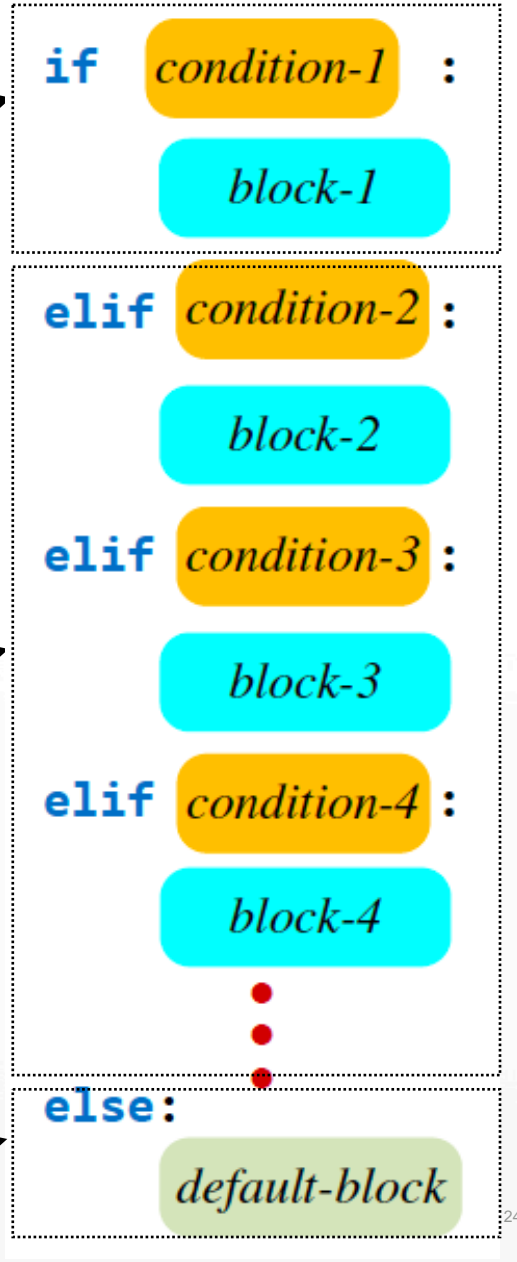
Nested and chained conditionals

Chained conditionals
(*if-elif-else* statement)

```
value = int(input("Please enter an integer in the range 0...5: "))
if value < 0:
    print("Too small")
elif value == 0:
    print("zero")
elif value == 1:
    print("one")
elif value == 2:
    print("two")
elif value == 3:
    print("three")
elif value == 4:
    print("four")
elif value == 5:
    print("five")
else:
    print("Too large")
print("Done")
```

Nested and chained conditionals

- Chained conditionals (*if-elif-else* statement)



Nested and chained conditionals

- Chained conditionals vs sequential conditionals

```
# Use a mult-way conditional statement
value = int(input())
if value == 0:
    print("zero")
elif value == 1:
    print("one")
elif value == 2:
    print("two")
elif value == 3:
    print("three")
elif value == 4:
    print("four")
elif value == 5:
    print("five")
print("Done")
```

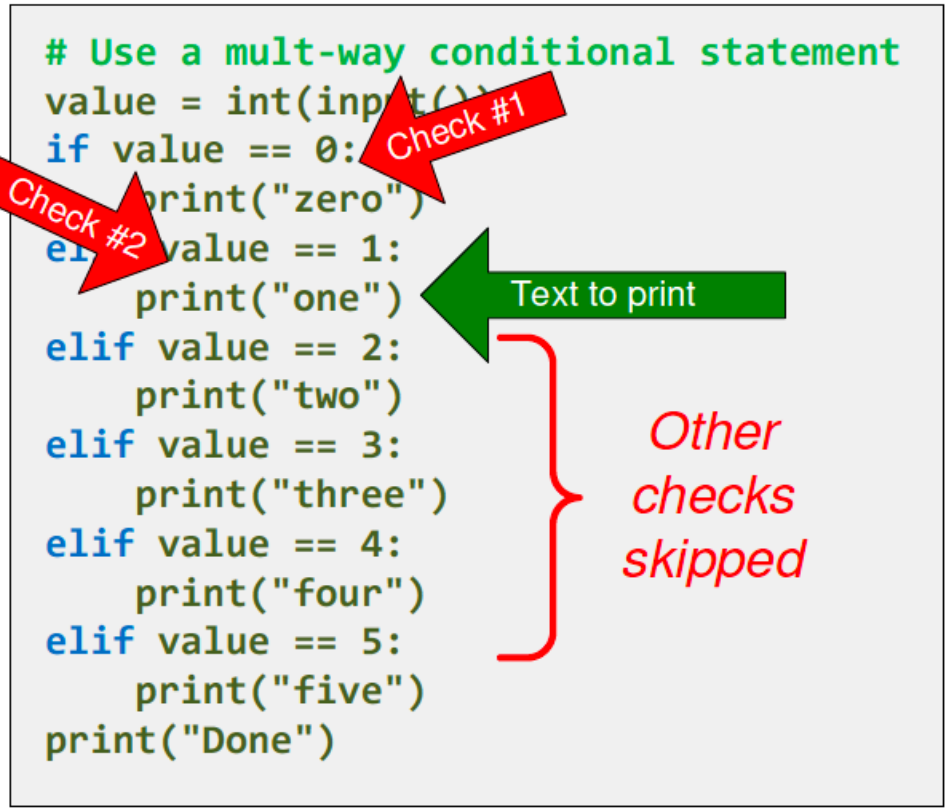
Chained conditionals

```
# Use sequential conditional statements
value = int(input())
if value == 0:
    print("zero")
if value == 1:
    print("one")
if value == 2:
    print("two")
if value == 3:
    print("three")
if value == 4:
    print("four")
if value == 5:
    print("five")
print("Done")
```

