

CMPE 102

Programming Logic and Design

Module 4

Input, Processing and Output

Input, Processing and Output

Concept:

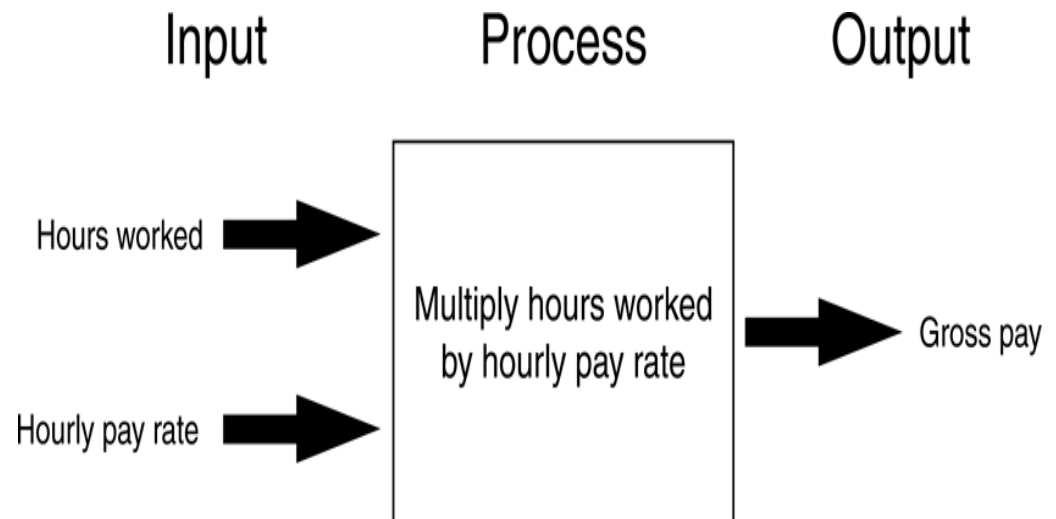
Input are data that the program receives. When a program receives data, it usually processes it by performing some operation with it. The result of the operation is sent out of the program as output.

Input, Processing and Output

Computer programs perform the following three steps:

1. Input is received
2. Some process is performed on the input
3. Output is produced

Figure 4-3 The input, processing, and output of the pay calculating program



Displaying the Output with `print` Statement

Concept:

You use the `print` statement to display output in a Python program.

Displaying the Output with **print** Statement

Program 4-1 (output.py)

```
1  print  'Kate Austen '  
2  print  '123 Dharma Lane '  
3  print  'Asheville, NC 28899 '
```

Displaying the Output with **print** Statement

Strings and String Literals

- A string is a sequence of characters that is used as data.
- When a string appears in the code of a program, it is called a string literal.
- In Python, single-quotes (') or double-quotes (") can be used to enclose string literals.

Displaying the Output with **print** Statement

Strings and String Literals

If a string literal contains either a single-quote or apostrophe, then enclose the string literal in double-quotes.

Program 4-3 (apostrophe.py)

```
1  print "Don't fear!"  
2  print "I'm here!"
```

Displaying the Output with `print` Statement

Strings and String Literals

If a string literal contains a double-quote, then enclose the string literal in single-quotes.

Program 4-4 (display_quote.py)

```
1 print 'Your assignment is to read "Hamlet" by tomorrow.'
```


Comments

Concept:

Comments are notes of explanations that document lines or sections of a program. Comments are part of the program, but the Python interpreter ignores them. They are intended for people who may be reading the source code.

Comments

Two types of comments:

1. Full line comment

This program calculates net pay

2. End-line comment

print "John Smith" # Display the name

Python begins a comment with the # character.

Variables

Concept:

A variable is a name that represents a value stored in the computer's memory.

Variables

Creating Variables with Assignment Statements

Assignment Statement

variable = expression

where,

variable

name of the variable

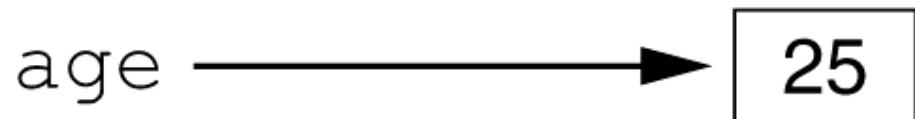
=

assignment operator

expression

value or piece of code that results in a value

Figure 4-4 The age variable references the value 25

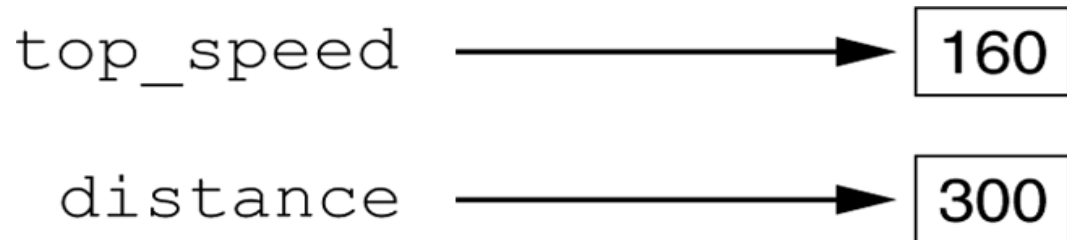


Variables

Program 4-5 (variable_demo2.py)

```
1 # Create two variables: top_speed and distance.
2 top_speed = 160
3 distance = 300
4
5 # Display the values referenced by the variables.
6 print 'The top speed is'
7 print top_speed
8 print 'The distance traveled is'
9 print distance
```

Figure 4-6 Two
variables



Variables

Variable Naming Rules

- Can not use Python key words
- Can not contain spaces
- First character must be one of the letters *a* through *z*, *A* through *Z*, or an underscore character(`_`)
- After the first character use letters *a* through *z*, *A* through *Z*, digits *0* through *9*, or underscore
- Uppercase and lowercase characters are distinct

Variables

Variable Naming Rules

For readability use one of the below styles for multiword variable name:

- Use the underscore character to represent a space

gross_pay

hot_dogs_sold_today

- **camelCase** naming convention

grossPay

hotDogsSoldToday

Variables

Variable Naming Rules

Table 4-1 Sample variable names

Variable Name	Legal or Illegal?
units_per_day	Legal
dayOfWeek	Legal
3dGraph	Illegal. Variable names cannot begin with a digit.
June1997	Legal
Mixture#3	Illegal. Variable names may only use letters, digits, or underscores.

Variables

Displaying Multiple Items with the `print` Statement

Use a comma to separate the items.

Program 4-9 (variable_demo3.py)

```
1  # This program demonstrates a variable.  
2  room = 503  
3  print 'I am staying in room number', room
```

Variables

Variable Reassignment

Variables are called “variables” because they can reference different values while a program is running.

Program 4-10 (variable_demo4.py)

```
1 # This program demonstrates variable reassignment.
2 # Assign a value to the dollars variable.
3 dollars = 2.75
4 print 'I have', dollars, 'in my account.'
5
6 # Reassign dollars so it references
7 # a different value.
8 dollars = 99.95
9 print 'But now I have', dollars, 'in my account!'
```

Figure 2-6 Variable reassignment in Program 2-10

The dollars variable after line 3 executes.

dollars → 2.75

The dollars variable after line 8 executes.

dollars → 2.75
dollars → 99.95

Variables

Numeric Data Types and Literals

A number that is written into a program's code is called a numeric literal.

- A numeric literal that is written as a whole number with no decimal point is considered an `int`.

For example:

7124, 503, and -9

- A numeric literal that is written with a decimal point is considered a `float`.

For example:

1.5, 3.1415, and 5.0

Variables

Storing Strings with the `str` Data Type

The `str` data type is used to store strings in memory.

Program 4-11 (string_variable.py)

```
1  # Create variables to reference two strings.
2  first_name = 'Kathryn'
3  last_name = 'Marino'
4
5  # Display the values referenced by the variables.
6  print first_name, last_name
```

Reading Input from the keyboard

Concept:

Programs commonly need to read input typed by the user on the keyboard. We will use the Python functions to do this.

Reading Input from the keyboard

Python uses built-in functions to read input from the keyboard.

A **function** is a piece of prewritten code that performs an operation and then returns a value back to the program.

The `input` function can be used to read numeric data from the keyboard.

