ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Chapter 8:

Introduction to Python

Nguyen Thanh Binh

Content

- Introduction
- Data types
- Control flow
- Functions
- Input
- Libraries

Introduction

- Created in 1991 by Guido van Rossum
- Very-high-level programming language
- Supports a multitude of programming paradigms
  - object-oriented and procedural
- Large standard library includes numeric modules, networking modules, GUI support, development tools, AI, machine learning…
- Open Source
- Useful for a wide variety of applications
- Easy to learn
- Supports quick development

Installing Python

- Windows
  - Download Python from http://www.python.org
  - Install Python
- MacOS
  - Python is already installed
  - Open a terminal and run `python`
  - At the Python shell prompt type *import idlelib.idle*
- Linux
  - Python is already installed
  - run `python` from the terminal

Interpreter

- The standard implementation of Python is interpreted
- Two modes: normal and interactive
  - Interactive mode: code is written and then directly executed by the interpreter
  - Normal mode: files.py are provided to the interpreter

Interactive mode
$ python

```
>>> print('Hello')
Hello
>>> str = "Hello "
>>> str = str*2
>>> str
'Hello Hello '
>>> print(str)
Hello Hello
>>> 2*3+4
10
```

Normal mode

$ python hello.py
hello

Some basis

- Python uses indentation to denote code blocks instead of {} or ()

- Comments
  - Single-line comments denoted by #
  - Multi-line comments begin and end with triple quote (“””)
  - Triple quote with comment occurs as the first statement in a function is a docstring
    - Can be read by using: print(function_name.__doc__)

```
# here’s a comment
def myfunc():
        """here’s a
comment about
        the myfunc
function"""
        print("I'm in a
function!”)

print(myfunc.__doc__)
```

here’s a comment about the myfunc function

Some basic data types

- Logical type
  - bool (True/False)
- Numeric types
  - int, long, float and complex
- Sequence types
  - List
  - Set
  - String
  - Tuple

Numeric Types

- Numeric
  - **int**: equivalent to C's long int in 2.x but unlimited in 3.x.
  - **float**: equivalent to C's doubles.
  - **long**: unlimited in 2.x and unavailable in 3.x.
  - **complex**: complex numbers.

  - Many supported operations

```
$ python
>>> 3 + 2
5
>>> 18 % 5
3
>>> abs(-7)
7
>>> float(9)
9.0
>>> int(5.3)
5
>>> complex(1,2)
(1+2j)
>>> 2 ** 8
256
```

List

- Lists - *compound* data type
- Lists are mutable – it is possible to change their contents
- Lists can be nested

```
mylist = [135, 'apple', 'banana', 123]
print(mylist)
mylist[2] = 'orange'
print(mylist)
mylist[3] = [['test', 'test'], [1, 2]]
print(mylist)
len(mylist)
a = [1, 2]
b = [3, 4]
c = a + b
print(c)
c.sort(reverse=True)
print(c)
```

```
[135, 'apple', 'banana', 123]
[135, 'apple', 'orange', 123]
[135, 'apple', 'orange', [['test', 'test'], [1, 2]]]
4
[1, 2, 3, 4]
[4, 3, 2, 1]
```

Set

- Set – a collection of unsorted elements
- Set is mutable
- No duplicate elements

```
# Creating an empty set
s = set()
print(s)

# Creating a set with data
initialization
str = 'Hello Students'
s = set(str)
print(s)

# Creating a set with the use of a
list
s = set(["Hello", "Students",
"Hello", "Teacher"])
print(s)
```

```
set()
{'t', 'e', 'u', 's', 'n', 'o', 'H', 'l', 'S', ' ', 'd'}
{'Teacher', 'Hello', 'Students'}
```

String

- Created by simply enclosing characters in either single-quotes or double-quotes
- A segment of a string is called a slice
- There are many built-in string functions
- Strings are immutable

```
>>> s = 'Monty Python’
>>> len(s)
12
>>> print(s[0])
M
>>> s[0:5]
'Monty’
>>> s[6:]
'Python’
>>> s[0] = ‘J’
TypeError: 'str' object does not
support item assignment
```

Tuple

- A tuple is a sequence of values
- Tuples are immutable

```
>>> mytuple = ('coffee', 'tea')
>>> mytuple
('coffee', 'tea')
>>> mytuple.index("tea")
1
>>> mytuple[0]
'coffee'
>>> mytuple[0] = 'beer'
TypeError: 'tuple' object does not support
item assignment
```

