**COP1000 – Chapter 2 – Input, Processing, and Output**

1. Designing a Program - Programs must be carefully designed before they are written. During the design process, programmers use tool such as pseudocode and flowcharts to create models of programs.
   1. The Program Development Cycle
      1. More to creating a program than just code
      2. Program development cycle
         1. Design the program
            1. Design program before writing code
            2. Several ways to design a program
         2. Write the code
            1. Programmer begins writing code in high-level language
         3. Correct syntax errors
            1. Almost all programs will have syntax errors when first finished
            2. Programmer will spend some time correcting errors before compiling and executing
         4. Test the program
            1. Tested for logic errors

A mistake that does not prevent the program from running but causes it to produce incorrect results

* + - 1. Correct logic errors
         1. If logic errors exist, programmer debugs the code
         2. Sometimes programmer discovers the original design must be changed and the whole cycle starts over
    1. More About the Design Process
       1. Arguably most important part of the cycle
       2. Program design is the foundation
       3. Can be summarized into two steps
          1. Understand the task that the program is to perform
          2. Determine the steps that must be taken to perform the task
    2. Understanding the Task That the Program is to Perform
       1. Essential to understand what a program is supposed to do before determining the steps to make it perform
       2. Typically, programmer gets the understanding by working with the customer
          1. Professional programmers are hired to design for a person, group, or organization (customer)
          2. Customer can also be boss, or manager of a department withing the company
       3. Programmer usually interviews the customer
          1. Customer will describe the task that the program should perform
          2. Programmer will ask questions to uncover as many details as possible about the task
          3. Follow-up interview is usually needed

Not everything is mentioned by customer or programmer think of additional questions

* + - 1. Programmer studies info and creates a list of software requirements
         1. Software requirement is a single task the program must perform
         2. Once customer agrees that the list is complete, the programmer moves onto the next phase
    1. Determine the Steps That Must Be Taken to Perform the Task
       1. Begin by breaking down the task into a series of steps that another person can follow
       2. List of steps broken down is called an algorithm
          1. Set of well-defined logical steps that must be taken to perform a task
          2. Steps are sequentially ordered
       3. Ex:
          1. Get the number of hours worked
          2. Get the hourly pay rate
          3. Multiply the number of hours worked by the hourly pay rate
          4. Display the result of the calculation performed in step 3
       4. Algorithm not ready to be executed yet
          1. Must be translated into code
       5. Two tools help programmers translate into code
          1. Pseudocode
          2. Flow charts
    2. Pseudocode
       1. Fake code
          1. Informal programming language with no syntax rules
          2. Not meant to be compiled or executed
       2. Used to create models or mock-ups of programs
       3. Since there are no syntax rules, programmers can focus all of their attention on the program’s design
       4. Once programmer is satisfied with pseudocode, translation directly to code can begin
    3. Flowcharts
       1. Another tool programmers use to design
       2. Flowcharts are a diagram that graphically depicts the steps that take place in a program
          1. Figure 2-2, pg. 35
       3. Three types of symbols
          1. Oval

