ISE 314X
Computer Programing for Engineers

Chapter 2
Writing Simple Programs

Yong Wang
Assistant Professor
Systems Science & Industrial Engineering
Binghamton University

Objectives

- To be able to understand and write Python statements to
  - input information from the keyboard
  - assign values to variables
  - output information to the screen
  - perform a counted loop

The Software Development Process

- Analyze the Problem
  - Figure out exactly the problem to be solved
  - Try to understand it as much as possible
- Determine Specifications
  - Describe exactly what your program will do
  - Describe the inputs, outputs, and how they relate to one another

The Software Development Process

- Create a Design
  - Formulate the overall structure of the program
  - You choose or develop your own algorithm that meets the specifications
- Implement the Design
  - Translate the design into code using a computer language

The Software Development Process

- Test/Debug the Program
  - Try out your program to see if it works
  - If there are any errors (*bugs*), they need to be located and fixed
  - This process is called *debugging*
  - Your goal is to find errors, so try everything that might "break" your program!

The Software Development Process

- Maintain the Program
  - Continue developing the program in response to the needs of your users
  - In the real world, most programs are never completely finished – they evolve over time

Example: Temperature Converter

- Analysis
  - The temperature is given in Celsius, the user wants it expressed in degrees Fahrenheit
- Specifications
  - Input: temperature in Celsius
  - Output: temperature in Fahrenheit
  - Relation: Output = (input)*9/5 + 32

Example: Temperature Converter

- Design
  - Input, Process, Output (IPO)
  - Prompt the user for input (Celsius temperature)
  - Convert it to Fahrenheit using F = (C)9/5 + 32
  - Output the result to the screen

Example: Temperature Converter

- In the design phase, we can generate some *pseudocode*, which is a rough draft of the program
- Pseudocode describes what a program does, step by step, in human language

Example: Temperature Converter

- Pseudocode
  - Input the temperature in degrees Celsius (call it C)
  - Calculate F as (9/5)*C+32
  - Output F

convert_temp.py

```
#convert_temp.py
#A program to convert Celsius temps to Fahrenheit

def main():
    celsius = eval(input("What is the Celsius temperature?"))
    fahrenheit = (9/5) * celsius + 32
    print("The temperature is",fahrenheit,"degrees Fahrenheit.")

main()
```

Example: Temperature Converter

- Run this program in IDLE multiple times:

```
What is the Celsius temperature? 0
The temperature is  32.0  degrees Fahrenheit.
What is the Celsius temperature? 100
The temperature is  212.0  degrees Fahrenheit.
What is the Celsius temperature? -40
The temperature is  -40.0  degrees Fahrenheit.
```

Elements of Programs

- Identifier
  - *Identifiers* are the names given to variables (celsius, fahrenheit), modules (main, convert_temp), etc.
  - Every identifier must begin with a letter or underscore _, followed by any sequence of letters, digits, or underscores

Examples

- **Are the following statements legal?**

```
>>> a = 3
>>> _a = 3
>>> a3 = 0
>>> 3a = 0
>>> +a = 0
>>> b@ = 1
```

Elements of Programs

– Identifiers are case sensitive

```
>>> a = 1
>>> A = 2
>>> a
1
>>> A
2
```

Elements of Programs

– **Which two of the following identifiers are the same?**
  - X
  - x
  - _x
  - Bread
  - bread
  - breAd

Elements of Programs

- Some identifiers are part of Python itself. These identifiers are known as *keywords*.
- Keywords are not available for you to use as an identifier for a variable
- Examples: and, del, for, while, with, is, raise, assert, elif, in, print, etc.
- For a complete list, see Table 2.1 in the textbook

Elements of Programs

- Keywords

```
>>> for = 3
File "<stdin>", line 1
    for = 3
        ^
SyntaxError: invalid syntax
>>> is = 3
File "<stdin>", line 1
    is = 3
     ^
SyntaxError: invalid syntax
```

Elements of Programs

- **Are For and iS python keywords?**
- **Are they legal python identifiers?**

```
>>> For = 3
>>> iS = 3
```

Elements of Programs

- Expressions
  - The fragments of code that produce or calculate new data values are called *expressions*
  - Simple expressions can be combined using *operators* such as +, -, *, /, **
  - Spaces are irrelevant within an expression

Elements of Programs

```
>>> x = 5
>>> x**2
25
>>> print(   x+  2)
7
```

Elements of Programs

```
>>> print(x2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x2' is not defined
>>>
```

- NameError is the error when you try to use a variable without a value assigned to it

Elements of Programs

- Output Statements
  - Each print statement will display on a separate line
  - A bare print will print a blank line

```
>>> print(3+4)
7
>>> print("The answer is", 3+4)
The answer is 7
>>> print()

>>>
```

Elements of Programs

- Output Statements
  - multiple expressions should be separated with commas

```
>>> print(3, 4)
3 4
>>> print(3 4)
  File "<stdin>", line 1
    print(3 4)
            ^
SyntaxError: invalid syntax
```

Assignment Statements

- Simple Assignment

```
<variable> = <expr>
```

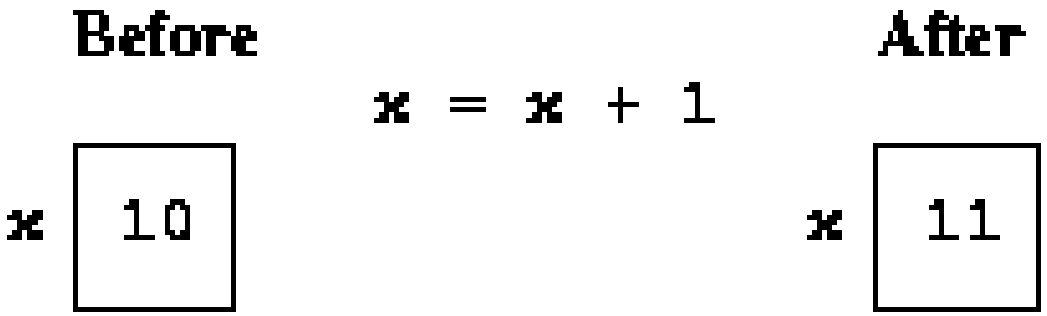
- variable is an identifier, expr is an expression
- The expression on the RHS is evaluated to produce a value which is then associated with the variable named on the LHS

Assignment Statements

```
>>> x = 0.5
>>> x = 3.9 * x * (1-x)
>>> x
0.975
>>> celsius = 0
>>> fahrenheit = 9/5 * celsius + 32
>>> fahrenheit
32.0
```

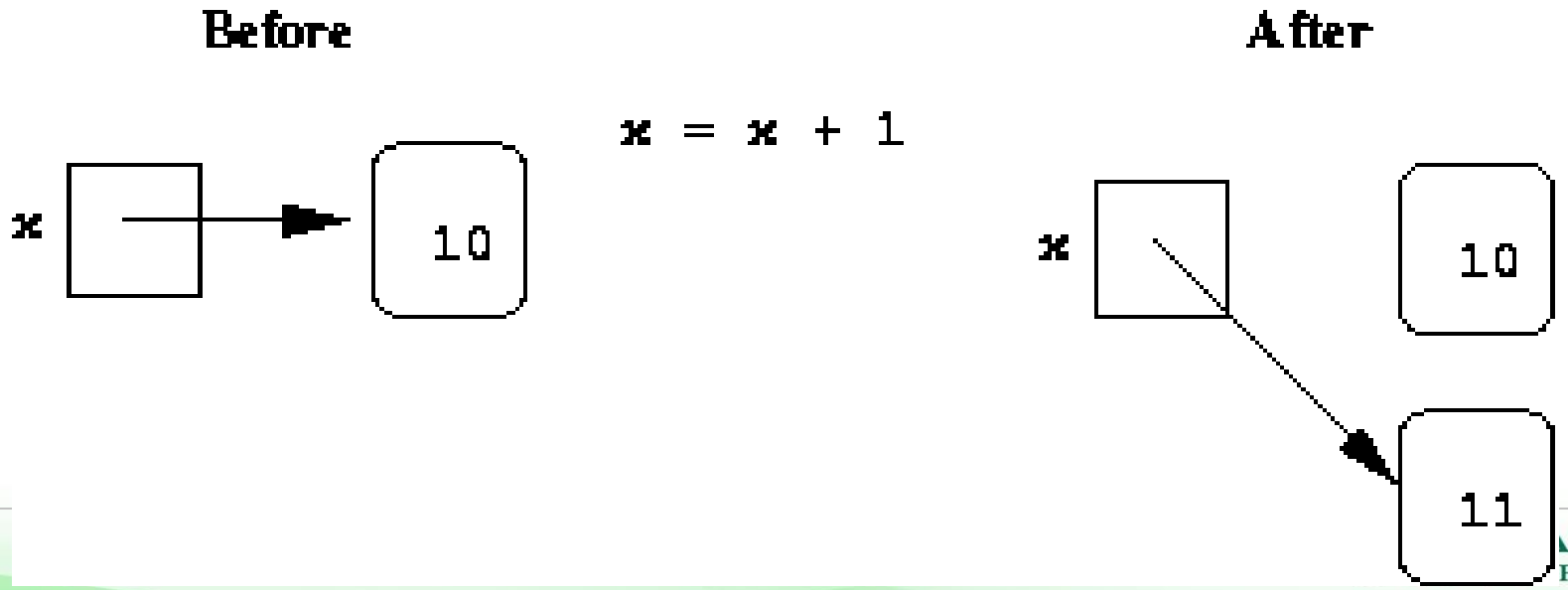
Assignment Statements

- Variables can be reassigned many times
- Logically, when a variable changes, the old value is erased and a new one is written in



Assignment Statements

- Physically, Python doesn't overwrite these memory locations
- Assigning a variable is more like putting a "sticky note" on a value and saying, "this is x"



Assigning Input

```
<variable> = eval(input(<prompt>))
```

- The purpose of an input statement is to get input from the user and store it into a variable
- First the prompt is printed. The `input` part waits for the user to enter a value and press <enter>

Assigning Input

```
<variable> = eval(input(<prompt>))
```

- The expression that was entered is `eval`uated to turn it from a string of characters into a number
- The number is assigned to the variable

Assigning Input

```
>>> age = eval(input("How old are you?"))
How old are you? 21
>>> age
21
```

Assigning Input

- Equivalent to the following two statements

```
>>> agestr = input("How old are you?")
How old are you? 21
>>> agestr
'21'
>>> age = eval(agestr)
>>> age
21
```

Simultaneous Assignment

- Several values can be calculated at the same time

```
<var>, <var>, … = <expr>, <expr>, …
```

- Evaluate the expressions in the RHS and assign them to the variables on the LHS

```
>>> x, y = 1, 2
>>> sum, diff = x+y, x-y
>>> sum
3
>>> diff
-1
```

Simultaneous Assignment

- Swap the values of two variables (the wrong way)

```
>>> x = 3
>>> y = 4
>>> print(x, y)
3 4
>>> x = y
>>> y = x
>>> print(x, y)
```

What is the output?

Simultaneous Assignment

- Swap the values of two variables (traditional way)

```
>>> x = 3
>>> y = 4
>>> print(x, y)
3 4
>>> temp = x
>>> x = y
>>> y = temp
>>> print(x, y)
4 3
```

Simultaneous Assignment

- Swap the values of two variables (python)

```
>>> x = 3
>>> y = 4
>>> print(x, y)
3 4
>>> x, y = y, x
>>> print(x, y)
4 3
```

Question

- Swap the values of three variables (python)

```
>>> x = 1
>>> y = 2
>>> z = 3
>>> print(x, y, z)
1 2 3
>>> ???
>>> print(x, y, z)
2 3 1
```

Simultaneous Assignment

```
# bread_eggs.py
def main():
    num1, num2 = eval(input("Enter # of slices of bread and # of eggs:"))
    print("You ordered", num1, "slices of bread and", num2, "eggs. Yum!")

main()
```

Simultaneous Assignment

- Run the program in IDLE:

```
Enter # of slices of bread and # of eggs: 3, 2
You ordered 3 slices of bread and 2 eggs. Yum!
```

Definite Loops

- A *definite loop* executes the body a definite number of times

```
for <var> in <sequence>:
    <body>
```

- The <var> is the *loop index*. It takes on each successive value in <sequence> in each iteration

Definite Loops

```
for <var> in <sequence>:
    <body>
```

- Do not forget :
- Do not forget the indentation

Definite Loops

```
>>> for odd in [1, 3, 5]:
...     print(odd*odd)
1
9
25
```

Definite Loops

```
>>> range(4)
range(0, 4)
>>> for i in range(4):
...     print(i)
0
1
2
3
```

- `range(n)` is a built-in Python function that generates a sequence of integer numbers from 0 to n-1
- The body of the loop executes n times