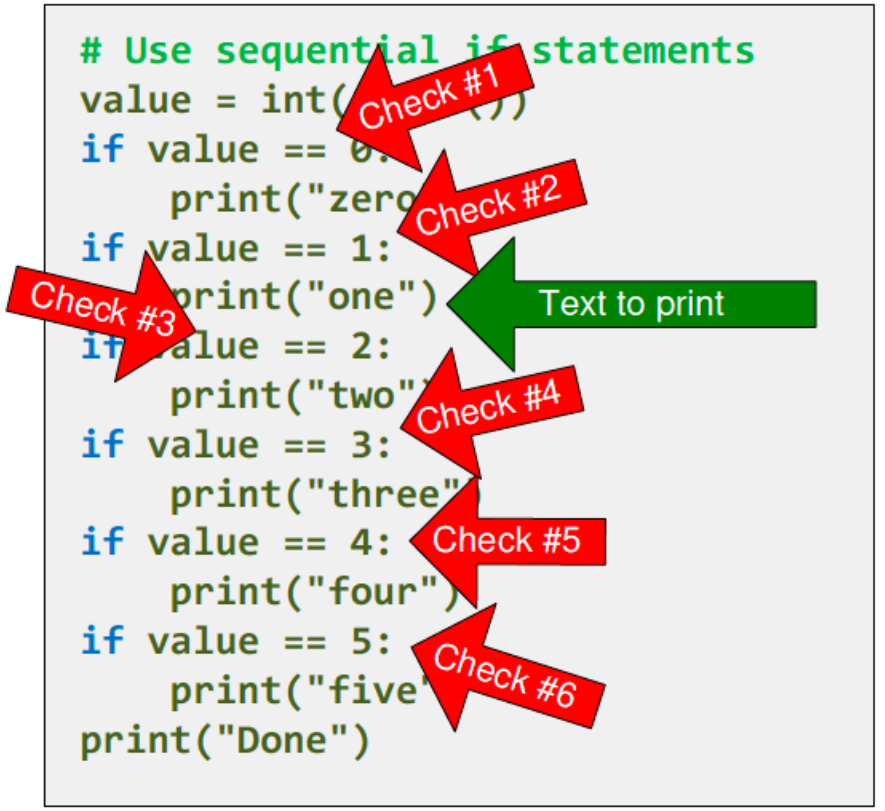
Sequential conditionals

Nested and chained conditionals

- Chained conditionals vs sequential conditionals




```
# Use a mult-way conditional statement
value = int(input())
if value == 0:
    print("zero")
elif value == 1:
    print("one")
elif value == 2:
    print("two")
elif value == 3:
    print("three")
elif value == 4:
    print("four")
elif value == 5:
    print("five")
print("Done")
```




```
# Use sequential if statements
value = int(input())
if value == 0:
    print("zero")
if value == 1:
    print("one")
if value == 2:
    print("two")
if value == 3:
    print("three")
if value == 4:
    print("four")
if value == 5:
    print("five")
print("Done")
```

Outline

- Boolean expressions
- Unary and binary selection
- Nested and chained conditionals
- **Conditional expression**

Conditional expression

- Conditional expression (ternary operator)

- *expression-1* is the overall value of the conditional expression if condition is true.
- *condition* is a normal Boolean expression that might appear in an *if* statement.
- *expression-2* is the overall value of the conditional expression if condition is false.

Conditional expression

- Conditional expression (ternary operator)

```
x = 2 if y > 0 else 3
```

- x equals 2 if y > 0, otherwise x equals 3

Conditional expression

- Conditional expression

```
#  Get the dividend and divisor from the user
dividend = int(input('Enter dividend: '))
divisor = int(input('Enter divisor: '))
#  We want to divide only if divisor is not zero; otherwise,
#  we will print an error message
if divisor != 0:
    print(dividend/divisor)
else:
    print('Error, cannot divide by zero')
```

```
#  Get the dividend and divisor from the user
dividend = int(input('Enter dividend: '))
divisor = int(input('Enter divisor: '))
#  We want to divide only if divisor is not zero; otherwise,
#  we will print an error message
msg = dividend/divisor if divisor != 0 else 'Error, cannot divide by zero'
print(msg)
```

Readings

- Think Python, chapters 5.1 – 5.7

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman