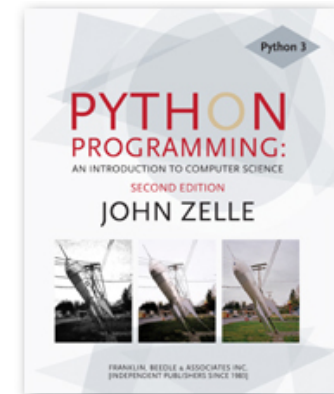


Python Programming: An Introduction to Computer Science



Chapter 2



Objectives

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



The Software Development Process

- The process of creating a program is often broken down into stages according to the information that is produced in each phase.



The Software Development Process

- **Analyze the Problem**

Figure out exactly the problem to be solved. Try to understand it as much as possible.



The Software Development Process

- **Determine Specifications**

Describe exactly what your program will do.

- Don't worry about *how* the program will work, but *what* it will do.
- Includes describing the inputs, outputs, and how they relate to one another.



The Software Development Process

- **Create a Design**

- Formulate the overall structure of the program.
- This is where the *how* of the program gets worked out.
- You choose or develop your own algorithm that meets the specifications.



The Software Development Process

- **Implement the Design**
 - Translate the design into a computer language.
 - In this course we will use Python.



The Software Development Process

■ **Test/Debug the Program**

- Try out your program to see if it worked.
- 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 Program: Temperature Converter

- Analysis – the temperature is given in Celsius, user wants it expressed in degrees Fahrenheit.
- Specification
 - Input – temperature in Celsius
 - Output – temperature in Fahrenheit
 - Output = $9/5(\text{input}) + 32$



Example Program: Temperature Converter

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



Example Program: Temperature Converter

- Before we start coding, let's write a rough draft of the program in *pseudocode*
- Pseudocode is precise English that describes what a program does, step by step.
- Using pseudocode, we can concentrate on the algorithm rather than the programming language.



Example Program: Temperature Converter

- Pseudocode:
 - Input the temperature in degrees Celsius (call it celsius)
 - Calculate fahrenheit as $(9/5)*\text{celsius}+32$
 - Output fahrenheit
- Now we need to convert this to Python!



Example Program: Temperature Converter

```
#convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

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

main()
```



Example Program: Temperature Converter

- Once we write a program, we should test it!

```
>>>
```

```
What is the Celsius temperature? 0
```

```
The temperature is 32.0 degrees Fahrenheit.
```

```
>>> main()
```

```
What is the Celsius temperature? 100
```

```
The temperature is 212.0 degrees Fahrenheit.
```

```
>>> main()
```

```
What is the Celsius temperature? -40
```

```
The temperature is -40.0 degrees Fahrenheit.
```

```
>>>
```



Elements of Programs

- Names

- Names are given to variables (celsius, fahrenheit), modules (main, convert), etc.
- These names are called *identifiers*
- Every identifier must begin with a letter or underscore (“_”), followed by any sequence of letters, digits, or underscores.
- Identifiers are case sensitive.



Elements of Programs

- These are all different, valid names
 - X
 - Celsius
 - Spam
 - spam
 - spAm
 - Spam_and_Eggs
 - Spam_And_Eggs



Elements of Programs

- Some identifiers are part of Python itself. These identifiers are known as *reserved words*. This means they are not available for you to use as a name for a variable, etc. in your program.
- and, del, for, is, raise, assert, elif, in, print, etc.
- For a complete list, see table 2.1



Elements of Programs

- Expressions
 - The fragments of code that produce or calculate new data values are called *expressions*.
 - *Literals* are used to represent a specific value, e.g. 3.9, 1, 1.0
 - Simple identifiers can also be expressions.



Elements of Programs

```
>>> x = 5
>>> x
5
>>> print(x)
5
>>> print(spam)
```

```
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in -toplevel-
    print spam
NameError: name 'spam' is not defined
>>>
```

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



Elements of Programs

- Simpler expressions can be combined using *operators*.
- $+$, $-$, $*$, $/$, $**$
- Spaces are irrelevant within an expression.
- The normal mathematical precedence applies.
- $((x1 - x2) / 2*n) + (spam / k**3)$



Elements of Programs

- Output Statements
 - A print statement can print any number of expressions.
 - Successive print statements will display on separate lines.
 - A bare print will print a blank line.



Elements of Programs

<code>print(3+4)</code>	7
<code>print(3, 4, 3+4)</code>	3 4 7
<code>print()</code>	
<code>print(3, 4, end=" ")</code> ,	3 4 7
<code>print(3 + 4)</code>	The answer is 7
<code>print("The answer is", 3+4)</code>	



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

- $x = 3.9 * x * (1-x)$
- $fahrenheit = 9/5 * celsius + 32$
- $x = 5$



Assignment Statements

- Variables can be reassigned as many times as you want!

```
>>> myVar = 0
```

```
>>> myVar
```

```
0
```

```
>>> myVar = 7
```

```
>>> myVar
```

```
7
```

```
>>> myVar = myVar + 1
```

```
>>> myVar
```

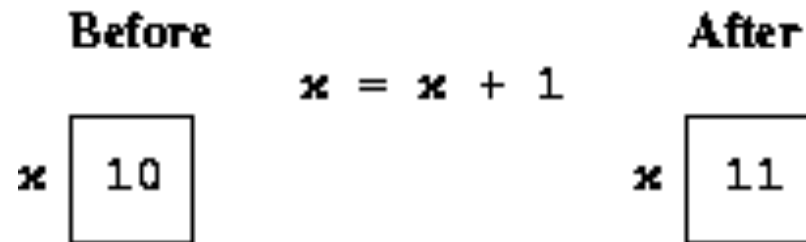
```
8
```

```
>>>
```



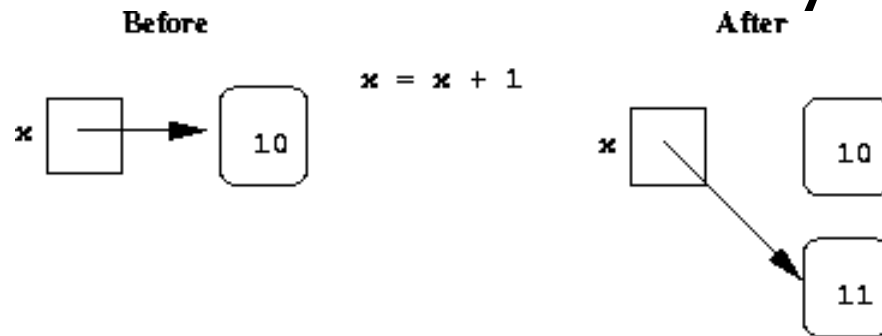
Assignment Statements

- Variables are like a box we can put values in.
- When a variable changes, the old value is erased and a new one is written in.



Assignment Statements

- Technically, this model of assignment is simplistic for Python.
- Python doesn't overwrite these memory locations (boxes).
- Assigning a variable is more like putting a “sticky note” on a value and saying, “this is x”.





Assigning Input

- The purpose of an input statement is to get input from the user and store it into a variable.
- `<variable> = eval(input(<prompt>))`



Assigning Input

- First the prompt is printed
- The `input` part waits for the user to enter a value and press `<enter>`
- The expression that was entered is `evaluated` to turn it from a string of characters into a Python value (a number).
- The value is assigned to the variable.



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



Simultaneous Assignment

- `sum, diff = x+y, x-y`
- How could you use this to swap the values for `x` and `y`?
 - Why doesn't this work?
`x = y`
`y = x`
- We could use a temporary variable...



Simultaneous Assignment

- We can swap the values of two variables quite easily in Python!

