



RV College of Engineering®

Autonomous  
Institution Affiliated  
to Visvesvaraya  
Technological  
University, Belagavi

Approved by AICTE  
New Delhi

# Introduction to **Python** Programming



## UNIT-1

**Prof. Rajesh R M**  
Dept. of AIML  
RV College of Engineering  
Bengaluru

*Go, Change the World*

# Outline

## → Getting Started

- Introducing Python
- Setting Up Python in windows
- Setting Up Python in other Operating Systems
- Introducing IDLE

## → Types, Variable, and Simple I/O

- Using Quotes with Strings
- Concatenating and Repeating Strings
- Working with Numbers
- Understanding the Variable
- Getting User Input
- Converting Values



# Introduction to Python

- Python is a powerful yet easy-to-use programming language
  - Developed by Guido van Rossum
  - First released in 1991.
- With Python, you can quickly write a small project
- But Python also scales up nicely and can be used for mission-critical, commercial applications



# Introduction to Python (Contd.)

The major goal of any programming language is to bridge the gap between the programmer's, brain and the computer



# Introduction to Python (Contd.)

- There are a lot of programming languages out there

## What's so great about Python?

Python Is Easy to Use

Python Is Powerful

Python Is Object-Oriented

Python Is a “Glue” Language

Python Runs Everywhere

Python Has a Strong Community

Python Is Free and Open Source





# Introduction to Python (Contd.)

## Python Is Easy to Use

- Most of the popular languages you've probably heard of, like Visual Basic, C#, and Java, are considered high-level languages, which means that they're closer to human language than machine language
- But Python, with its clear and simple rules, is even closer to English
- Creating Python programs is so straightforward that it's been called “programming at the speed of thought”
- Python programs are shorter and take less time to create than programs in many other popular languages

# Introduction to Python (Contd.)

## Python Is Easy to Use

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
    printf("Hello, World!\n");
}
```

- Java

```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```

- now in Python

```
print "Hello, World!"
```



# Introduction to Python (Contd.)

## Python Is Powerful

- Python has all the power you'd expect from a modern programming language
- Python employ a GUI, process files, and use a variety of data structures
- Python is powerful enough to attract developers from around the world as well as companies such as
  - Google, IBM, Industrial Light + Magic, Microsoft, NASA, Red Hat, Verizon, Xerox, and Yahoo!
- Python is also used as a tool by professional game programmers. Electronic Arts, Games, and the Disney Interactive Media Group all publish games that incorporate Python



# Introduction to Python (Contd.)

## Python Is Object-Oriented

- Object-Oriented Programming (OOP) is a modern approach to solving problems with computers
- It embodies an intuitive method of representing information and actions in a program
- It's certainly not the only way to write programs, but, for large projects, it's often the best way to go
- Languages like C#, Java, and Python are all object-oriented. But Python does them one better.
  - In C# and Java, OOP is not optional. This makes short programs unnecessarily complex
- Python takes a different approach
  - In Python, using OOP techniques is optional
  - You have all of OOP's power at your disposal, but you can use it when you need it

# Introduction to Python (Contd.)

## Python Is Object-Oriented

- C

```
#include <stdio.h>

int main(int argc, char ** argv)
{
    printf("Hello, World!\n");
}
```

- Java

```
public class Hello
{
    public static void main(String argv[])
    {
        System.out.println("Hello, World!");
    }
}
```

