CHAPTER 2

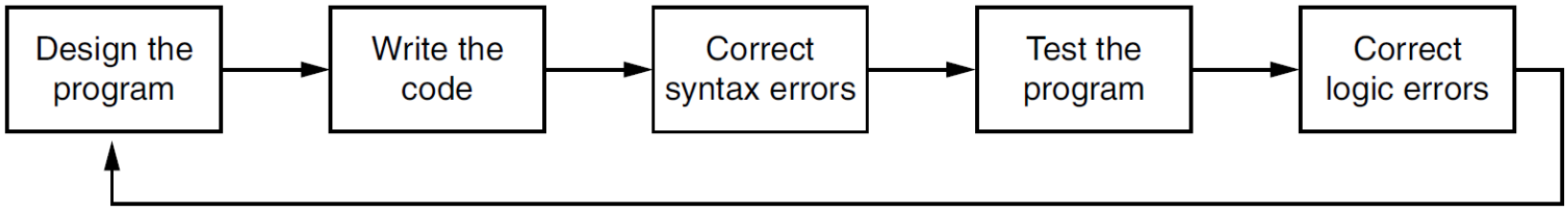
Input, Processing, and Output

Topics

- **Designing a Program**
  - Algorithm , pseudocode and flow charts.
  - We will spend couple of hours on flow charts.
- **Input, Processing, and Output**
- **Displaying Output with `print` Function**
- **Comments**
- **Variables**
- **Reading Input from the Keyboard**
- **Performing Calculations**
- **More About Data Output**
- **Named Constants**
- **Introduction to Turtle Graphics**

Designing a Program

- **Programs must be designed before they are written**
- **Program development cycle:**
  - Design the program
  - Write the code – this is also called implementation.
  - 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. In other word, programmer must know what exactly needed to be done.
  - 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
- **Algorithm: set of well-defined logical steps that must be taken to perform a task**

For example, payment calculator program algorithm:

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 that was performed in step 3.

You may consider the algorithm as the recipe for a delicious food. Every chief has a recipe to cook, so as programmer you might have different approaches to problem. Best one is the most clear and efficient algorithm to solve the problem.

Pseudocode

- **Pseudocode: 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

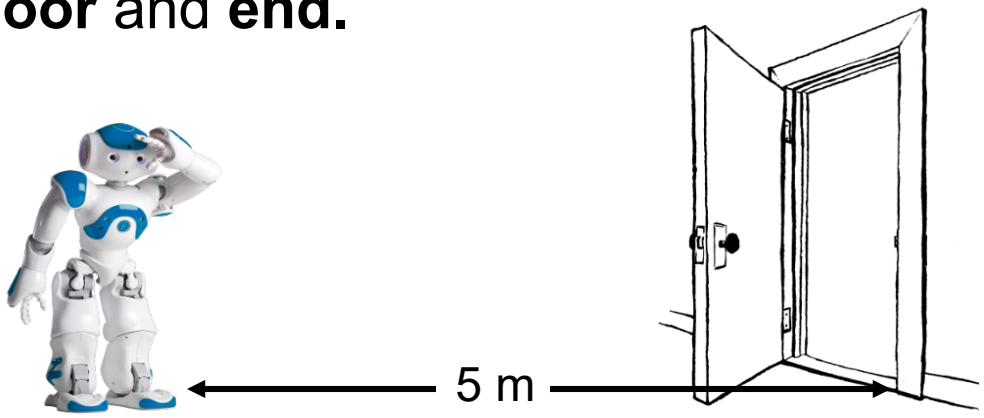
For example, payment calculator program pseudocode:

Input the hours worked
Input the hourly pay rate
Calculate gross pay as hours worked multiplied by pay rate
Display the gross pay

Pseudocode is the program in plain language and can be considered as the intermediate stage between algorithm and implemented/written code.

Example: Pseudocode for a Robot

Write a pseudocode for a robot to get out of the classroom. Consider a robot placed in a classroom close to the room as shown in the figure below. You may use instructions in plain English like **walk, if the door open, open the door, close the door** and **end.**



Pseudocode

Walk 5m

If the Door is open Walk out, Close the door. End

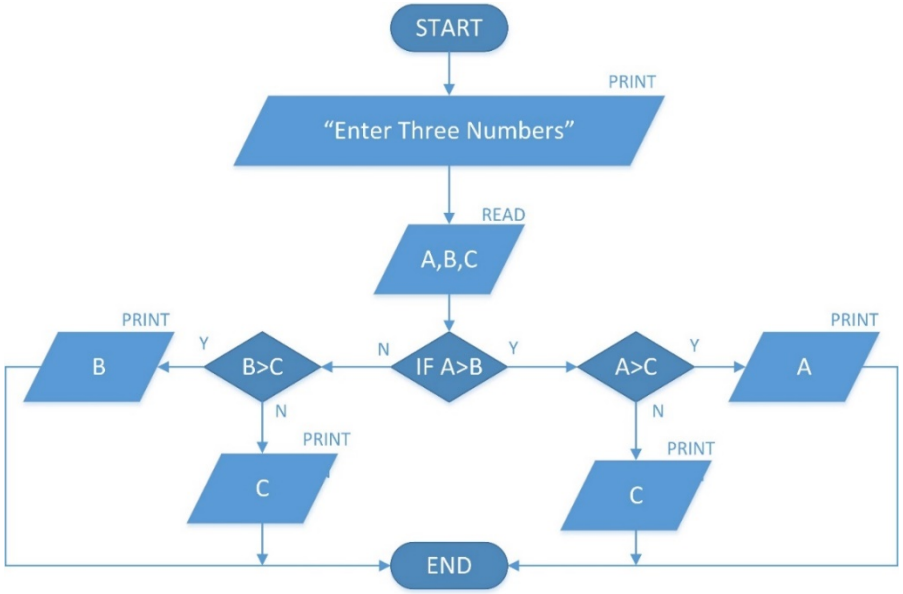
Open the Door Walk out, Close the door. End

Flow Charts

Flowchart: diagram that graphically depicts/represents the steps in a program

As beginner, it is easier to understand concept of programming by using flow-charts. After getting experience, you don't need to use the flow charts to begin with.

Below is a flow-chart for a program that prints the largest of three numbers.

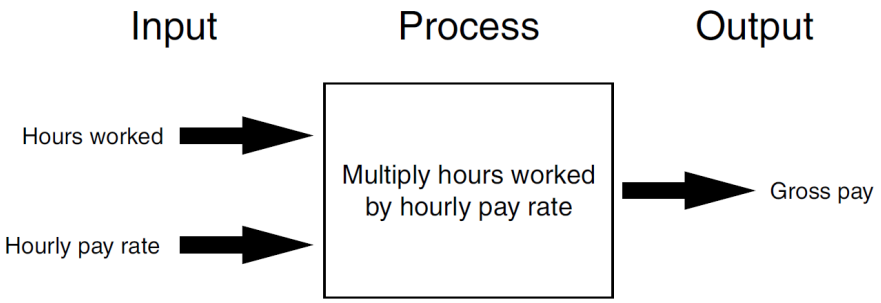


See the flow chart slides to get an idea about how to draw a flow chart.

Input, Processing, and Output

- **Typically, computer performs three-step process**
  1. **Receive input**
     - **Input:** any data that the program receives while it is running
  2. **Perform some process on the input**
     - **Example:** mathematical calculation or determining something/making some decisions
  3. **Produce output**
     - **Output:** any data that the program produce and sends while it is running

Example: Gross Payment Calculation Program



Displaying Output with the `print` Function

- **Function: piece of prewritten code that performs a specific operation/task.**
- **`print` function: displays output on the screen**

Example: **print ('Hello World!')**

- **Argument: data given to a function in ( ).**
  - 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

- **String: sequence of characters that is used as data – string is a text -**
- **String literal: string that appears in Python**
  - Must be enclosed in single (') or double (") quote marks
  - String literal can be also enclosed in triple quotes (''' or """)
    - Enclosed string can contain both single and double quotes and can have multiple lines

```
print( 'Kate Austen' )
print( '123 Full Circle Drive' )
print( 'Asheville, NC 28899' )
```

```
print( "Kate Austen" )
print( "123 Full Circle Drive" )
print( "Asheville, NC 28899" )
```

```
print( """Kate Austen
123 Full Circle Drive
Asheville, NC 28899""" )
```

All 3-programs produce the same output

Program Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

Printing Strings with ’ and ”

- Printing either **a *single-quote or an apostrophe*** as part of the string, you can enclose the string literal in double-quote marks.

```
print("Don't fear!")
print("I'm here!")
```

Program Output

```
Don't fear!
I'm here!
```

- Printing either **a *double-quote or quotation*** as part of the string, you can enclose the string literal in single-quote marks.

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

Program Output

```
Your assignment is to read "Hamlet" by tomorrow.
```

- **Python also allows you** to enclose string literals in triple quotes (either """ or '''). Triple-quoted strings can contain both single quotes and double quotes as part of the string.

```
print("""I'm reading "Hamlet" tonight.""")
```

Program Output

```
I'm reading "Hamlet" tonight.
```

2.7 Write a statement that displays your name.

```
print("John Nash")
```

2.8 Write a statement that displays the following text:
Python's the best!

```
print('Python's the best!')          ✗
print("Python's the best!")          ✓
print("""Python's the best!""")      ✓
print('Python\'s the best!')         ✓
```

2.9 Write a statement that displays the following text:
The cat said "meow."

```
print('The cat said "meow."')              ✓
print("The cat said "meow."")              ✗
print("""The cat said "meow.""""")         ✓
print("The cat said \"meow.\"")            ✓
```

Comments

- **Comments: notes of explanation within a program**
  - Ignored by Python interpreter
    - Intended for a person reading the program's code
  - Begin with a # character, example program given below.

```
# This program displays a person's
# name and address.
print('Kate Austen')
print('123 Full Circle Drive')
print('Asheville, NC 28899')
```

Program Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

- **End-line comment: appears at the end of a line of code**
  - Typically explains the purpose of that line, example given below.

```
# This program displays a person's
# name and address.
print('Kate Austen')              # Display the name.
print('123 Full Circle Drive')    # Display the address.
print('Asheville, NC 28899')      # Display the city, state, and ZIP.
```

Program Output

```
Kate Austen
123 Full Circle Drive
Asheville, NC 28899
```

Variables

- **Variable: 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
- **Assignment statement: used to create a variable and make it reference data**
  - General format is `variable = expression`

Example:

```
name = 'Kate Austen'
age = 29
print(name)
print(age)
print(name,'is',age,'years old')
```

Program Output

```
Kate Austen
29
Kate Austen is 29 years old.
```

- **Assignment operator**: the equal sign (=)

Variables (cont'd.)

Example:

```
name = 'Kate Austen'   #single or double-quotes around strings
age = 29               # No quote used for numbers
print(name)
print(age)
print(name,'is',age,'years old')
```

- **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**
- **Python allows one to display multiple items with a single call to `print`**

Variable Naming Rules

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

Table 2-1 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.

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

- **Data types: categorize value in memory**
  - e.g., int for integer, float for real number, str used for storing strings in memory

```
x = 27                      # x is an integer variable
y = 3.14                    # y is a float variable
z = 'Los Angeles, CA'       # z is a string variable
```

- **Numeric literal: number written in a program**
  - No decimal point considered int, otherwise, considered float

```
a = '27'          #a is numerical literal
b ='3.14'         #b is numerical literal
```

- **Some operations behave differently depending on data type – below is just one example.**

```
print(2*x)
print(2*y)
print(2*z)
print(2*a)
print(2*b)
```

Program Output

```
54
6.28
Los Angeles, CALos Angeles, CA
2727
3.143.14
```

Variable Reassignment

- **Variables can reference different values while program is running**
- **Garbage collection: 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
- **A variable in Python can refer to items of any type**

Figure 2-7 The variable x references an integer

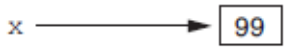
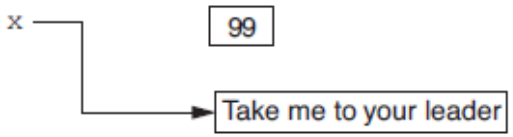


Figure 2-8 The variable x references a string



2.11 Which of the following are illegal variable names in Python, and why?

```
x
99bottles
july2009
theSalesFigureForFiscalYear
r&d
grade_report
```

2.12 Is the variable name `Sales` the same as `sales`? Why or why not?

2.13 Is the following assignment statement valid or invalid? If it is invalid, why?

```
72 = amount
```

2.14 What will the following code display?

```
val = 99
print('The value is', 'val')
```

2.15 Look at the following assignment statements:

```
value1 = 99
value2 = 45.9
value3 = 7.0
value4 = 7
value5 = 'abc'
```

After these statements execute, what is the Python data type of the values referenced by each variable?

2.16 What will be displayed by the following program?

```
my_value = 99
my_value = 0
print(my_value)
```

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

Examples:

```
x = input()                    #Inputting without any prompt
name = input('Enter a string')
school_name = input('Enter School Name:')
```

Reading Numbers with the `input` Function

- **`input` function always returns a string**
- **Built-in Data Conversion functions**
  - `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/gives an exception error

Examples: `function1(function2(argument))`

```
age = int(input('Enter Your Age:'))
weight = float(input('Enter Your Weight:'))
```

Alternatively, we first input them as string then convert to int or float

```
age = input('Enter Your Age:')
weight = input('Enter Your Weight:')
age = int(age)
weight = float(weight)
```

This is longer.
But still sometimes used.

Performing Calculations in Python

- **Math expression: performs calculation and gives a value**
  - Math operator: tool for performing calculation

  Math operators in Python are **+ , - , *, / , % , //** and ******
  - 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

Example: Assume `x = 5` and `y = 2`

```
z=x**y
print('z =',z)
print('x + y =', x + y)
print('x * y =', x * y)
print('x / y =', x / y)
print('x % y =', x % y)
print('x // y=', x // y)
```

Program Output

```
z = 25
x + y = 7
x * y = 10
x / y = 2.5
x % y = 1
x // y= 2
```

Operator Precedence 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

Page 80

Table 2-5 More expressions and their values

Expression	Value
`(5 + 2) * 4`	28
`10 / (5 - 3)`	5.0
`8 + 12 * (6 - 2)`	56
`(6 - 3) * (2 + 7) / 3`	9.0

The Exponent Operator and the Remainder Operator

- **Exponent operator (**): Raises a number to a power**
  - `x ** y` means/gives 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/turns and distances, and to detect odd or even numbers

Examples: What will be the output of the following program.

```
x = 15
y = 21
z = 25
print( 4%2, 5%2 )
print( 3**2, 2**3)
print( x%5, y%5, z%5 )
```

Program Output

```
0 1
9 8
0 1 0
```

See 2-17 Time Converter Program at Page 82 as example to usage of remainder operator.

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

Table 2-7 Algebraic and programming expressions

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

Mixed-Type Expressions and Data Type Conversion in Python

- **Data type resulting from math operation depends on data types of operands**
  - Two `int` values: result is an `int` except division.
  - 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 and fractional part is neglected.

Example: Assume `a=2, b=3, c=3.2, d= 4.3`

```
e = a + b        e will be 5 ( int + int gives int )
f = c + d        f will be 7.5 ( float + float gives float)
g = a + c        g will be 5.2 (int + float gives float)
h = int(c+d)     h will be 7 (.5 fraction is neglected)
```

Note: String and number mixture yields error.

`+` operator on string does a concatenation.

`*` operator on string gives the string n times.

Example: Math Operator Precedence

Page 87

Checkpoint

2.19 Complete the following table by writing the value of each expression in the Value column:

Expression	Value
`6 + 3 * 5`	21
`12 / 2 - 4`	2.0
`9 + 14 * 2 - 6`	31
`(6 + 2) * 3`	24
`14 / (11 - 4)`	2.0
`9 + 12 * (8 - 3)`	69

2.20 What value will be assigned to `result` after the following statement executes?

```
result = 9 // 2
```

Assigns 4 to result variable.

2.21 What value will be assigned to `result` after the following statement executes?

```
result = 9 % 2
```

Assigns 1 to result variable.

Breaking Long Statements into Multiple Lines

- **Multiline continuation character (\): Allows to break a long statement into multiple lines**

```
result = var1 * 2 + var2 * 3 + \
         var3 * 4 + var4 * 5
```

- **Any part of a statement that is enclosed in parentheses can be broken without the line continuation character.**

```
print("Monday's sales are", monday,
      "and Tuesday's sales are", tuesday,
      "and Wednesday's sales are", Wednesday)

total = (value1 + value2 +
         value3 + value4 +
         value5 + value6)
```

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('One', end=' ')
print('Two', end=',')
print('Three', end='.')
print('Four', end='')
print('Five')
```

Program Output

```
One Two,Three.FourFive
```

- **`print` function uses space as item separator**
  - Special argument `sep='`*`delimiter`*`'` causes `print` to use *`delimiter`* as item separator

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

Program Output

```
One Two Three
OneTwoThree
One*Two*Three
```

More About Data Output (cont'd.)

- **Special characters appearing in string literal**
  - Preceded/followed by backslash (\) are called escape characters
    - Examples: newline (\n), horizontal tab (\t)
  - Treated as commands embedded in string

Table 2-8 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.

Example:

```
print("Read \"Hamlet\" by tomorrow.")
print('I\'m ready to begin.')
print('Mon\tTues\tWed')
print('Thur\tFri\tSat')
print('One\nTwo\nThree')
```

Program Output

```
Read "Hamlet" by tomorrow.
I'm ready to begin.
Mon     Tues    Wed
Thur    Fri     Sat
One
Two
Three
```

Formatting Numbers

- **Built-in `format` function can be used to change the format of the displayed numbers**
  - Two arguments in format function:
    - Numeric value to be formatted
    - Format specifier or also called as place holder
    - Usage: `format(variable,'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

Example:

```
>>> print(format(12345.6789, '.1f')) Enter
12345.7
>>>
>>> print('The number is', format(12345.6789, '12.2f')) Enter
The number is     12345.68
>>>
```

Formatting Numbers (cont'd.)

- **The `%` symbol can be used in the format string of `format` function to format number as percentage**
  - `%` specifier is used for float numbers.

```
print(format(0.5, '%'))
print(format(0.5, '.0%'))
print(format(0.5, '.2%'))
```

Program Output

```
50.000000%
50%
50.00%
```

- **To format an integer using `format` function:**
  - `d` specifer is used for float numbers
  - Do not specify precision for integer numbers
  - Can still use `format` function to set field width or comma separator for integer numbers.

```
print(453)
print(format(453,'4d'))
print(format(453,'5d'))
print(format(453,'6d'))
print(format(453,'5d'))
print(format(453,'4d'))
print(453)
```

Program Output

```
453
 453
  453
   453
  453
 453
453
```

See 2-22 in page 93 as example to usage of field widths and precisions.

Magic Numbers

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

  Example: Assume you see the following in a program.

```
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 Problem with Magic Numbers

- **It can be difficult to determine the purpose of the number.**

- **If the magic number is used in multiple places in the program, it can take a lot of effort to change the number in each location, should the need arise.**

- **You take the 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: At the beginning of your program you should define

```
INTEREST_RATE = 0.069
```

- **This creates a named constant named `INTEREST_RATE`, assigned the value 0.069. It can be used instead of the magic number:**

```
amount = balance * INTEREST_RATE
```

Advantages of using Named Constants

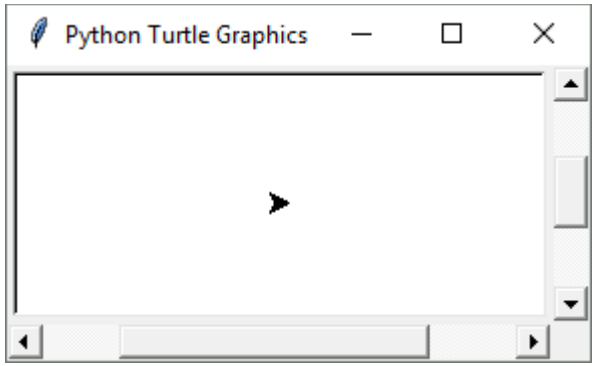
- **Makes code self-explanatory (self-documenting)**
- **Makes code easier to maintain (change) if needed**
- **Helps prevent typographical errors that are common when using magic numbers**

Introduction to Turtle Graphics

- In the late 1960s, an MIT professor used a robotic “turtle” to teach concept of programming. The turtle was tethered to a computer where a student could enter commands, causing the turtle to move. The turtle also had a pen that could be raised and lowered, so it could be placed on a sheet of paper and programmed to draw images. Python has a *turtle graphics* system that simulates a robotic turtle. The system displays a small cursor (as turtle) on the screen. One can use Python statements to move the turtle around the screen, drawing lines and shapes.
- **To use the turtle graphics system, you must import the turtle module with this statement:**

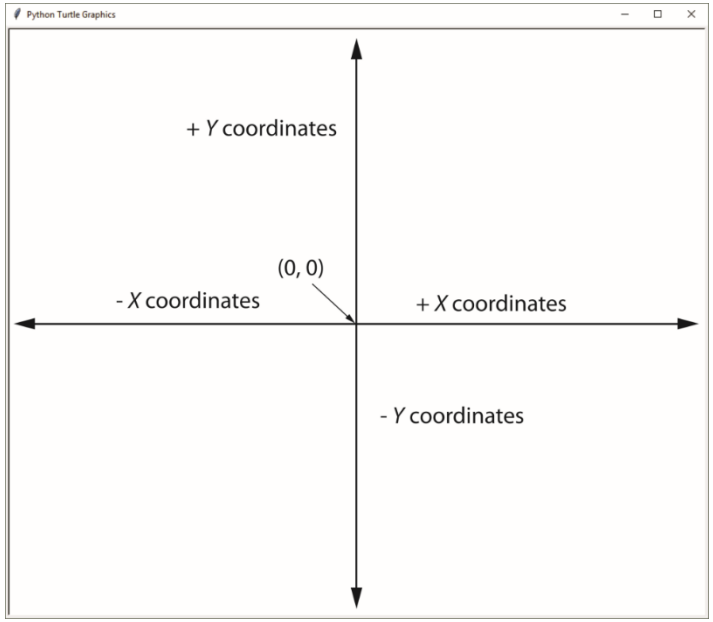
```
import turtle
```

This loads the turtle module into memory



How does the turtle works? Working with Coordinates

- **The turtle uses Cartesian Coordinates**



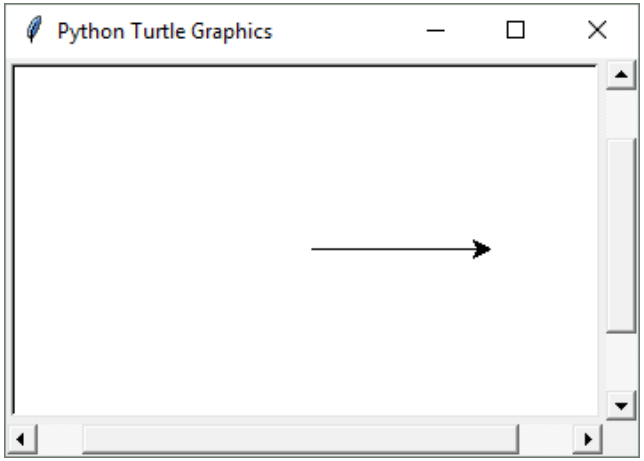
- **`turtle.screensize()` statement shows the maximum `x` and `y` values (position of right top corner) in the screen.**

```
>>> import turtle
>>> turtle.screensize()
(400, 300)
>>>
```

Moving the Turtle Forward

- **Use the `turtle.forward(`*n*`)` statement to move the turtle forward *n* pixels.**

```
>>> import turtle
>>> turtle.forward(100)
>>>
```

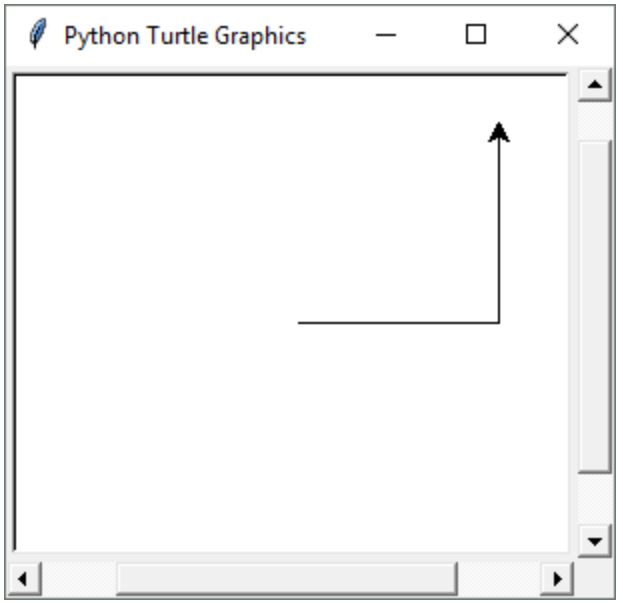


- **`Tip of the arrow shape shows the direction.`**
- **`turtle.backward(`*n*`)` statement to move the turtle backward *n* pixels.**

Turning the Turtle

- **The turtle's initial heading is 0 degrees (east)**
- **Use the `turtle.right(`*`angle`*`)` statement to turn the turtle right by *`angle`* degrees.**
- **Use the `turtle.left(`*`angle`*`)` statement to turn the turtle left by *`angle`* degrees.**

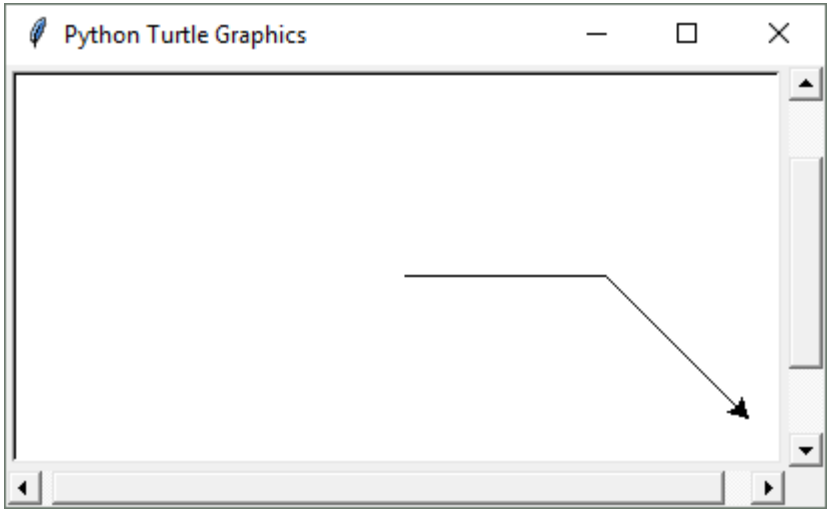
```
>>> import turtle
>>> turtle.forward(100)
>>> turtle.left(90)
>>> turtle.forward(100)
>>>
```



Turning the Turtle (cont'd.)

- **`turtle.right(`*`angle`*`)` statement can be used to turn at specific angles.**

```
>>> import turtle
>>> turtle.forward(100)
>>> turtle.right(45)
>>> turtle.forward(100)
>>>
```

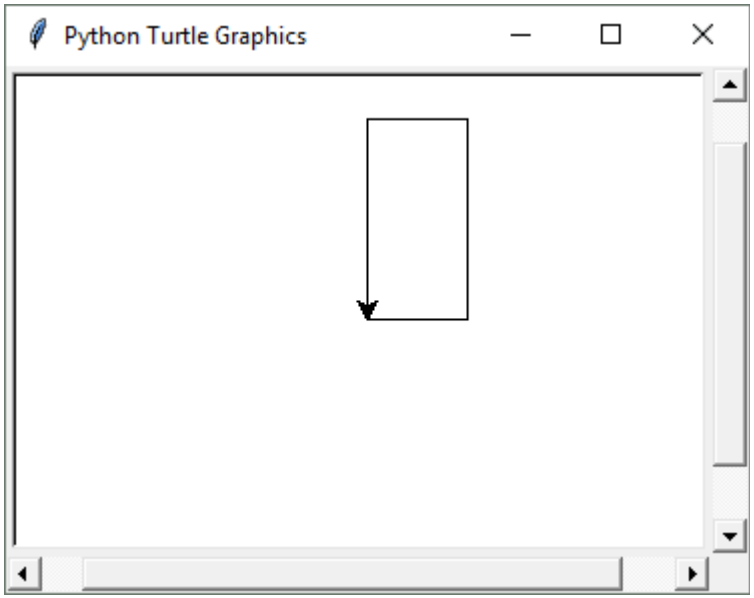


- **Similarly the `turtle.left(`*`angle`*`)` can be used to turn at specific angles.**

Setting the Turtle's Heading

- **Use the `turtle.setheading(`*`angle`*`)` statement to set the turtle's heading to a specific angle.**

```
>>> import turtle
>>> turtle.forward(50)
>>> turtle.setheading(90)
>>> turtle.forward(100)
>>> turtle.setheading(180)
>>> turtle.forward(50)
>>> turtle.setheading(270)
>>> turtle.forward(100)
>>>
```



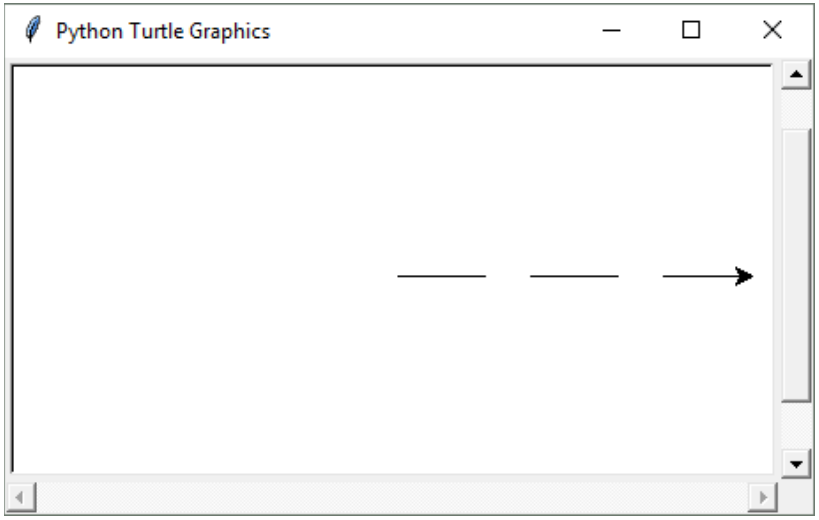
- **This can be also achieved by using turning statements.**