Dictionary

- Dictionary represents a **mapping** from **keys** to **values**
- Duplicate keys are not allowed
- Duplicate values are fine
- Function **dict**() creates a dictionary
  - d = **dict**(name = "John", age = 36, country = "Norway")

```
majors = {
        'IT':"Information Technology",
        'CS':"Computer Science",
        'CE':"Computer Engineering",
        'EE':"Electronic Engineering",
        'CN':"Computer Networking"
}
print(majors)

for m in majors:
        print(m, majors[m])
```

```
{'IT': 'Information
Technology', 'CS': 'Computer
Science', 'CE': 'Computer
Engineering', 'EE':
'Electronic Engineering',
'CN': 'Computer Networking'}
IT Information Technology
CS Computer Science
CE Computer Engineering
EE Electronic Engineering
CN Computer Networking
```

Common sequence operations

All sequence data types support the following operations

Operation	Result
`x in s`	True if an item of s is equal to x, else False.
`x not in s`	False if an item of s is equal to x, else True.
`s + t`	The concatenation of s and t.
`s * n, n * s`	n shallow copies of s concatenated.
`s[i]`	ith item of s, origin 0.
`s[i:j]`	Slice of s from i to j.
`s[i:j:k]`	Slice of s from i to j with step k.
`len(s)`	Length of s.
`min(s)`	Smallest item of s.
`max(s)`	Largest item of s.
`s.index(x)`	Index of the first occurrence of x in s.
`s.count(x)`	Total number of occurrences of x in s.

Common sequence operations

Mutable sequence types further support the following operations

Operation	Result
`s[i] = x`	Item i of s is replaced by x.
`s[i:j] = t`	Slice of s from i to j is replaced by the contents of t.
`del s[i:j]`	Same as s[i:j] = [].
`s[i:j:k] = t`	The elements of s[i:j:k] are replaced by those of t.
`del s[i:j:k]`	Removes the elements of s[i:j:k] from the list.
`s.append(x)`	Add x to the end of s.

Common sequence operations

Mutable sequence types further support the following operations

`s.extend(x)`	Appends the contents of x to s.
`s.count(x)`	Return number of i's for which s[i] == x.
`s.index(x[, i[, j]])`	Return smallest k such that s[k] == x and i <= k < j.
`s.insert(i, x)`	Insert x at position i.
`s.pop([i])`	Same as x = s[i]; del s[i]; return x.
`s.remove(x)`	Same as del s[s.index(x)].
`s.reverse()`	Reverses the items of s in place.
`s.sort([cmp[, key[, reverse]]])`	Sort the items of s in place.

Control flow

Things that are False

- The boolean value False
- The numbers 0 (integer), 0.0 (float) and 0j (complex)
- The empty string ""
- The empty list [], empty dictionary {} and empty set set()

Things that are True

- The boolean value True
- All non-zero numbers
- Any string containing at least one character
- A non-empty data structure

Control flow

- Conditional statement has the following general form

```
if expression:
    statements
```

- If the boolean expression evaluates to True, the statements are executed. Otherwise, they are skipped entirely.

```
a = 1
b = 0
if a:
    print("a is true!“)
if not b:
    print("b is false!“)
if a and b:
    print("a and b are
true!“)
if a or b:
    print("a or b is
true!”)
```

```
a is true!
b is false!
a or b is true!
```

Control flow

- Conditional statement

```
if expression:
    statements
else:
    statements
```

- The elif keyword can be used to specify an else if statement
- Note: All the statements indented by the same amount after a programming construct are considered to be part of a single block of code.

```
a = 1
b = 0
c = 2
if a > b:
    if a > c:
        print("a is greatest")
    else:
        print("c is greatest")
elif b > c:
    print("b is greatest")
else:
    print("c is greatest")
```

c is greatest

Control flow

- While loops have the following general form

```
while
expression:
        statem
ents
```

- *statements* refers to one or more lines of code. The conditional expression may be any expression, where any non-zero value is true. The loop iterates while the expression is true.

```
i = 1
while i < 4:
    print(i)
    i = i + 1
flag = True
while flag and i < 8:
    print (flag, i)
    i = i + 1
```

```
1
2
3
True 4
True 5
True 6
True 7
```

Control flow

- The for loop has the following general form.

```
for var in sequence:
    statements
```