Reading Input from the keyboard

Reading Numbers with the `input` Function

Use the `input` function in an assignment statement:

`variable = input (prompt)`

where,

<i><code>variable</code></i>	name of the variable that will reference the data
<i><code>=</code></i>	assignment operator
<i><code>input</code></i>	name of the function
<i><code>prompt</code></i>	string that is displayed on the screen

For example:

```
hours = input ('How many hours did you work?')
```

Performing Calculations

Concept:

Python has numerous operators that can be used to perform mathematical calculations.

Performing Calculations

A programmer's tools for performing calculations are *math operators*, and the following are provided by Python:

Table 2-2 Python math operators

Symbol	Operation	Description
+	Addition	Adds two numbers
—	Subtraction	Subtracts one number from another
*	Multiplication	Multiplies one number by another
/	Division	Divides one number by another and gives the quotient
%	Remainder	Divides one number by another and gives the remainder
**	Exponent	Raises a number to a power

Performing Calculations

For Example: A retail business is planning to have a storewide sale where the prices of all items will be 20 percent off. We have been asked to write a program to calculate the sale price of an item after the discount is subtracted.

Program 2-15 (sale_price.py)

```
1  # This program gets an item's original price and
2  # calculates its sale price, with a 20% discount.
3
4  # Get the item's original price.
5  original_price = input("Enter the item's original price: ")
6
7  # Calculate the amount of the discount.
8  discount = original_price * 0.2
9
10 # Calculate the sale price.
11 sale_price = original_price - discount
12
13 # Display the sale price.
14 print 'The sale price is', sale_price
```

Performing Calculations

Integer Division

When an integer is divided by an integer the result will also be an integer. This behavior is known as *integer division*.

For Example:

num_add = 17 + 5 # the answer will be 22

num_div_int = 3 / 2 # the answer is 1; 0.5 is the fractional part
 # and it is truncated.

num_div_real = 3.0 / 2.0 # the answer is 1.5

Performing Calculations

Operator Precedence

The precedence of the math operators, from highest to lowest, are:

1. Operations that are enclosed in parentheses.
2. Exponentiation **
3. Multiplication *, division /, and remainder %
4. Addition + and subtraction -

Table 2-3 Some expressions

Expression	Value
$5 + 2 * 4$	13
$10 / 2 - 3$	2
$8 + 12 * 2 - 4$	28
$6 - 3 * 2 + 7 - 1$	6

Performing Calculations

Grouping with Parentheses

Table 2-4 More expressions and their values

Expression	Value
$(5 + 2) * 4$	28
$10 / (5 - 3)$	5
$8 + 12 * (6 - 2)$	56
$(6 - 3) * (2 + 7) / 3$	9

Performing Calculations

The Exponent and Remainder Operators

****** is the exponent operator. Its purpose is to raise a number to a power.

For Example:

`area = length**2`

% is the remainder operator. Its purpose is to perform division and return the remainder.

For Example:

`leftover = 17 % 3`

`#remainder is 2`

Performing Calculations

Converting Math Formulas to Programming Statements

Algebraic Expression

$$x = \frac{a + b}{3}$$

Programming Expression

$$x = (a + b) / 3$$

Table 4-6 Algebraic and programming expressions

Algebraic Expression	Python Statement
$y = 3\frac{x}{2}$	<code>y = 3 * x / 2</code>
$z = 3bc + 4$	<code>z = 3 * b * c + 4</code>
$a = \frac{x + 2}{b - 1}$	<code>a = (x + 2) / (b - 1)</code>

Performing Calculations

Data Type Conversion

Python follows the following rules when evaluating mathematical expressions:

- When an operation is performed on two `int` values, the result will be an `int`.
- When an operation is performed on two `float` values, the result will be a `float`.
- When an operation is performed on an `int` and a `float`, the `int` value will be temporarily converted to a `float` and the result of the operation will be a `float`.

Performing Calculations

Data Type Conversion

Python built-in `float()` and `int()` functions.

For Example:

```
x = 27.9
```

```
y = int(x) # y will be assigned 27
```

```
x = -12.9
```

```
y = int(x) # y will be assigned -12
```

```
y = 7
```

```
x = float(y) # x will be assigned 7.0
```

Performing Calculations

Breaking Long Statements into Multiple Lines

Python allows for a break in a statement into multiple lines by using the **line continuation characters**, which is a backslash (\).

