Chapter 2: Input, Processing and Output
Topics

- Designing a Program
- Input, Processing, and Output
- Displaying Output with `print` Function
- Comments
- Variables
- Reading Input from the Keyboard
- Performing Calculations
- More About Data Output

Designing a Program

- **Programs must be designed before they are written**
- **Program development cycle:**
  - Design the program
  - Write the code
  - Correct syntax errors
  - Test the program
  - Correct logic errors

Designing a Program (cont'd.)

- **Design is the most important part of the program development cycle**
- **Understand the task that the program is to perform**
  - Work with customer to get a sense what the program is supposed to do
  - Ask questions about program details
  - Create one or more software requirements

Designing a Program (cont'd.)

- **Determine the steps that must be taken to perform the task**
  - Break down required task into a series of steps
  - Create an algorithm, listing logical steps that must be taken
- **<u>Algorithm</u>: set of well-defined logical steps that must be taken to perform a task**

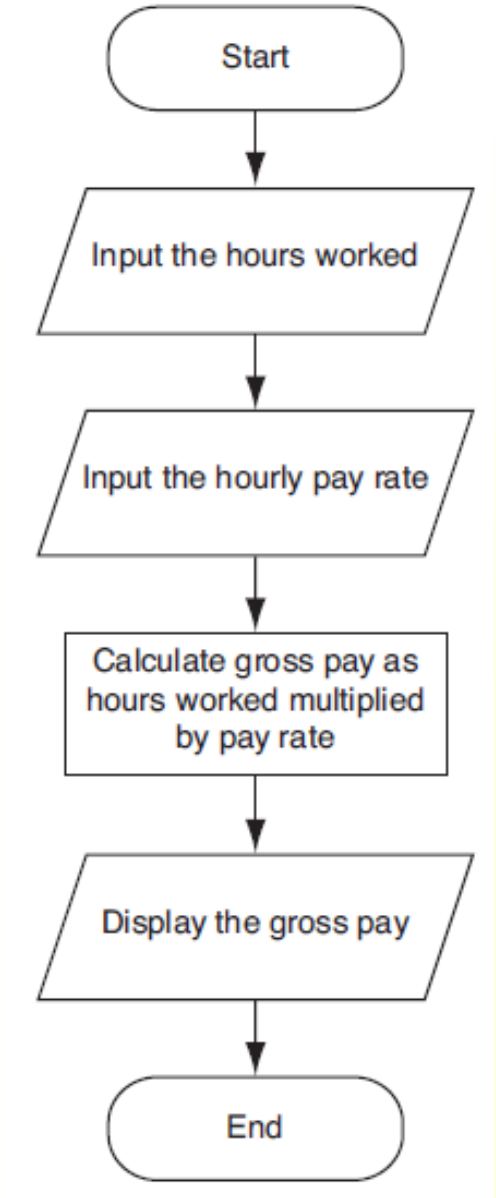
Pseudocode

- **<u>Pseudocode</u>: fake code**
  - Informal language that has no syntax rule
  - Not meant to be compiled or executed
  - Used to create model program
    - No need to worry about syntax errors, can focus on program's design
    - Can be translated directly into actual code in any programming language

Flowcharts

- **<u>Flowchart</u>: diagram that graphically depicts the steps in a program**
  - Ovals are terminal symbols
  - Parallelograms are input and output symbols
  - Rectangles are processing symbols
  - Symbols are connected by arrows that represent the flow of the program

Figure 2-2 Flowchart for the pay calculating program



Input, Processing, and Output

- **Typically, computer performs three-step process**
  - Receive input
    - Input: any data that the program receives while it is running
  - Perform some process on the input
    - Example: mathematical calculation
  - Produce output

Displaying Output with the `print` Function

- **Function: piece of prewritten code that performs an operation**
- **`print` function: displays output on the screen**
- **Argument: data given to a function**
  - Example: data that is printed to screen
- **Statements in a program execute in the order that they appear**
  - From top to bottom

Strings and String Literals

- **<u>String</u>: sequence of characters that is used as data**
- **<u>String literal</u>: string that appears in actual code of a program**
  - Must be enclosed in single (‘) or double (“) quote marks
  - String literal can be enclosed in triple quotes (''' or """)
    - Enclosed string can contain both single and double quotes and can have multiple lines

Comments

- **<u>Comments</u>: notes of explanation within a program**
  - Ignored by Python interpreter
    - Intended for a person reading the program's code
  - Begin with a # character
- **<u>End-line comment</u>: appears at the end of a line of code**
  - Typically explains the purpose of that line

Variables

- **<u>Variable</u>: name that represents a value stored in the computer memory**
  - Used to access and manipulate data stored in memory
  - A variable references the value it represents
- **<u>Assignment statement</u>: used to create a variable and make it reference data**
  - General format is `variable = expression`
    - Example: `age = 29`
    - <u>Assignment operator</u>: the equal sign (=)

Variables (cont'd.)