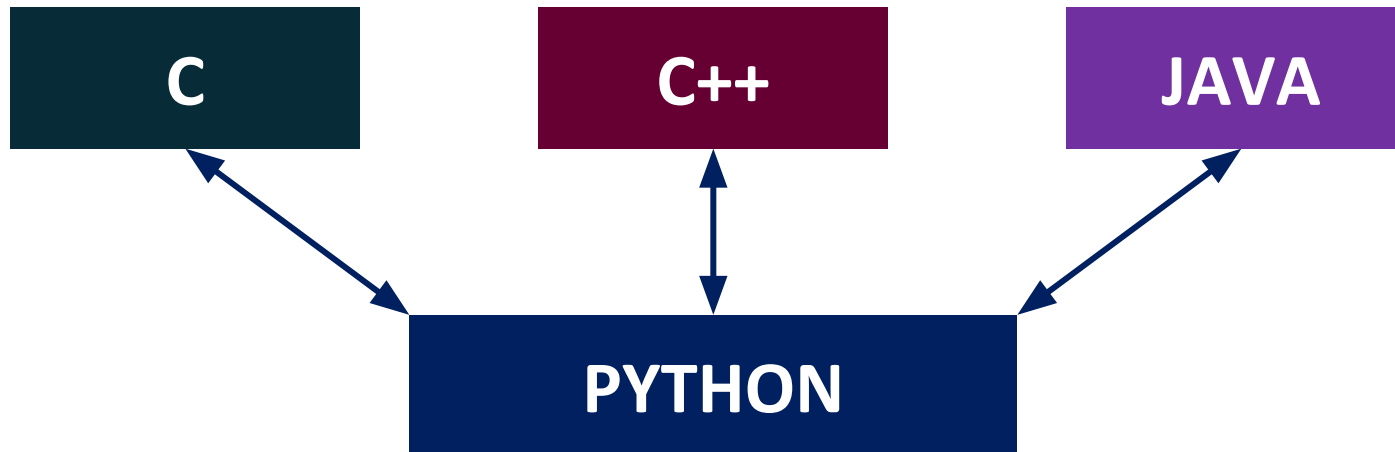
- now in Python

```
print "Hello, World!"
```

# Introduction to Python (Contd.)

## Python Is a “Glue” Language

- Python can be integrated with other languages such as C, C++, and Java. This means that a programmer can take advantage of work already done in another language while using Python
- It also means that he or she can leverage the strengths of other languages, such as the extra speed that C or C++ might offer, while still enjoying the ease of development that's a hallmark of Python programming.





# Introduction to Python (Contd.)

## Python Runs Everywhere

- You can run Python on **Windows, Macintosh, or Linux machines**
- Python programs are **platform independent**, which means that regardless of the operating system you use to create your program, it'll run on any other computer with Python
- So if you write a program on your PC, you can e-mail a copy to your friend who runs Linux or to your aunt who has a Mac, and the program will work
- Only condition **Python installed on their computers**



# Introduction to Python (Contd.)

## Python Has a Strong Community

- Most programming languages have a dedicated newsgroup, but Python also has something called the Python Tutor mailing list, a more informal way for beginning programmers to ask those first questions
- The list is at: <http://mail.python.org/mailman/listinfo/tutor>
- Although the list is called Tutor, anyone, whether novice or expert, can answer questions
- There are other Python communities focused on different areas, but the common element they share is that they tend to be friendly and open. That only makes sense since the language itself is so approachable for beginners



