Programming Python

Conditional Statements

Prof. Chang-Chieh Cheng

Information Technology Service Center

National Chiao Tung University

Boolean Value

- **`False`**
  - Zero number
  - None
  - Empty
- **`True`**
  - Non-false value

Comparison Operators

- `>` Greater than
  - `x > y` is `True` if `x` is grater than `y`; otherwise, returns `False`
- `<` Less than
- `>=` Greater than or equal
- `<=` Less than or equal
- `==` Equal
- `!=` Not equal

```
x = 1
y = 2
print(x > y)    # False
print(x >= y)   # False
print(x < y)    # True
print(x <= y)   # True
print(x == y)   # False
print(x != y)   # True
```

Logical Operators

- and

x \ y	True	False
True	True	False
False	False	False

- or

x \ y	True	False
True	True	True
False	True	False

- not
  - not True ➔ False
  - not False ➔ True

```
x = 5
print(x > 0 and x < 10)          # True
print(x > 0 and x < 5)           # False
print(x > 0 or x < 5)            # True
print(x < 0 or x > 5)            # False
print(not(x < 0 or x > 5) )      # True
```

Logical Operators

- Logical complements
  - `not (x and y)` ⟷ `(not x) or (not y)`

```
x = 5
print(not(x > 0 and x < 10) )          # False
print(not(x > 0) or not(x < 10) )      # False
print(x <= 0 or x >= 10 )              # False
```

  - `not (x or y)` ⟷ `(not x) and (not y)`

```
x = 5
print(not(x < 0 or x > 10) )           # True
print(not(x < 0) and not(x > 10) )     # True
print(x >= 0 and x <= 10 )             # True
```

Membership Operators

- `in`
  - Let `L` be a data sequence such like a list
  - `x` **`is in`** `L` results `True` if `x` belongs to `L`
- `not in`
  - `x` **`not in`** `L` results `True` if `x` does not belong to `L`

```
L = [1, 2, 3, 4, 5]
print( 3 in L )                 # True
print( 6 in L )                 # False

print( 3 not in L )             # False
print( 6 not in L )             # True
```

Identity Operators

- is
  - Given two objects, `x` and `y`
  - `x is y` results `True` if `x` and `y` refer to the same object
- is not
  - `x is not y` results `True` if `x` and `y` refer to two different objects respectively

```
L1 = [1]
L2 = [1]
print( L1 is L1 )              # True
print( L1 is L2 )              # False
print( L1 is not L1 )          # False
print( L1 is not L2 )          # True
```

Identity Operators

- Notice that all small integers between -5 to 256, including -5 and 256, are default objects

```
x = 256
y = 0
z = x + y
print( x is x )          # True
print( x is y )          # False
print( x is z )          # True, why?

x = -5
y = 0
z = x + y
print( x is x )          # True
print( x is y )          # False
print( x is z )          # True

x = 257
y = 0
z = x + y
print( x is x )          # True
print( x is y )          # False
print( x is z )          # False
```

Do not use `is` to compare any two integers!

Identity Operators

- two string are the same object if they refer to the same literal text

```
s1 = 'Hello'
s2 = 'Hello'
print( s1 is s1 )       # True
print( s1 is s2 )       # True

s1 = input()
s2 = input()
print( s1 is s2 )       # False
```

Do not use `is` to compare any two strings!

`if` statement

- Syntax:

```
if condition:
    indented_statement_block
```

- For example, check a number is even

```
x = int(input('Input an integer:'))
if x % 2 == 0:
    print(x, 'is even')
```

`if-else` statement

- Syntax:

```
if condition:
    indented_statement_block
else:
    indented_statement_block
```

- For example, check a number is even or odd

```
x = int(input('Input a integer:'))
if x % 2 == 0:
    print(x, 'is even')
else:
    print(x, 'is odd')
```

if-elif-else statement

- Syntax:

```
if condition1:
    indented_statement_block1
elif condition2:
    indented_statement_block2
elif condition3:
    indented_statement_block3
...
else:
    indented_statement_blockE
```