- If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *var*. Next, the statements are executed. Each item in the sequence is assigned to *var*, and the statements are executed until the entire sequence is exhausted.

```
for letter in
"aeiou":
    print("vowel: ",
letter)
for i in [1,2,3]:
    print(i)
for i in range(0,3):
    print(i)
```

```
vowel: a
vowel: e
vowel: i
vowel: o
vowel: u
1
2
3
0
1
2
```

Control flow

- Function **range**() for creating a range of integers, typically used in loop **for**

```
for i in
range(0,3,1):
    print (i)
for i in
range(0,8,2):
    print (i)
for i in
range(20,14,-2):
    print (i)
```

```
0
1
2
0
2
4
6
20
18
16
```

Control flow

- Four statements provided for manipulating loop structures: break, continue, pass, and else.
- **break**: terminates the current loop.
- **continue**: immediately begin the next iteration of the loop.
- **pass**: do nothing. Use when a statement is required syntactically.
- **else**: represents a set of statements that should execute when a loop terminates.

```
for num in range(10,20):
    if num%2 == 0:
        continue
    for i in range(3,num):
        if num%i == 0:
            break
    else:
        print (num, 'is a prime number')
```

```
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
```

Calculating Fibonacci numbers

f1 = 1, f2 = 1
fn = fn-1 + fn-2

```
f1, f2 = 1, 2
while f2 < 10:
        print(f1)
    f1, f2 = f2, f1 + f2
print(f1)
print(f2)
```

Python supports multiple assignment at once.
Right hand side is fully evaluated
before setting the variables.

```
1
2
3
5
8
13
```

Functions

- A function is created with the **def** keyword
- The statements in the block of the function must be indented

```
def function_name(args):
    statements
```

- The **return** keyword is used to specify a list of values to be returned
- All parameters in the Python language are passed by reference
- Only mutable objects can be changed in the called function

```
def func(str, list):
    print ("String: ", str, "\n")
    str = "New string"
    list[1] = 5
    return 1, 2

str = "Para"
mylist = [1, 2]
a,b = func(str, mylist)
print (str, mylist)
print (a, b)
```

String: Para

Para [1, 5]
1 2

Functions

- Iteraive function

```
def fibonacci(n):
        f1, f2 = 1, 1
        i = 2
        while i < n:
                i = i + 1
                f1, f2 = f2, f1 + f2
        return f2

for i in range(1,10):
        print(fibonacci(i))
```

- Recursive function

```
def fibonacci(n):
        if n == 1 or n == 2:
                return 1
        else:
                return (fibonacci(n-1) + fibonacci(n-2))

for i in range(1,10):
        print(fibonacci(i))
```

Functions

- What is the output of the following code?

```
def sum(n):
        even, odd = 0, 0

        for i in range(1,n):
                if i%2:
                        odd = odd + i
                else:
                        even = even + i
        return odd, even

print(sum(10))
```

Input

- input(arg)
  - Asking the user for a string of input and returning the string
  - If an argument is provided, it will be used as a prompt

- Note: input() returns a string
  - Use int() to convert string to int
  - Use eval(arg) to evaluate the argument

```
>>> name = input('What is your
name?\n')
What is your name?
Hung
>>> name
'Hung'
>>> n = input()
100
>>> n
‘100’
>>> n = int(n)
>>> n
100
```

Input

- Computing the real solutions of a quadratic equation

```
import math
def quadratic():
        #input under form: a, b, c, for example: 1, 4, 4
    a, b, c = eval(input("Enter the coefficients (a, b, c): "))

    delta = b * b - 4 * a * c
    if a != 0:
        if delta >= 0:
                root1 = (-b + math.sqrt(delta)) / (2 * a)
                root2 = (-b - math.sqrt(delta)) / (2 * a)
                print("The solutions are:", root1, root2)
        else:
                print("No solution.")
    else:
        print("This is not a quadratic equation.")

quadratic()
```

Libraries

- The Python Standard Library
  - Data types
  - Text processing
  - Numeric and mathematic modules
  - File and directory
  - Databases
  - Networking, Internet
  - Graphical User Interface
  - …
  - https://docs.python.org/3/library/

Libraries

- Some Python libraries for Machine Learning
  - Numpy: large multi-dimensional array and matrix processing
  - Scipy: Machine Learning algorithms
  - Scikit-learn: Machine Learning algorithms built on NumPy and SciPy
  - Theano: is used to define, evaluate and optimize mathematical expressions
  - TensorFlow: is used in deep learning research and application
  - Keras: deep learning framework
  - PyTorch: Machine Learning algorithms
  - Pandas: provides high-level data structures and tools for data analysis
  - Matplotlib: tools for data visualization