- $x, y = y, x$

```
>>> x = 3
```

```
>>> y = 4
```

```
>>> print x, y
```

```
3 4
```

```
>>> x, y = y, x
```

```
>>> print x, y
```

```
4 3
```



Simultaneous Assignment

- We can use this same idea to input multiple variables from a single input statement!
- Use commas to separate the inputs

```
def spamneggs():  
    spam, eggs = eval(input("Enter # of slices of spam followed by # of eggs: "))  
    print ("You ordered", eggs, "eggs and", spam, "slices of spam. Yum!")
```

```
>>> spamneggs()  
Enter the number of slices of spam followed by the number of eggs: 3, 2  
You ordered 2 eggs and 3 slices of spam. Yum!  
>>>
```



Definite Loops

- A *definite* loop executes a definite number of times, i.e., at the time Python starts the loop it knows exactly how many *iterations* to do.
- for <var> in <sequence>:
 <body>
- The beginning and end of the body are indicated by indentation.



Definite Loops

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

- The variable after the *for* is called the *loop index*. It takes on each successive value in *sequence*.



Definite Loops

```
>>> for i in [0,1,2,3]:  
    print (i)
```

```
0  
1  
2  
3
```

```
>>> for odd in [1, 3, 5, 7]:  
    print(odd*odd)
```

```
1  
9  
25  
49
```

```
>>>
```



Definite Loops

- In `chaos.py`, what did `range(10)` do?

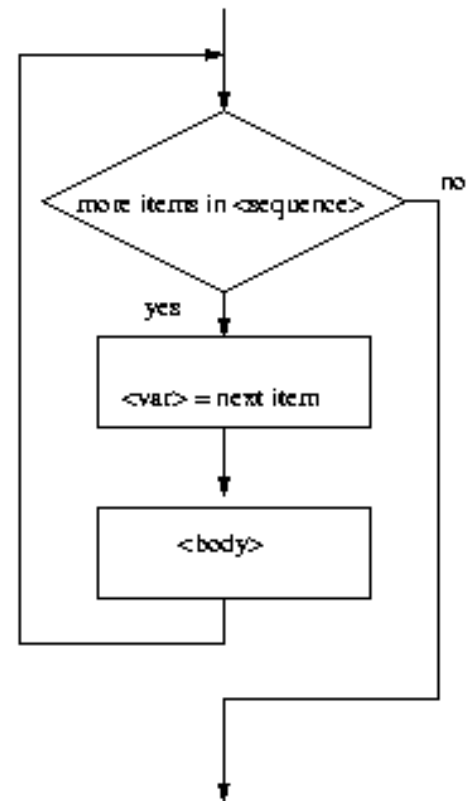
```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- `range` is a built-in Python function that generates a sequence of numbers, starting with 0.
- `list` is a built-in Python function that turns the sequence into an explicit list
- The body of the loop executes 10 times.



Definite Loops

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





Example Program: Future Value

- Analysis
 - 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 Program: Future Value

- Specification
 - User enters the initial amount to invest, the principal
 - User enters an annual percentage rate, the interest
 - The specifications can be represented like this ...



Example Program: Future Value

- **Program** Future Value
- **Inputs**
 - principal** The amount of money being invested, in dollars
 - apr** The annual percentage rate expressed as a decimal number.
- **Output** The value of the investment 10 years in the future
- **Relationship** Value after one year is given by $principal * (1 + apr)$. This needs to be done 10 times.



Example Program: 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

- Implementation
 - Each line translates to one line of Python (in this case)
 - Print an introduction

```
print ("This program calculates the future")  
print ("value of a 10-year investment.")
```
 - Input the amount of the principal

```
principal = eval(input("Enter the initial principal: "))
```



Example Program: Future Value

- Input the annual percentage rate
`apr = eval(input("Enter the annual interest rate: "))`
- Repeat 10 times:
`for i in range(10):`
- Calculate $\text{principal} = \text{principal} * (1 + \text{apr})$
`principal = principal * (1 + apr)`
- Output the value of the principal at the end of 10 years
`print ("The value in 10 years is:", principal)`



Example Program: Future Value

```
# futval.py
# A program to compute the value of an investment
# carried 10 years into the future

def main():
    print("This program calculates the future 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

```
>>> main()
```

This program calculates the future value of a 10-year investment.

Enter the initial principal: 100

Enter the annual interest rate: .03

The value in 10 years is: 134.391637934

```
>>> main()
```

This program calculates the future value of a 10-year investment.

Enter the initial principal: 100

Enter the annual interest rate: .10

The value in 10 years is: 259.37424601