  - Notice that else part must be the final part

- For example, classify a score

```
x = int(input('Input a score:'))
if x >= 90:
    print(x, 'is excellent!')
elif x >= 80:
    print(x, 'is good!')
elif 80 > x >= 60:
    print(x, 'is ok!')
else:
    print(x, 'is failed!')
```

Nested `if-elif-else` statement

```
x = int(input('Input a score:'))
if x >= 60:
    print('Pass!')
    if x >= 90:
        print(x, 'is excellent!')
    elif x >= 80:
        print(x, 'is good!')
    else:
        print(x, 'is ok!')
else:
    if x >= 50:
        print(x, 'still has a chance.')
    else:
        print(x, 'is failed!')
```

Conditional Expressions

- Syntax:

```
true_value if condition else false_value
```

- Example:

```
x = int(input('Input a number:'))
print('Pass!') if x >= 60 else print('Failed!')
```

```
x = int(input('Input a score:'))
x = 100 if x > 90 else x
```

Conditional Expressions

- Use conditional expression carefully!
- Example 1:

```
x = int(input('Input a score:'))
x += 10 if x > 90 else x
print(x)
```

```
x = int(input('Input a score:'))
x += (10 if x > 90 else 0)
print(x)
```

- Example 2:

```
s = input('Input a string:')
s = "NHWC" if s == "NWC" else "NCHW"
d = 1 if s == "NWC" else 2
print(d)
```

Exercises

- Finding the median from four numbers without any loop and calling sort().
- For example:
  - 30, 40, 5, 20
    - The median is 20
  - 2, 3, 2, 3
    - The median is 2
  - 3, 2, 3, 3
    - The median is 3

`try-except` statement

- Some errors could happen during a command execution.
- Such error is called an exception
- **`try`**`:`

  `code_block`

  - Try to catch exceptions generated from `code_block`
- **`except`** `exception_object:`

  `exception_procedure`

  - If `exception_object` is caught from `code_block` then execute `exception_procedure`

try-except statement

- Check whether an input data is a number

```
s = input('Input a number: ')
try:
    x = float(s)
    print(x, 'is a number.')
except ValueError:
    print(s, 'is not a number!')
```

This line won't be run if an error happened during the running of `x = float(s)`

- **`ValueError`** is a built-in exception object, it raised when a built-in operation or function receives an argument that has the right type but an inappropriate value.
  - `float(s)`
    - `s` is a string ➔ Type OK!
    - But if `s` represents a non-number text and cannot be converted to a float number ➔Error!

try-except statement

- **ZeroDivisionError**
  - Raised when the second argument of a division or modulo operation is zero.

```
try:
    x = int(input('Input x: '))
    print(x, 'is an integer.')
    y = int(input('Input y: '))
    print(y, 'is an integer.')
    z = x / y
    print(z)
except ValueError:
    print('x or y is not an integer!')
except ZeroDivisionError:
    print('x divided by zero!')
```

try-except statement

- Catch all exceptions

```
try:
    x = int(input('Input x: '))
    print(x, 'is an integer.')
    y = int(input('Input y: '))
    print(y, 'is an integer.')
    z = x / y
    print(z)
except:
    print('An error occurred!')
```

try-except statement

- **TypeError**
  - Raised when an operation or function is applied to an object of inappropriate type.

```
try:
    s = input('Input x: ')
    x = int(s)
except ValueError:
    x = s

try:
    s = input('Input y: ')
    y = int(s)
except ValueError:
    y = s

try:
    z = x + y
    print(z)
except TypeError:
    print('x and y are different types!')
```

Exercise

- Design a program that allows a user to input two data x and y
  - If x and y can regarded a two float numbers, sum their absolute values
  - If x and y are string, sum their of lengths
    - You can use `len(s)` to obtain the length of a string `s`.
  - If x is a number but y is a string, returns x.
  - If x is a string but y is a number, returns x + str(y).