Top and bottom of flow charts

Called terminal symbols

* + - * 1. Parallelograms

Input symbols and output symbols

Represent steps in which the program reads the input or displays output

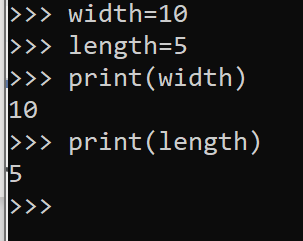
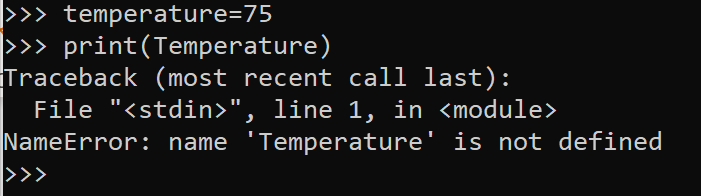
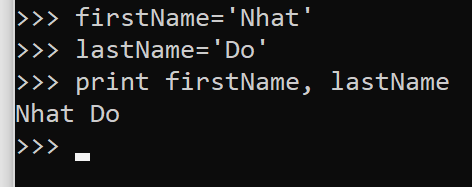
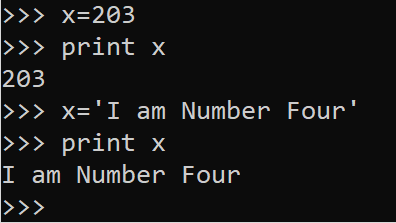
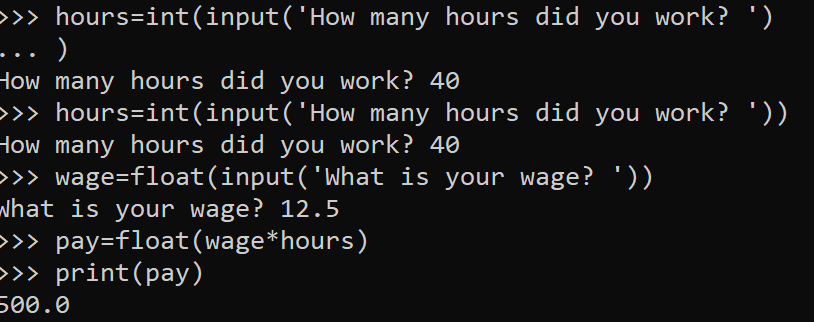
* + - * 1. Rectangles