- **In assignment statement, variable receiving value must be on left side**
- **A variable can be passed as an argument to a function**
  - Variable name should not be enclosed in quote marks
- **You can only use a variable if a value is assigned to it**

Variable Naming Rules

- **Rules for naming variables in Python:**
  - Variable name cannot be a Python key word
  - Variable name cannot contain spaces
  - First character must be a letter or an underscore
  - After first character may use letters, digits, or underscores
  - Variable names are case sensitive
- **Variable name should reflect its use**

Displaying Multiple Items with the `print` Function

- **Python allows one to display multiple items with a single call to `print`**
  - Items are separated by commas when passed as arguments
  - Arguments displayed in the order they are passed to the function
  - Items are automatically separated by a space when displayed on screen

Variable Reassignment

- **Variables can reference different values while program is running**
- **<u>Garbage collection</u>: removal of values that are no longer referenced by variables**
  - Carried out by Python interpreter
- **A variable can refer to item of any type**
  - Variable that has been assigned to one type can be reassigned to another type

Numeric Data Types, Literals, and the `str` Data Type

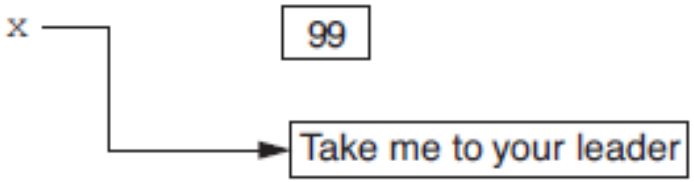
- **<u>Data types</u>: categorize value in memory**
  - e.g., int for integer, float for real number, str used for storing strings in memory
- **<u>Numeric literal</u>: number written in a program**
  - No decimal point considered int, otherwise, considered float
- **Some operations behave differently depending on data type**

Reassigning a Variable to a Different Type

- A variable in Python can refer to items of any type

Figure 2-7 The variable x references an integer

Figure 2-8 The variable x references a string



Reading Input from the Keyboard

- **Most programs need to read input from the user**
- **Built-in `input` function reads input from keyboard**
  - Returns the data as a string
  - Format: `variable = input(prompt)`
    - `prompt` is typically a string instructing user to enter a value
  - Does not automatically display a space after the prompt

Reading Numbers with the `input` Function

- **`input` function always returns a string**
- **Built-in functions convert between data types**
  - `int(`*`item`*`)` converts *`item`* to an `int`
  - `float(`*`item`*`)` converts *`item`* to a `float`
  - Nested function call: general format:
    `function1``(`*`function2`*`(`*`argument`*`))`
    - value returned by function2 is passed to function1
  - Type conversion only works if item is valid numeric value, otherwise, throws exception

Performing Calculations

- **Math expression: performs calculation and gives a value**
  - Math operator: tool for performing calculation
  - Operands: values surrounding operator
    - Variables can be used as operands
  - Resulting value typically assigned to variable
- **Two types of division:**
  - / operator performs floating point division
  - // operator performs integer division
    - Positive results truncated, negative rounded away from zero

Operator Precedence and Grouping with Parentheses

- **Python operator precedence:**
  1. Operations enclosed in parentheses
     - Forces operations to be performed before others
  2. Exponentiation (**)
  3. Multiplication (*), division (/ and //), and remainder (%)
  4. Addition (+) and subtraction (-)
- **Higher precedence performed first**
  - Same precedence operators execute from left to right

The Exponent Operator and the Remainder Operator

- **<u>Exponent operator (`**`)</u>: Raises a number to a power**
  - `x ** y` = x^y
- **Remainder operator (`%`): Performs division and returns the remainder**
  - a.k.a. modulus operator
  - e.g., `4%2=0, 5%2=1`
  - Typically used to convert times and distances, and to detect odd or even numbers

Converting Math Formulas to Programming Statements

- **Operator required for any mathematical operation**
- **When converting mathematical expression to programming statement:**
  - May need to add multiplication operators
  - May need to insert parentheses

Mixed-Type Expressions and Data Type Conversion

- **Data type resulting from math operation depends on data types of operands**
  - Two `int` values: result is an `int`
  - Two `float` values: result is a `float`
  - `int` and `float`: `int` temporarily converted to `float`, result of the operation is a `float`
    - Mixed-type expression
  - Type conversion of `float` to `int` causes truncation of fractional part

Breaking Long Statements into Multiple Lines

- **Long statements cannot be viewed on screen without scrolling and cannot be printed without cutting off**
- **<u>Multiline continuation character (\)</u>: Allows to break a statement into multiple lines**
  - Example:

```
print('my first name is',\
first_name)
```

More About Data Output

- **`print` function displays line of output**
  - Newline character at end of printed data
  - Special argument `end='`*`delimiter`*`'` causes `print` to place *`delimiter`* at end of data instead of newline character
- **`print` function uses space as item separator**
  - Special argument `sep='`*`delimiter`*`'` causes `print` to use *`delimiter`* as item separator

More About Data Output (cont'd.)