For Example:

```
print 'We sold', units_sold, \  
    'for a total of', sales_amount  
result = var1 * 2 + var2 * 3 + \  
    var3 * 4 + var4 * 5
```

More about data output

Suppressing the `print` Statement's Newline

The `print` statement displays a string and then prints an unseen newline character.

For Example:

```
print 'One'  
print 'Two'  
print 'Three'
```

Output:

```
One  
Two  
Three
```

If you do not want to start a new line of output, you can use a trailing comma.

For Example:

```
print 'One',  
print 'Two',  
print 'Three'
```

Output:

```
One Two Three
```

Suppressing print newline

- `print` function usually goes to next line when done
 - Special argument `end='delimiter'` causes `print` to place *delimiter* at end
 - Delimiter can be what you want, usually
 - Space : ' '
 - No space: ''

```
Print('One', end=' ')\nPrint('two', end=' ')\nPrint('three')
```

Output

One Two Three

Suppressing print Space

- `print` function uses space to separate items in print statement
 - Special argument `sep='delimiter'` causes `print` to use *delimiter* as item separator
 - Delimiter is usually no space in this case "", but can be anything

```
>>> print ('One','Two','Three', sep='')
```

Output:

OneTwoThree

Formatting Numbers

- **format** function: can format floating point numbers up to 12 significant digits
 - *format(numeric_value, format_specifier)*
 - Returns string containing formatted number
 - Format specifier typically includes precision and data type
 - Format: 'decimal_width(type)'
 - Ex: '2f' for 2 digits after floating point
 - Ex: '2e' for 2 digits in exponent format
 - Determine width of number: [field_width] decimal_width(type)
 - Comma separator: ',.2f' include commas in numbers >999
 - Numbers >= 5 are rounded up

```
>>> print(format(123.4567, '.2f'))
```

123.46

Formatting the Output

"format string" % (parameter, parameter, ...)

- *Placeholders* insert formatted values into a string:
 - %d an integer
 - %f a real number
 - %s a string
 - %8d an integer, 8 characters wide, right-aligned
 - %08d an integer, 8 characters wide, padding with 0s
 - %-8d an integer, 8 characters wide, left-aligned
 - %12f a real number, 12 characters wide
 - %.4f a real number, 4 characters after decimal
 - %6.2f a real number, 6 total characters wide, 2 after decimal

```
>>> x = 3; y = 3.14159; z = "hello"  
>>> print("%-8s, %04d is close to %.3f" % (z, x, y))  
hello , 0003 is close to 3.142
```

Formatting Field Width Example

```
>>> print('The number is', \
        format(12345.6789, '12,.2f'))
```

Output:

The number is 12,345.68

```
>>> print('The number is', \
        format(12345.6789, '12.2f'))
```

Output:

The number is 12345.68

Formatting Numbers Example

```
# This program demonstrates how a floating-point
# number can be displayed as currency.
# Notice the use of sep= so $ is next to number
monthly_pay = 5000.0
annual_pay = monthly_pay * 12
print('Your annual pay is $', \
      format(annual_pay, ',.2f'), sep='')
```

Output:

```
Your annual pay is $60,000.00
```

```
>>> print(format(12345.6789, 'e'))
1.2345678e+04
```

```
>>> print(format(12345.6789, '2e'))
1.23e+04
```

Formatting Field Width Example

```
num1 = 127.899
num2 = 3465.148
num3 = 3.776
num4 = 264.821
# Display each number in a field of 7 spaces
# with 2 decimal places.
print(format(num1, '7.2f'))
print(format(num2, '7.2f'))
print(format(num3, '7.2f'))
print(format(num4, '7.2f'))
```

Output:

```
127.90
3465.15
 3.78
264.82
```

Formatting Numbers as Percentage

- The % symbol can be used in the format string of `format` function to format number as percentage

```
>>> print(format(0.5, '%'))
```

```
50.000000%
```

```
>>> print(format(0.5, '0%'))
```

```
50%
```

Formatting Integers

- To format an integer using `format` function:
 - Use `d` as the type designator
 - Do not specify precision
 - Can still use `format` function to set field width or comma separator

```
>>> print(format(123456, 'd'))
```

```
123456
```

```
>>> print(format(123456, ',d'))
```

```
123,456
```

Magic Numbers

- A magic number is an unexplained numeric value that appears in a program's code.