Processing symbols

Represent steps in which the program performs some process on data

Ex: math equation

* + - 1. Symbols are connected by arrows to show the flow of the program

1. Input, Processing, and Output – Input is 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.
   1. Programs typically perform the same three-step process
      1. Input is received
      2. Some process is performed on the input
      3. Output is produced
   2. Input is any data the program receives while running
      1. Common form is input from a keyboard
   3. Once input is received a process, usually math, is performed on it
      1. Results of the process are sent out as output
2. Displaying Output with the *print* Function – You use the print function to display output in a Python program
   1. Introduction to *print* Function
      1. Function is a prewritten piece of code that performs an operation
         1. Python has many built in functions
         2. Most fundamental is the print function
      2. When executing a function, programmers say they are calling the function
      3. When using calling *print,* you type the function followed by a set of parentheses
         1. Anything inside the parentheses is called an argument
            1. Argument is the data you want displayed
   2. Strings and String Literals
      1. Programs always work with data of some type
         1. Sequences of characters that are used as data is called a *string*
      2. When a string appears in the code of a program it is called a *string literal*
      3. In Python, string literals must be enclosed in quote marks
         1. Quote marks simply mark where the string data begins and ends
      4. String literals can be enclosed in ‘ or “
         1. Figures 2-1 and 2-2, pg. 37
      5. If a string contains an apostrophe or single-quote, double-quote must be used to enclose string literal
         1. Figure 2-3, pg. 38
      6. Single-quotes are used if the string literal contains double-quote marks
         1. Figure 2-4, pg. 38
      7. Triple-quote marks can be used if the string literal contains both single and double-quote marks
         1. Figure 2-4, pg. 38
      8. Triple-quote marks also used to enclose multiple strings
3. Comments – Comments are notes of explanation 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.
   1. Short notes explaining how certain parts of a program work
   2. Critical part of a program, but ignored by interpreter
   3. Intended for any person reading the program’s code
   4. For Python, comments start with #
      1. Interpreter ignores everything from # to the end of the line
      2. Figure 2-5, pg. 39
   5. End-line comments are commonly used
      1. Comment that appears at the end of a line of code
      2. Usually explains the statement that appears in that line
      3. Figure 2-6, pg. 40
   6. Crucial to take extra time to write comments
      1. They save time for all programmers in the future when you have to debug the code
      2. Large and complex programs can be almost impossible to read and understand if they are not properly commented
4. Variables – A variable is a name that represents a value stored in the computer’s memory.
   1. Introduction
      1. Variables are used by programs to access and manipulate data that is stored in memory
      2. A name that represents a value in the computer’s memory
         1. Ex: variable name *tax* to calculate sales tax for online shopping
   2. Creating Variables with Assignment Statements
      1. Used to create a variable that references to a piece of data
         1. Ex: age = 25
         2. Figure 2-4, pg. 41
      2. Written in a general format
         1. variable = expression
            1. expression is going to be a value
      3. “=” is called an assignment operator
      4. Ex:
      5. First statements set variable, second *print* function displays the value
      6. Cannot use a variable until it has an assigned value to it
         1. Must be same capitilization as well
            1. Ex:
   3. Variable Naming Rules
      1. Variable names must follow these rules
         1. Cannot use a key word (Table 1-2)
         2. Cannot contain spaces
            1. Top\_speed
         3. First character must be a letter or underscore
         4. After first character numbers or letters may be used
         5. Uppercase and lowercase letters are distinct
            1. i.e itemsordered and ItemsOrdered are different variables
      2. Names should indicate what the variable is used for
         1. Ex: speed, temperature, distance, instead of x, y, z
      3. Variables can be multiple words
         1. Grosspay, payrate, hotdogssoldtoday
      4. Underscores are sometimes used to separate words
         1. Gross\_pay, pay\_rate, hot\_dogs\_sold\_today
      5. Also can use camelCase naming scheme
         1. grossPay, payRate, hotDogsSoldToday
      6. Table 2-1, pg. 44 shows legal and illegal variable names in Python
   4. Displaying Multiple Items with the *print* Function
      1. Figure 2-9, 2-10, pg. 45-46
   5. Variable Reassignment
      1. Figure2-10
   6. Numeric Data Types and Literals
      1. Data types are used by Python to categorize values in memory
         1. Similar operations on them are carried out in different ways
      2. A number written into code is called a numeric literal
      3. Numeric literals are interpreted according to rules
         1. Numeric literal written as a whole number with no decimal point is considered an *int*
         2. Numeric literal that is written with a decimal point is considered a *float*
      4. Important to be aware of data type when storing in memory
         1. Some operations behave differently depending on the type of data
         2. Some operations can only be done on specific data types
      5. Currency symbols, spaces, commas cannot be sed in numeric literals
   7. Storing Strings with the *str* data type
      1. Along with *int* and *float*, Python uses *str* to store strings in memory
      2. Figure 2-11, pg. 48
      3. Ex:
   8. Reassigning A Variable to a Different Type
      1. Identical variables can be assigned and reassigned
      2. *Print* variable will recall last saved variable
         1. Ex:
         2. Figure 2-7, 2-8, pg. 48
5. Reading Input from the Keyboard – Programs commonly need to read input typed by the user on the keyboard. We will use the Python functions to do that.
   1. Introduction
      1. Variable = input(prompt)
         1. Prompt is a string that is displayed on screen
         2. Ex: name=input(‘What is your name?’)
         3. Refer back to figure 2-12, pg. 50 to learn more
   2. Reading Numbers with the *input* Function
      1. Refer to table 2-2, pg. 51
   3. Nested functions
      1. One function inside another
      2. Ex: hours=int(input('How many hours did you work? '))
      3. Ex:
      4. Figure 2-13, pg. 52
   4. Performing Calculations – Python has numerous operators that can be used to perform mathematical calculations.
      1. Introduction
         1. Programmer’s tools for performing calculations are math operators
            1. Table 2-3 pg. 54 lists all the operators in Python
            2. Variables may be used in a math expression
            3. Calculated values are normally stored to a variable

Ex: pay=wage\*payrate

* + - 1. Floating-Point and Integer Division
         1. Values on left and right of math operator are called operands

Operands are the numbers that the operator interacts with

* + - * 1. / is floating-point division

Ex: 5/2=2.5

* + - * 1. // is integer division

Ex: 5/2=2

Any decimal is dropped when positive

Rounded away from zero to the nearest integer when negative

* + - 1. Operator Precedence
         1. Statements can be written using complex mathematical expressions involving several operators

Ex: outcome=12.0+6.0/3.0

Easy to predict since Python follows the same order of operations

* + - * 1. Operations that are enclosed in parentheses are performed first

When two operators share an operand, the operator with the higher precedence applies first

* + - * 1. Precedence

Exponentiation: \*\*

Multiplication, division, and remainder \* / // %

Addition and subtraction + -

When operators share the same precedence, they are executed left to right

* + - 1. Grouping with Parentheses
         1. Grouping forces some operators to be performed before others

Ex: results=(a+b)/4

* + - 1. The Remainder Operator