Setting the Pen Up or Down

- **Turtle has a pen which draws as the turtle moves.**
- **When the turtle's pen is down, the turtle draws a line as it moves. *By default, the pen is down.***
- **When the turtle's pen is up, the turtle does not draw as it moves.**
- **Use the `turtle.penup()` statement to raise the pen.**
- **Use the `turtle.pendown()` statement to lower the pen.**

Setting the Pen Up or Down

```
>>> import turtle
>>> turtle.forward(50)
>>> turtle.penup()
>>> turtle.forward(25)
>>> turtle.pendown()
>>> turtle.forward(50)
>>> turtle.penup()
>>> turtle.forward(25)
>>> turtle.pendown()
>>> turtle.forward(50)
>>>
```

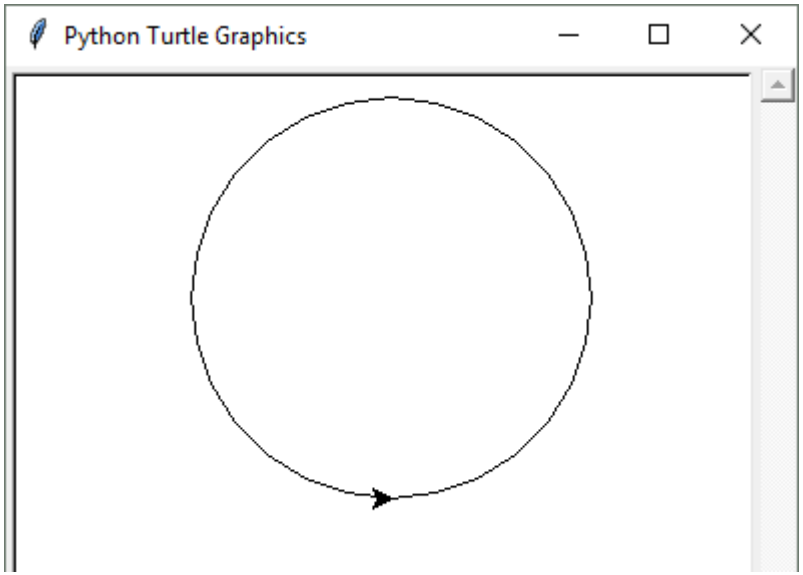


Notice that the Turtle draws lines when the pen is down. While pen is up, the turtle moves but no trace is left.

Drawing Circles

- **Use the `turtle.circle(`*`radius`*`)` statement to move the turtle in a circle with a specified radius.**
- **`turtle.circle(`*`radius`*`)` statement can be used to draw a circle of specific radius.**

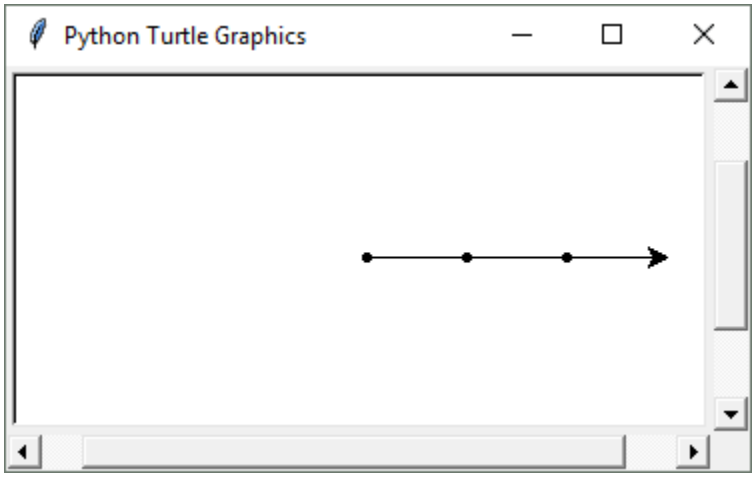
```
>>> import turtle
>>> turtle.circle(100)
>>>
```



Drawing Dots

- **Use the `turtle.dot()` statement to draw a simple dot at the turtle's current location.**

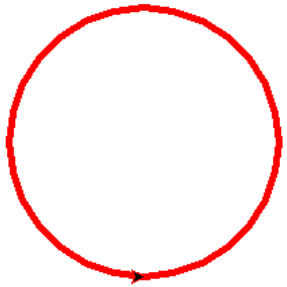
```
>>> import turtle
>>> turtle.dot()
>>> turtle.forward(50)
>>> turtle.dot()
>>> turtle.forward(50)
>>> turtle.dot()
>>> turtle.forward(50)
>>>
```



Pen Size and Drawing Color

- **Use the `turtle.pensize(`*`width`*`)` statement to change the width of the turtle's pen, in pixels.**
- **Use the `turtle.pencolor(`*`color`*`)` statement to change the turtle's drawing color.**

```
>>> import turtle
>>> turtle.pensize(5)
>>> turtle.pencolor('red')
>>> turtle.circle(100)
>>>
```



- *See Appendix D in your textbook for a complete list of colors.*

These are the defined color names that can be used with the turtle graphics library, matplotlib, and tkinter.

'snow'	'ghost white'	'white smoke'
'gainsboro'	'floral white'	'old lace'
'linen'	'antique white'	'papaya whip'
'blanched almond'	'bisque'	'peach puff'
'navajo white'	'lemon chiffon'	'mint cream'
'azure'	'alice blue'	'lavender'

Working with the Turtle's Window

- **Use the `turtle.bgcolor(`*`color`*`)` statement to set the window's background color.**
  - *See Appendix D in your textbook for a complete list of colors*.
- **Use the `turtle.setup(`*`width, height`*`)` statement to set the size of the turtle's window, in pixels.**
  - The *`width`* and *`height`* arguments are the width and height, in pixels.

Example: The following interactive session creates a graphics window that is 640 pixels width and 480 pixels height

```
>>> import turtle
>>> turtle.bgcolor('yellow')
>>> turtle.setup(640, 400)
>>>
```

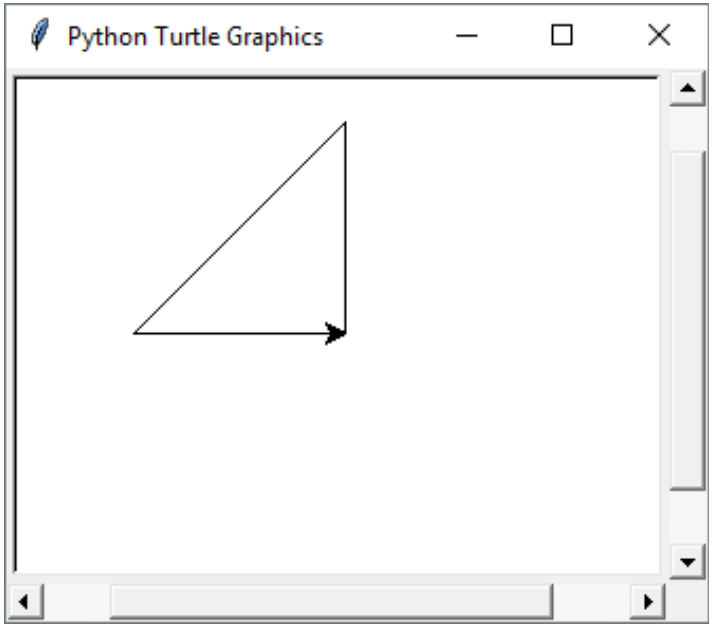
Resetting the Turtle's Window

- **The `turtle.reset()` statement:**
  - Erases all drawings that currently appear in the graphics window.
  - Resets the drawing color to black.
  - Resets the turtle to its original position in the center of the screen.
  - Does *not* reset the graphics window's background color.
- **The `turtle.clear()` statement:**
  - Erases all drawings that currently appear in the graphics window.
  - Does *not* change the turtle's position.
  - Does *not* change the drawing color.
  - Does *not* change the graphics window's background color.
- **The `turtle.clearscreen()` statement:**
  - Erases all drawings that currently appear in the graphics window.
  - Resets the drawing color to black.
  - Resets the turtle to its original position in the center of the screen.
  - Resets the graphics window's background color to white.