# Introduction to Python (Contd.)

## Python Is Free and Open Source

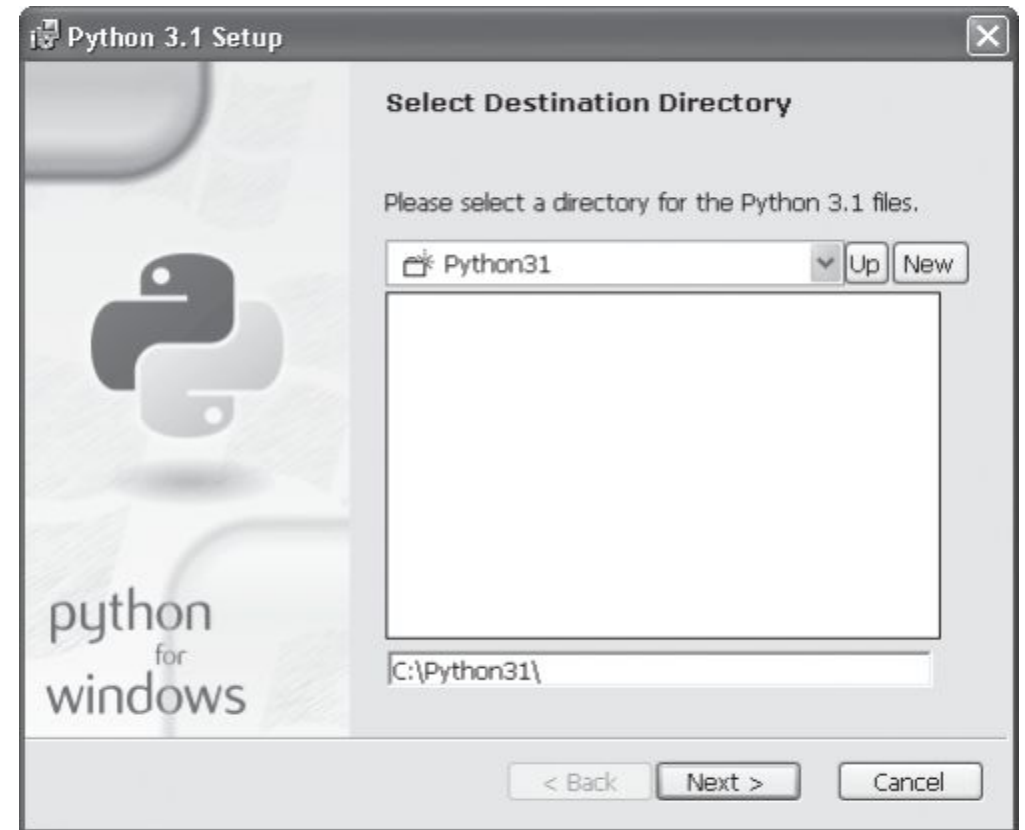
- Python is **free**. You can install it on your computer and **never pay a penny**
- But Python's license lets you do much more than that. **You can copy or modify Python. You can even resell Python**
- if you want (but don't quit your day job just yet). Embracing open-source ideals like this is part of what makes Python so popular and successful.

# Setting Up Python in Windows

- Before you can jump in and write your first Python program, you need to get the language on your computer

## → Installing Python on Windows

1. Download the Python Windows installer from the companion website ([www.courseptr.com/downloads](http://www.courseptr.com/downloads)). The file is in the Software folder, inside the Python subfolder; the file name is python-3.1.msi.
2. Run the Python Windows Installer, python-3.1.msi. Figure 1.2 shows the installer in action.
3. Accept the default configuration. Once you're done, you'll have Python 3.1 on your system.



# Introducing IDLE

- Python comes with an **Integrated Development Environment (IDE)** called **IDLE**
- A development environment is a **set of tools that makes writing programs easier**
- You can think of it as a word processor for your programs. But it's even more than **a place to write, save, and edit your work**
- IDLE provides two modes in which to work
  - An interactive Mode
  - Script Mode



**IDLE IDE**





# Introducing IDLE (Contd.)

- Programming in Interactive Mode
  - Finally, it's time to get your hands dirty with some actual Python programming.
  - The quickest way is to start Python in interactive mode. In this mode, you can tell Python what to do and it'll respond immediately.
- Writing Your First Program
  - To begin your interactive session, from the Start menu, choose All Programs, Python 3.1, IDLE (Python GUI)

# Variable

- A variable is the **name of a memory location**
- Python is **weakly typed**, i.e., you don't declare variables to be a specific type
- A variable **has the type that corresponds to the value you assign to it**
- Variable names **begin with a letter or an underscore and can contain letters, numbers, and underscores**
- Python has **reserved words** that you can't use as variable names
- Example: class, def, else, except, try, if, while, etc.

```
x = 50  
type(x)
```

```
a = "Hello"  
type(x)
```

```
y = '5.000'  
type(x)
```