- **Special characters appearing in string literal**
  - Preceded by backslash (`\`)
    - Examples: newline (`\n`), horizontal tab (`\t`)
  - Treated as commands embedded in string
- **When + operator used on two strings in performs string concatenation**
  - Useful for breaking up a long string literal

Formatting Numbers

- **Can format display of numbers on screen using built-in `format` function**
  - Two arguments:
    - Numeric value to be formatted
    - Format specifier
  - Returns string containing formatted number
  - Format specifier typically includes precision and data type
    - Can be used to indicate scientific notation, comma separators, and the minimum field width used to display the value

Formatting Numbers (cont'd.)

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

Summary

- **This chapter covered:**
  - The program development cycle, tools for program design, and the design process
  - Ways in which programs can receive input, particularly from the keyboard
  - Ways in which programs can present and format output
  - Use of comments in programs
  - Uses of variables
  - Tools for performing calculations in programs

Practice Exercises (1)

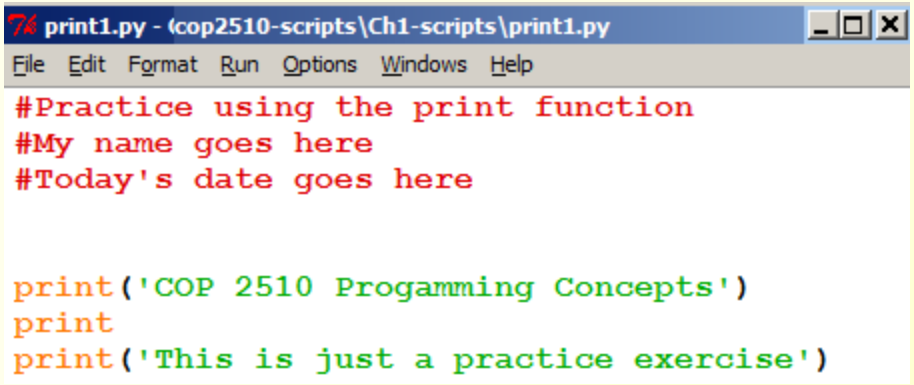
Displaying Output with *print* Function.

Type the lines below in your Python IDLE, save the file as **print1.py,** and run it

```
print('COP 2510 Programming Concepts')
print
print('My name goes here')
```

Using Comments

Add these 2 lines at the top you code, save the file, and run it



Practice Exercises (2)

Variables

Type the lines below in your Python IDLE, save the file as **variables1.py,** and run it

```
#Practice using variables
#My name goes here
#Today's date goes here

# Assign 5 to the width variable
width = 5

# Assign 4 to the length variable
length = 4

#Display the contents of both variables
print (width)
print(length)
```

Practice Exercises (3)

Variables (continued)

Type the lines below in your Python IDLE, save the file as **variables2.py,** and run it

```
#Practice using variables
#My name goes here
#Today's date goes here

room = 501
print('my room number is: ')
print (room)
```

Practice Exercises (4)

Variables (continued)

Type the lines below in your Python IDLE, save the file as **variables3.py,** and run it

```
#Practice using reassigning values to  variables
#My name goes here
#Today's date goes here

dollars = 90.75
print('I have ', dollars , 'in my account')

#Change the value of dollars
dollars = 200.99
print('And now I have ', dollars , 'in my account')
```

Practice Exercises (5)

Reading Input from the Keyboard

Type the lines below in your Python IDLE, save the file as **input1.py,** and run it

```
#Practice using getting input from the keyboard
#My name goes here
#Today's date goes here

#Get the user's first name
first_name = input('Enter your first name:')

#Get the user's last name
last_name = input('Enter your last name: ')

#Print greeting to the user
print('Hello', first_name, last_name)
```

Practice Exercises (6)

Reading Input from the Keyboard (continued)

Type the lines below in your Python IDLE, save the file as **input2.py,** and run it

```
#Practice reading numbers with the input Function
#My name goes here
#Today's date goes here

#Get the user's name, age, and income
name = input('Enter your name:')
age = input('Enter your age:')
income = input('Enter your income')

print('Here is what you entered')
print('Name: ', name)
print('Age: ', age)
print('Income: ', income)
```

Practice Exercises (7)

Performing Calculations

Type the lines below in your Python IDLE, save the file as **calculations1.py,** and run it

```
#Practice performing simple calculations
#My name goes here
#Today's date goes here

#Get hours and payrate
#Convert the value for hours to integer, and convert the value for pay_rate to float/decimal
hours = int(input('How many hours did you work:'))
pay_rate= float(input('How much do you make per hour:'))

#Calculate wages
wages = hours * pay_rate

 print('You worked ', hours , 'Hours')
print('at ', pay_rate, ' per hour')
print('and got paid', wages , 'dollars')
```