Moving the Turtle to a Specific Location

- **Use the `turtle.goto(x, y)` statement to move the turtle to a specific location.**

```
>>> import turtle
>>> turtle.goto(0, 100)
>>> turtle.goto(-100, 0)
>>> turtle.goto(0, 0)
>>>
```



- The `turtle.pos()` statement displays the turtle's current *X*,*Y* coordinates.
- The `turtle.xcor()` statement displays the turtle's current *X* coordinate and the `turtle.ycor()` statement displays the turtle's current *Y* coordinate.

Animation Speed

- **Use the `turtle.speed(speed)` command to change the speed at which the turtle moves.**
  - The *`speed`* argument is a number in the range of 0 through 10.
  - If you specify 0, then the turtle will make all of its moves instantly (animation is disabled).

Hiding and Displaying the Turtle

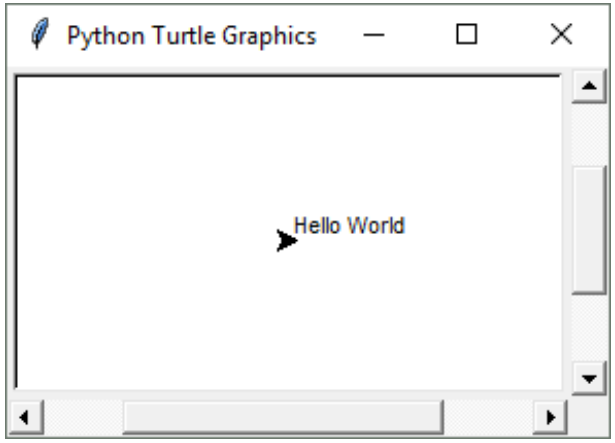
- **Use the `turtle.hideturtle()` command to hide the turtle.**
  - This command does not change the way graphics are drawn, it simply hides the turtle icon.

- **Use the `turtle.showturtle()` command to display the turtle.**

Displaying Text

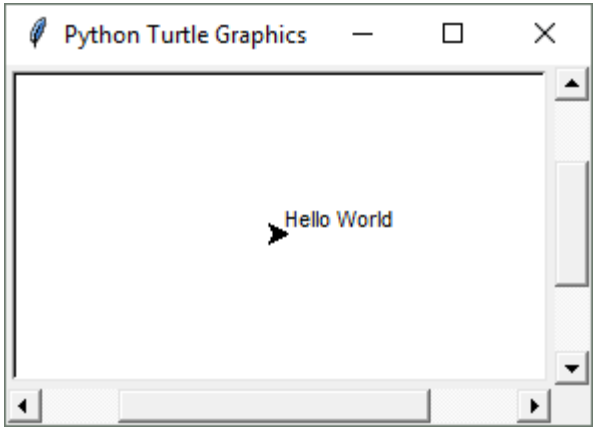
- **Use the `turtle.write(`*`text`*`)` statement to display text in the turtle's graphics window.**
  - The *`text`* argument is a string that you want to display.
  - The lower-left corner of the first character will be positioned at the turtle's *X* and *Y* coordinates.

```
>>> import turtle
>>> turtle.write('Hello World')
>>>
```



Displaying Text

```
>>> import turtle
>>> turtle.write('Hello World')
>>>
```

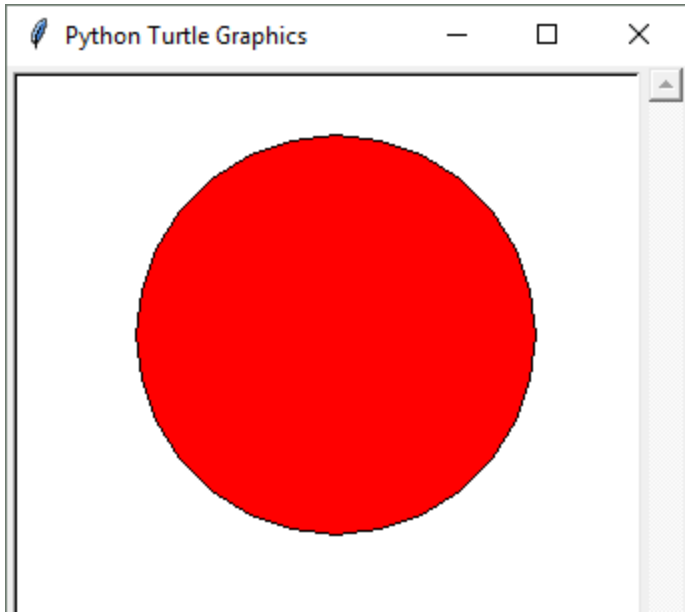


Filling Shapes

- **To fill a shape with a color:**
  - Use the `turtle.begin_fill()` command before drawing the shape
  - Then use the `turtle.end_fill()` command after the shape is drawn.
  - When the `turtle.end_fill()` command executes, the shape will be filled with the current fill color. But it has to be a closed area.

Filling Shapes

```
>>> import turtle
>>> turtle.hideturtle()
>>> turtle.fillcolor('red')
>>> turtle.begin_fill()
>>> turtle.circle(100)
>>> turtle.end_fill()
>>>
```



Keeping the Graphics Window Open

- **When running a turtle graphics program outside IDLE, the graphics window closes immediately when the program (.py file) is done.**

- **To prevent this, add the `turtle.done()` statement to the very end of your turtle graphics programs.**
  - This will cause the graphics window to remain open, so you can see its contents after the program finishes executing.

Turtle Graphics Commands - Summary

- **import turtle** :To use the turtle graphics system, you must import the turtle module with this statement.
- **turtle.screensize()**: statement shows the maximum x and y values.
- **turtle.forward(n)**: statement to move the turtle forward n pixels.
- **turtle.backward(n)**: statement to move the turtle backward n pixels.
- **turtle.right(angle)**: statement to turn the turtle right by angle degrees.
- **turtle.left(angle)**: statement to turn the turtle left by angle degrees.
- **turtle.setheading(angle)**: statement to set the turtle's heading to a specific angle.
- **turtle.penup()**: statement to raise the pen.NO DRAWING
- **turtle.pendown()**: statement to lower the pen. DRAWING. By default pen is down.
- **turtle.circle(radius)**: statement to move the turtle in a circle with a specified radius.
- **turtle.pensize(width)**: statement to change the width of the turtle's pen, in pixels.
- **turtle.pencolor(color)**: statement to change the turtle's drawing color.
- **turtle.bgcolor(color)**: statement to set the window's background color.
- **turtle.setup(width, height)**: statement to set the size of the turtle's window, in pixels.
- **turtle.reset()**: erases all content, reset color and position. But does not change background color.
- **turtle.clear()**: erases all content. But doesn't change any setting.
- **turtle.clearscreen()**: erases all content and resets all setting including background color and screen size.
- **turtle.goto(x, y)**: statement to move the turtle to a specific location.
- **turtle.pos()**: statement displays the turtle's current X,Y coordinates.
- **turtle.xcor()**: statement displays the turtle's current X coordinate.
- **turtle.ycor()**: statement displays the turtle's current Y coordinate.
- **turtle.goto(x, y)**: statement to move the turtle to a specific location.
- **turtle.speed(speed)**: statement to change the speed at which the turtle moves.0-10 and 0 means no animation. 1-slowest
- **turtle.hideturtle()**: statement to hide the turtle.
- **turtle.showturtle()**: statement to display the turtle.
- **turtle.write(text)**: statement to display text in the turtle's graphics window.
- **turtle.begin_fill()**: command before drawing the shape
- **turtle.end_fill()**: command after the shape is drawn. The shape will be filled with the current fill color.
- **turtle.fillcolor(color)**: command to determine the color of shape fill.
- **turtle.done()**: statement to the very end of your turtle graphics programs. for IDLE not necessary.