**x**

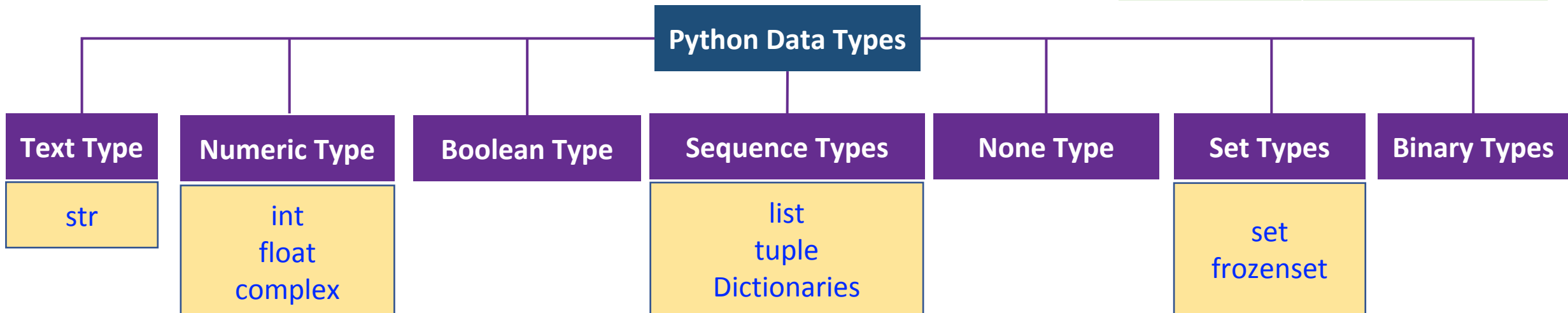


Memory	
10000	
10004	
10008	50
10012	
10016	

# Python Data Types

- Every value in Python has a **datatype**
- In python, the **interpreter** implicitly binds the value with its **type**
- We can use the **type()** function along with the variable to know the type of the data

Employee Details	
Employee Id	00001823
Name	Peter
Age	42
Salary	150000.68
Address	Jayanagar, Bangalore
Zip Code	560 001





# Using Quotes With the Strings

- You can use either a pair of single ( ' ') or double quotes ( " ") to create string values

```
print("Hello RVCE")
```

- To print multiple values with a single call to the print() function—just list multiple argument values, separated by commas.

```
print("Same", "message", "as before")
```

```
print ("Just",  
      "a bit",  
      "bigger")
```

# Using Quotes With the Strings (Contd.)

## → Text Type

- Text type is a sequence characters
- Strings start and end with quotation mark " or apostrophe ' characters
- Examples:  
"hello"  
"This is a string"  
"This, too, is a string. It can be very long!"
- Escape Sequence: An escape sequence is a sequence of characters that does not represent itself when used

## Some String Operations

```
# Declaring & Printing the String  
x = "RV College of Engineering"  
print (x)
```

```
# Getting the length of the String  
print(len(x))
```

```
# Concatenation Operation  
print(x+"-"+"AIML")
```

```
# Concatenation Operation  
print("Pie" * 10)
```

# Using Quotes With the Strings (Contd.)

## Specifying a Final String to Print

- By default, the `print()` function prints a newline character as a final value. This means that a subsequent call to `print()` would display text on the following line
- Generally, python allows you to specify your own final string to be printed
- For Example: Specify that a space be the final character printed (instead of the newline) when you call the `print()` function. This would mean a subsequent `print()` statement would begin printing values right after that space.

```
print("Here", end=" ")  
print("it is...")
```

# Using Quotes With the Strings (Contd.)

## Creating Triple-Quoted Strings

- Certainly, the coolest part of the program is that it prints out “Game Over” in a big block of text

```
"""
  _ _ _ _ _
 / _ | / _ | / _ | / _ | / _ |
| | | / / | / / | / / | / / |
| | | / _ | / _ | / _ | / _ |
 \ _ | / / | / / | / / | / / |
  _ _ _ _ _

 / _ \ | | / _ \ | | / _ \ | |
| | | | | / / | | | | | / / |
| | | | | / _ | | | | | / _ |
 \ _ \ | | | | | \ _ \ | | |
  _ _ _ _ _
"""
```