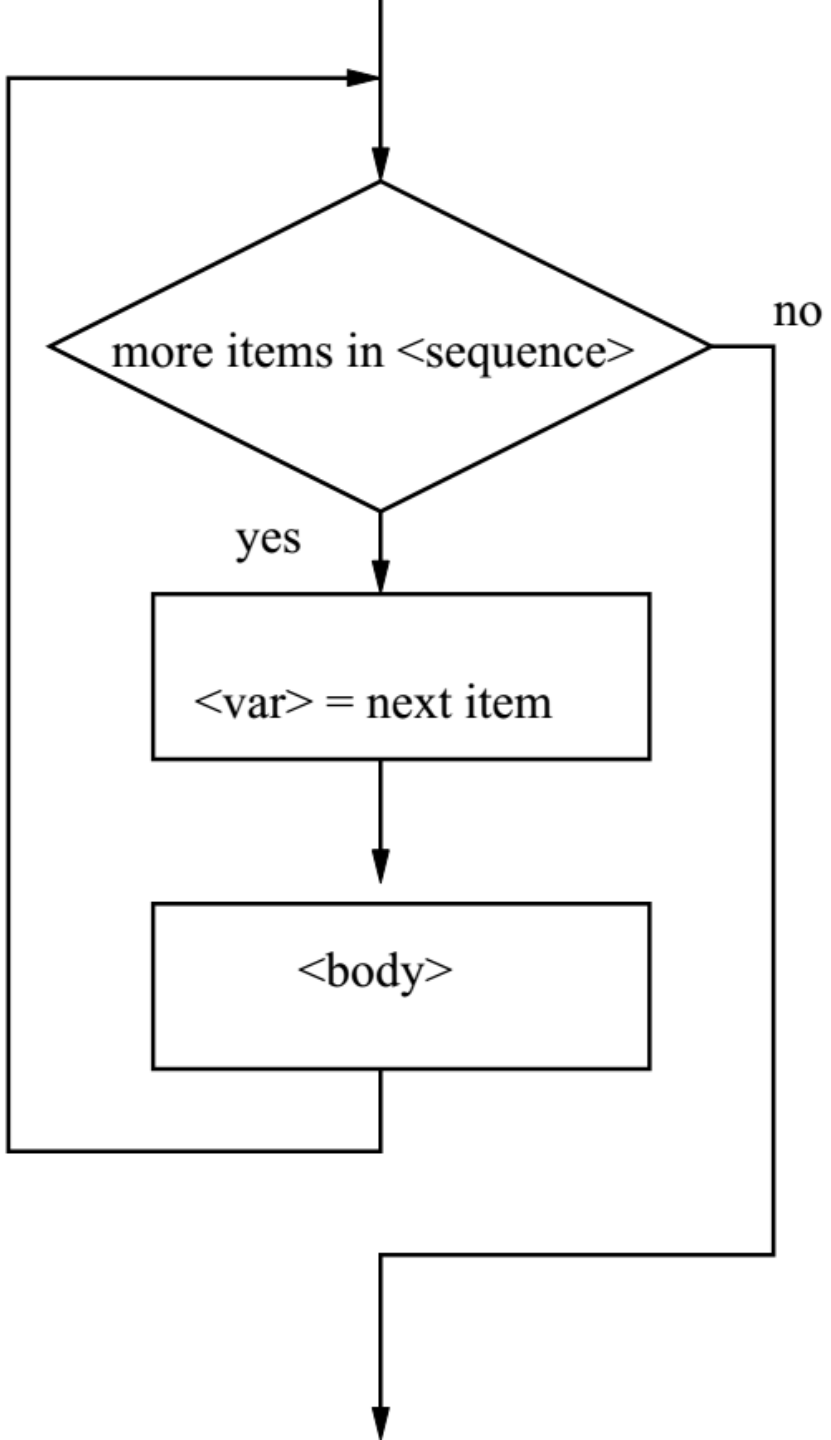
Definite Loops

```
>>> list(range(4))
[0, 1, 2, 3]
```

- `list` is a built-in Python function that turns the sequence into an explicit list

Definite Loops

- *for loops* alter the flow of program execution, so they are referred to as *control structures*



Example: Future Value

- Money deposited in a bank account earns interest
- How much will the account be worth 10 years from now?
- Inputs: principal, interest rate
- Output: value of the investment in 10 years

Example: Future Value

- Design
  - Print an introduction
  - Input the amount of the principal (principal)
  - Input the annual percentage rate (apr)
  - Repeat 10 times: principal = principal * (1 + apr)
  - Output the value of principal

Example Program: Future Value

```
# futval.py
def main():
    print("This program calculates the future")
    print("value of a 10-year investment.")
    principal = eval(input("Enter the initial principal: "))
    apr = eval(input("Enter the annual interest rate: "))
    for i in range(10):
        principal = principal * (1 + apr)
    print("The value in 10 years is:", principal)

main()
```

Example Program: Future Value

- Run the program in IDLE:

```
This program calculates the future
value of a 10-year investment.
Enter the initial principal: 100
Enter the annual interest rate: 0.10
The value in 10 years is: 259.37424601000026
```