Example:

```
amount = balance * 0.069
```

- What is the value 0.069? An interest rate? A fee percentage? Only the person who wrote the code knows for sure.

The Problems with Magic Numbers

- It can be difficult to determine the purpose of the number.
- It can take a lot of effort to change the number in multiple locations.
- Risk of making a mistake each time you type the magic number in the program's code.
 - For example, suppose you intend to type 0.069, but you accidentally type .0069. This mistake will cause mathematical errors that can be difficult to find.

Named Constants

- You should use named constants instead of magic numbers.
- A named constant is a name that represents a value that does not change during the program's execution.
- Example:

```
INTEREST_RATE = 0.069
```

- This creates a named constant named `INTEREST_RATE`, assigned the value 0.069.

```
amount = balance * INTEREST_RATE
```

Advantages of using Named Constants

- Named constants make code self-explanatory (self-documenting)
- Named constants make code easier to maintain
 - Change the value assigned to the constant, and the new value takes effect everywhere the constant is used
- Named constants help prevent typographical errors that are common when using magic numbers

More about data output

Escape Characters

A special character that is preceded with a backslash (\), appearing inside a string literal.

For Example:

```
print 'One\nTwo\nThree'
```

Table 2-7 Some of Python's escape characters

Escape Character	Effect
\n	Causes output to be advanced to the next line.
\t	Causes output to skip over to the next horizontal tab position.
\'	Causes a single quote mark to be printed.
\"	Causes a double quote mark to be printed.
\\	Causes a backslash character to be printed.

Output

One

Two

Three

More about data output

Displaying Multiple Items with the + Operator

When the + is used with two strings, it performs string concatenation.

For Example:

```
print 'this is' + 'one string.'    # one string is appended to another
```

Output:

This is one string

More about data output

Formatting Numbers

Floating-point Numbers:

- Without formatting the number can be displayed with up to 12 significant digits
- The % symbol is a **string format operator** when the operand on the left side of the % is a string
- A **formatting specifier** is a special set of characters that specify how a value should be formatted

For Example:

```
my_value = 7.23456
```

```
print 'The value is %.2f' % my_value
```

Output

```
The value is 7.23
```

More about data output

Formatting Numbers

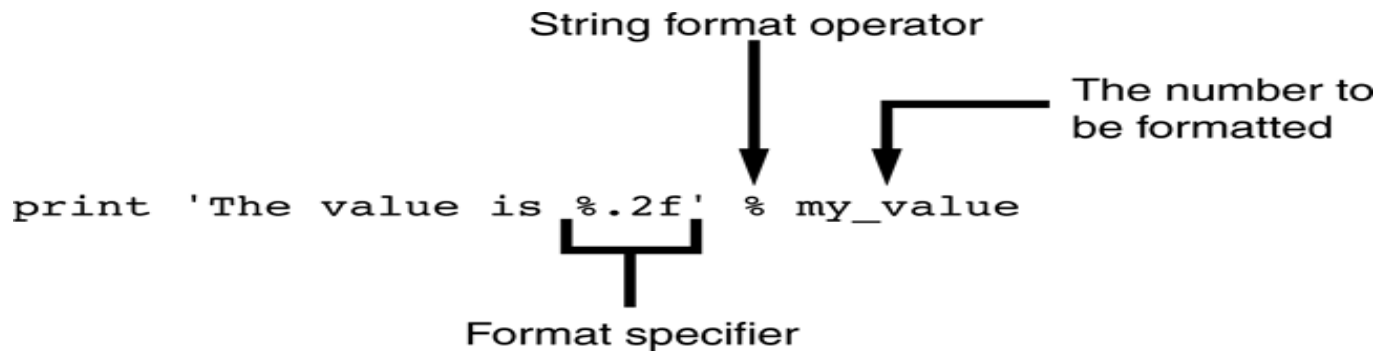
Figure 4-8 Using the string format operator

String format operator

The number to be formatted

```
print 'The value is %.2f' % my_value
```

Format specifier



Program 4-21 (formatting.py)

```
1  # This program demonstrates how a floating-point
2  # number can be formatted.
3  amount_due = 5000.0
4  monthly_payment = amount_due / 12.0
5  print 'The monthly payment is %.2f' % monthly_payment
```