- Triple quotes are used for
  - Multi-line String Creation
  - Multi-line Commenting

```
my_str = """
    I
    am
    a
    Geek !
    """
```

```
# check data type of
print(type(my_str))
print(my_str)
```

# Using Quotes With the Strings (Contd.)

## Using Escape Sequences with Strings

- Escape sequences allow you to put special characters into your strings
- They give you greater control and flexibility over the text you display

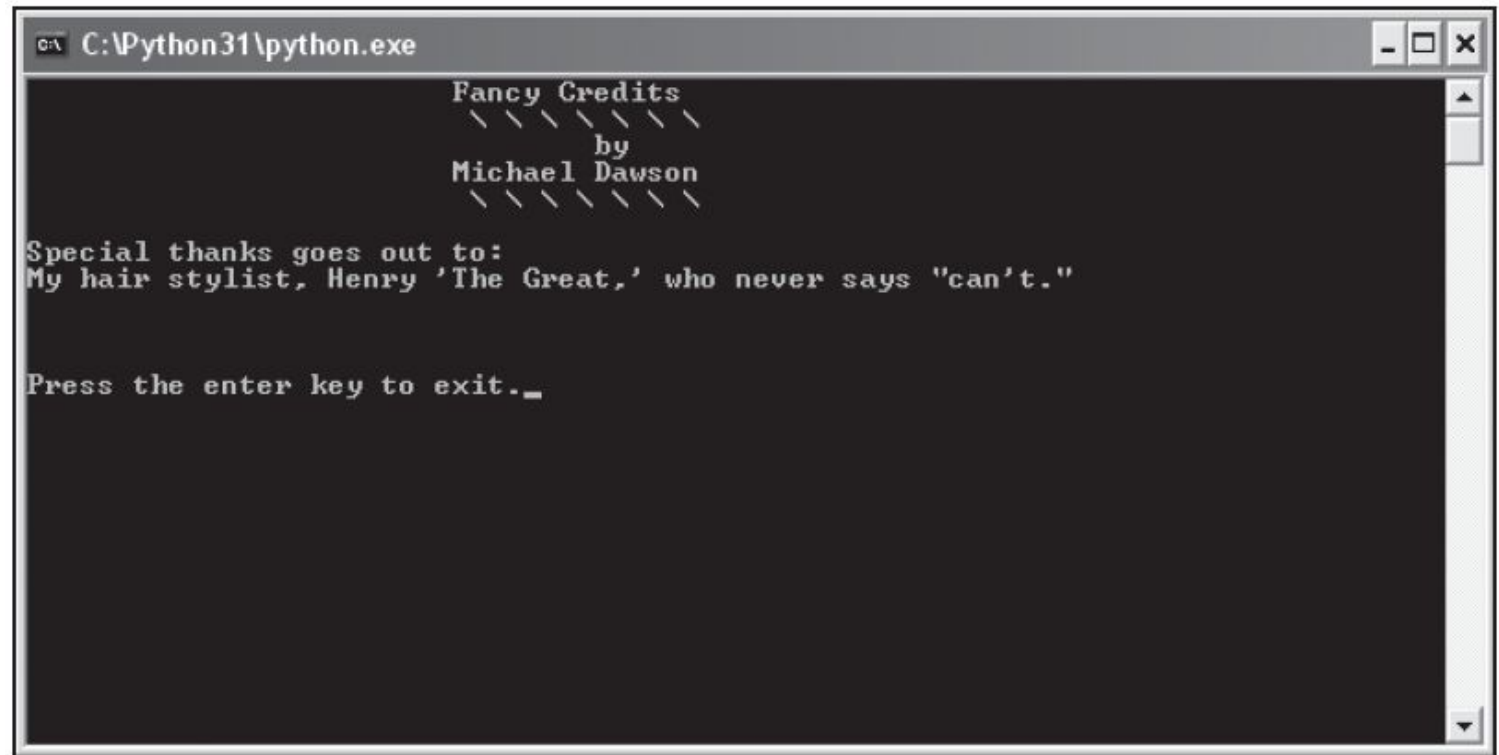
Sequence	Description
\\	Backslash. Prints one backslash.
\'	Single quote. Prints a single quote.
\"	Double quote. Prints a double quote.
\a	Bell. Sounds the system bell.
\n	Newline. Moves cursor to beginning of next line.
\t	Horizontal tab. Moves cursor forward one tab stop.