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 and named constants
  - Tools for performing calculations in programs
  - The turtle graphics system

Review Questions

Multiple Choice

1. A _____ error does not prevent the program from running, but causes it to produce incorrect results.
   a. syntax
   b. hardware
   c. logic
   d. fatal

6. A _____ is a sequence of characters.
   a. char sequence
   b. character collection
   c. string
   d. text block

9. A string literal in Python must be enclosed in _____.
   a. parentheses.
   b. single-quotes.
   c. double-quotes.
   d. either single-quotes or double-quotes.

Review Questions

Page 123-124

Multiple Choice

11. A(n) _____ makes a variable reference a value in the computer's memory.
    a. variable declaration
    b. assignment statement
    c. math expression
    d. string literal

12. This symbol marks the beginning of a comment in Python.
    a. &
    b. *
    c. **
    d. #

19. Which built-in function can be used to read input that has been typed on the keyboard?
    a. `input()`
    b. `get_input()`
    c. `read_input()`
    d. `keyboard()`

Programming Exercises

3. Pounds to Kilograms

One pound is equivalent to 0.454 kilograms. Write a program that asks the user to enter the mass of an object in pounds and then calculates and displays the mass of the object in kilograms.

Program

```
# Get the number of pounds.
pounds = float(input('Enter a value in pounds: '))
# Calculate the amount of kilograms.
kilos = pounds * 0.454
# Print the result.
print(pounds, 'pounds is', format(kilos, '.2f'), 'kilograms.')
```

Example Program Output

```
Enter a value in pounds: 3.41
3.41 pounds is 1.55 kilograms.
```

Programming Exercises

8. Tip, Tax, and Total

Write a program that calculates the total amount of a meal purchased at a restaurant. The program should ask the user to enter the charge for the food, then calculate the amounts of a 18 percent tip and 7 percent sales tax. Display each of these amounts and the total.

Program

```
# Declare variables for food charges, tip, tax, and total.
food = 0.0
tip = 0.0
tax = 0.0
total = 0.0
# Constants for the tax rate and tip rate.
TAX_RATE = 0.07
TIP_RATE = 0.18
# Get the food charges.
food = float(input("Enter the charge for food: "))
tip = food * TIP_RATE    # Calculate the tip.
tax = food * TAX_RATE    # Calculate the tax.
total = food + tip + tax # Calculate the total.
# Print the tip, tax, and total.
print ("Tip: $", format(tip, '.2f'))
print ("Tax: $", format(tax, '.2f'))
print ("Total: $", format(total, '.2f'))
```

Example Program Output

```
Enter the charge for food: 30
Tip: $ 5.40
Tax: $ 2.10
Total: $ 37.50
```

Programming Exercises

9. Circle Measurements

Write a program that asks the user to enter the radius of a circle. The program should calculate and display the area and circumference of the circle using πr^2 for the area and 2πr for the circumference.

Hint: You can either use 3.14159 as the value of pi (π), or add the statement "import math" to the start of the program and then use "`math.pi`" wherever you need the value of pi in the program.

Program

```
# Import the math module. - for math.pi -
import math
# Get radius from user.
radius = float(input('Enter radius of circle: '))
area = math.pi * radius ** 2          # Calculate area
circumference = 2 * math.pi * radius # Calculate circumference
# Print the results.
print('\nArea of circle:', format(area, '.2f'))
print('Circumference of circle:', format(circumference, '.2f'))
```

Example Program Output

```
Enter radius of circle: 2.24

Area of circle: 15.76
Circumference of circle: 14.07
```

Out of Book Exercise

Arithmetic: Write a program that asks the user to enter two numbers, obtains them from the user and prints their sum, product, difference, quotient and remainder.

Program

```
# Getting two integers from user
# Assigning the entered numbers to x and y
x=int(input('Enter the first number:'))
y=int(input('Enter the second number:'))
#Printing the results
print('Sum of numbers is',x+y)
print('Product of numbers is',x*y)
print('Difference of numbers is',x-y)
print('Quotinent of numbers is',x/y)
print('Remainder of numbers is',x%y
```

Example Program Output

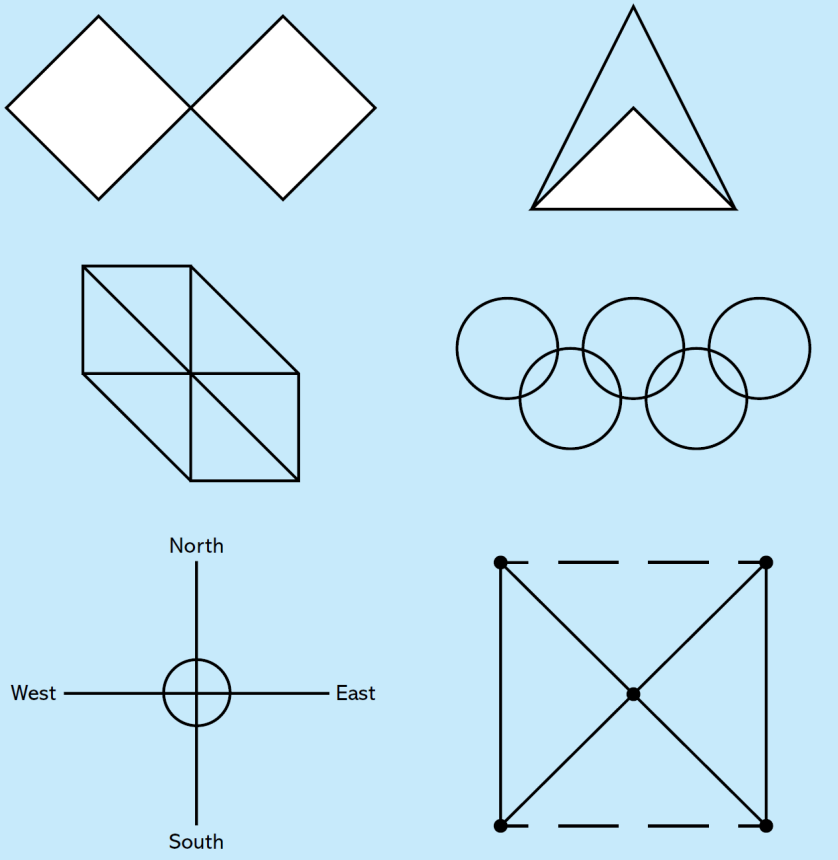
```
Enter the first number:12
Enter the second number:5
Sum of numbers is 17
Product of numbers is 60
Difference of numbers is 7
Quotinent of numbers is 2.4
Remainder of numbers is 2
```

Programming Exercises

15. Turtle Graphics Drawings

Use the turtle graphics library to write programs that reproduce each of the designs shown in Figure 2-34.