More about data output

Formatting Numbers

Formatting Multiple Values

For Example:

value1 = 6.7891234

value2 = 1.2345678

print 'The values are %.1f and %.3f' % (value1, value2)

Output

The values are 6.8 and 1.235

More about data output

Formatting Numbers

Specify a Minimum Field Width

Program 4-23 (columns.py)

```
1  # This program displays the following
2  # floating-point numbers in a column
3  # with their decimal points aligned.
4  num1 = 127.899
5  num2 = 3465.148
6  num3 = 3.776
7  num4 = 264.821
8  num5 = 88.081
9  num6 = 799.999
10
11 # Display each number in a field of 7 spaces
12 # with 2 decimal places.
13 print '%7.2f' % num1
14 print '%7.2f' % num2
15 print '%7.2f' % num3
16 print '%7.2f' % num4
17 print '%7.2f' % num5
18 print '%7.2f' % num6
```

More about data output

Formatting Numbers

Formatting Integers and Strings

- The %d formatting specifier is used to format an integer
- The %s formatting specifier is used to format a string

For Example:

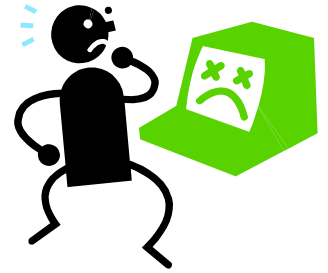
```
name = 'Ringo'
day = 'Monday'
sales = 8450.5543
print 'Hello %s. Good to see you!' % name
print 'The sales on %s were $%.2f.' % (day, sales)
```

Output

```
Hello Ringo. Good to see you!
The sales on Monday were $8450.55.
```


Common Programming Errors

Syntax Errors

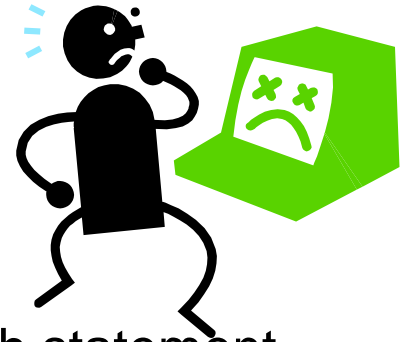


- Mis-spelling names of keywords
 - e.g., `printt()` instead of `print()`
- Forgetting to match closing quotes or brackets to opening quotes or brackets e.g., `print("hello)`
- Using variables before they've been named (allocated in memory).
- Program name: 15error_syntax.py

```
print(num)
num = 123
```

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


Runtime Errors



- Occur as a program is executing (running).
- The syntax of the language has *not* been violated (each statement follows the rules/syntax).
- During execution a serious error is encountered that causes the execution (running) of the program to cease.
- With a language like Python where translation occurs just before execution (interpreted) the timing of when runtime errors appear won't seem different from a syntax error.
- But for languages where translation occurs well before execution (compiled) the difference will be quite noticeable.
- A common example of a runtime error is a division by zero error

Runtime Error¹: An Example

- Program name: 16error_runtime.py

```
num2 = int(input("Type in a number: "))  
num3 = int(input("Type in a number: "))  
num1 = num2 / num3 # When zero is entered  
print(num1)
```

1 When 'num3' contains zero

```
[csc intro 39 ]> python3 error_runtime.py  
Type in a number: 1  
Type in a number: 2  
0.5
```

```
[csc intro 38 ]> python3 error_runtime.py  
Type in a number: 1  
Type in a number: 0  
Traceback (most recent call last):  
  File "error_runtime.py", line 3, in <module>  
    num1 = num2 / num3  
ZeroDivisionError: division by zero
```

Logic Errors

- The program has no *syntax errors*.
- The program runs from beginning to end with *no runtime errors*.
- But the logic of the program is incorrect (it doesn't do what it's supposed to and may produce an incorrect result).

- Program name: 17error_logic.py

```
print ("This program will calculate the area of a rectangle")
length = int(input("Enter the length: "))
width = int(input("Enter the width: "))
area = length + width
print("Area: ", area)
```

```
This program will calculate the area of a rectangle
Enter the length: 3
Enter the width: 4
Area: 7
```