Figure 2-34 Designs

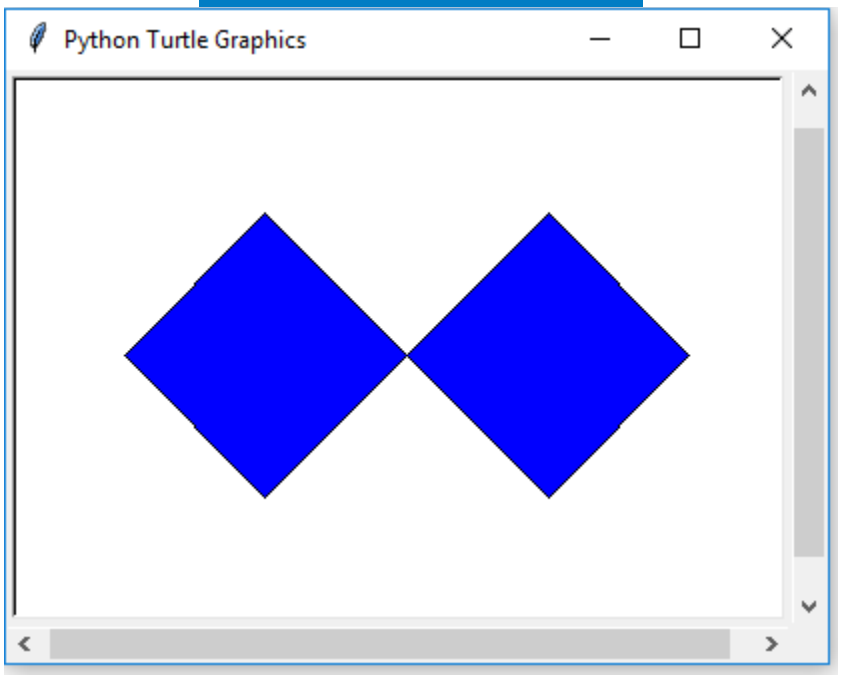


Drawing Two Diamonds

15. Turtle Graphics Drawing

```
import turtle
# Hide the turtle.
turtle.hideturtle()

# Set the fill color to blue.
turtle.fillcolor('blue')

# Draw the first diamond.
turtle.begin_fill()
turtle.left(135)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.end_fill()

# Draw the second diamond.
turtle.begin_fill()
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.end_fill()
```

Program Output

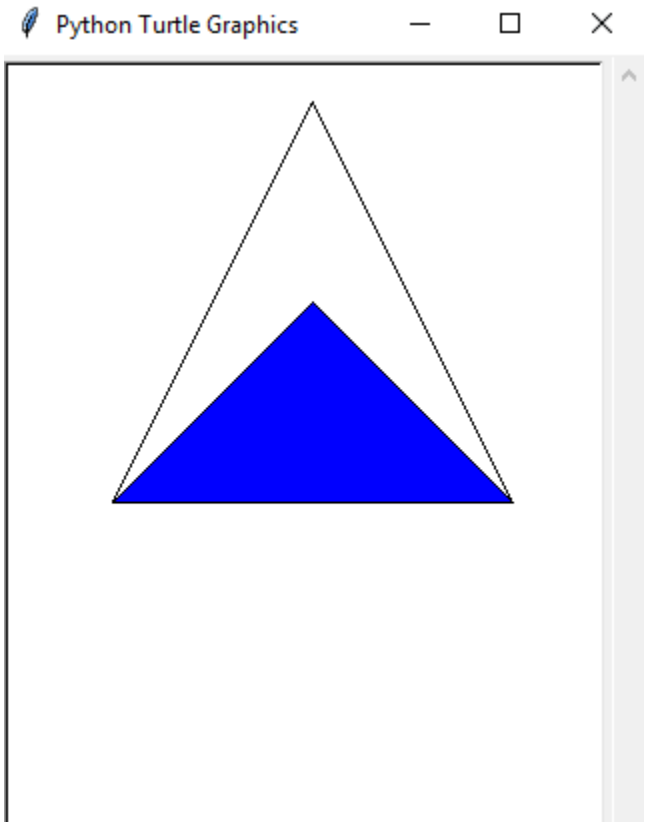


Drawing Double Triangles

15. Turtle Graphics Drawing

```
import turtle
# Named constants - Coordinates
OUTER_TOP_X = 0
OUTER_TOP_Y = 200
INNER_TOP_X = OUTER_TOP_X
INNER_TOP_Y = OUTER_TOP_Y / 2
BASE_LEFT_X = -100
BASE_LEFT_Y = 0
BASE_RIGHT_X = 100
BASE_RIGHT_Y = 0

# Hide the turtle and raise the pen.
turtle.hideturtle()
turtle.penup()

# Move the pen to the bottom right corner.
turtle.goto(BASE_RIGHT_X, BASE_RIGHT_Y)

# Set the fill color to blue and lower the pen.
turtle.fillcolor('blue')
turtle.pendown()

# Draw the outer triangle.
turtle.goto(OUTER_TOP_X, OUTER_TOP_Y)
turtle.goto(BASE_LEFT_X, BASE_LEFT_Y)
turtle.goto(BASE_RIGHT_X, BASE_RIGHT_Y)

# Draw the inner triangle.
turtle.begin_fill()
turtle.goto(INNER_TOP_X, INNER_TOP_Y)
turtle.goto(BASE_LEFT_X, BASE_LEFT_Y)
turtle.goto(BASE_RIGHT_X, BASE_RIGHT_Y)
turtle.end_fill()
```

Program Output

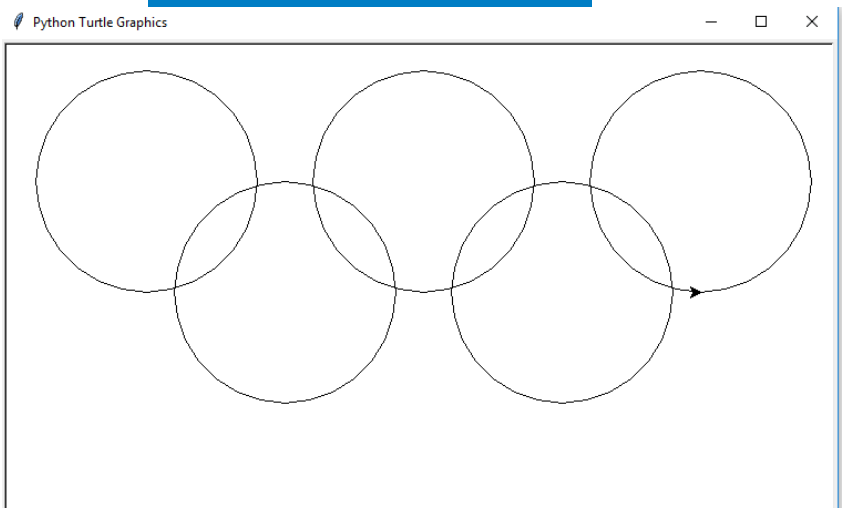


5 Circles

15. Turtle Graphics Drawing

```
import turtle
RADIUS = 100 # Named constants
STARTING_POINT_X = -250
STARTING_POINT_Y = 0
HSHIFT = 125
VSHIFT = 100
x = STARTING_POINT_X # Drawing circle #1
y = STARTING_POINT_Y
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.circle(RADIUS)
x += HSHIFT          # Drawing circle #2
y -= VSHIFT
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.circle(RADIUS)
x += HSHIFT          # Drawing circle #3
y = 0
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.circle(RADIUS)
x += HSHIFT          # Drawing circle #4
y -= VSHIFT
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.circle(RADIUS)
x += HSHIFT          # Drawing circle #5
y = 0
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.circle(RADIUS)
```

Program Output



Drawing Cardinal Directions

15. Turtle Graphics Drawing

```
import turtle
# Named constants
CENTER_X = 0
CENTER_Y = 0
X_AXIS_LENGTH = 200
Y_AXIS_LENGTH = 200
RADIUS = 25  SOUTH = 270 EAST = 0
# Hide the turtle and set the animation speed.
turtle.hideturtle()
turtle.speed(0)
# Draw the X axis
x = CENTER_X - (X_AXIS_LENGTH / 2) y = CENTER_Y
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.forward(X_AXIS_LENGTH)
# Draw the Y axis
x = CENTER_X y = CENTER_Y + (Y_AXIS_LENGTH / 2)
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.setheading(SOUTH)
turtle.forward(X_AXIS_LENGTH)
# Draw the center circle
x = CENTER_X     y = CENTER_Y - RADIUS
turtle.penup()
turtle.setheading(EAST)
turtle.goto(x, y)
turtle.pendown()
turtle.circle(RADIUS)
# Write "North"
x = CENTER_X - 10 y = CENTER_Y + (Y_AXIS_LENGTH / 2)
turtle.penup()
turtle.goto(x, y)
turtle.pendown()
turtle.write("North")
```

Program Output