# Using Quotes With the Strings (Contd.)

## Introducing the Fancy Credits Program

- Just telling a player that the game is over, a program often displays credits, a list of all the people who worked so hard to make it a reality.
- Fancy Credits uses **escape sequences** to achieve some effects it just couldn't without them



```
C:\Python31\python.exe

          Fancy Credits
          \ \ \ \ \ \ \ \
              by
        Michael Dawson
          \ \ \ \ \ \ \ \

Special thanks goes out to:
My hair stylist, Henry 'The Great,' who never says "can't."

Press the enter key to exit._
```



# Using Quotes With the Strings (Contd.)

## Introducing the Fancy Credits Program: Example

```
# Fancy Credits
# Demonstrates escape sequences

print("\t\t\t Fancy Credits")

print("\t\t\t \\ \\ \\ \\ \\ \\ \\")
print("\t\t\t\t by")
print("\t\t\t Michael Dawson")
print("\t\t\t \\ \\ \\ \\ \\ \\ \\")

print("\n Special thanks goes out to:")
print("My hair stylist, Henry \'The Great,\' who never says \"can\'t.\"")

# sound the system bell
print("\a")

input("\n\n Press the enter key to exit.")
```

# Using Quotes With the Strings (Contd.)

## Moving Forward a Tab Stop

- Sometimes you'll want to set some text off from the left margin where it normally prints. In a word processor, you could use the Tab key
- With strings, you can use the escape sequence for a tab, `\t`

```
print("\t\t\t Fancy Crédits")
```

- Here, `\t` is used three times in a row. So, when the program prints the string, it prints three tabs and then Fancy Credits.
- This makes Fancy Credits look nearly cantered in the console window
- The **Tab sequences are good for setting off text**, as in this program, but **they're also perfect for arranging text into columns**

# Using Quotes With the Strings (Contd.)

## Printing a Backslash

- If you've thought ahead, you may be wondering how you can print a backslash if the computer always interprets a backslash as the beginning of an escape sequence
- Well, the solution is pretty simple: just use two backslashes in a row. Each of the following lines prints three tabs followed by seven backslashes (as a result of the seven \\ sequences), separated by spaces:

```
print("\t\t\t \\ \\ \\ \\ \\ \\ \\")  
print("\t\t\t \\ \\ \\ \\ \\ \\ \\")
```

# Using Quotes With the Strings (Contd.)

## Inserting a Newline

- One of the most useful sequences at your disposal is the newline sequence. It's represented by `\n`
- By using this sequence, python insert a newline character into your strings for a blank line where you need it. You can use a newline right at the beginning of a string to separate
- It from the text last printed. That's what I did in the line:

```
print("\n Special thanks goes out to:")
```

- The computer sees the `\n` sequence, prints a blank line, then prints Special thanks goes out to

# Using Quotes With the Strings (Contd.)

## Inserting a Quote

- Inserting a quote into a string, even the type of quote you use to bookend it, is simple. Just use the sequence `\'` for a single quote and `\"` for a double quote. They mean “put a quote here,” and won’t be mistaken by the computer as a marker for the end of your string.
- This is what I used to get both kinds of quotes in one line of text:

```
print("My hair stylist, Henry \'The Great,\' who never  
says \"can\'t.\")
```

- The pair of double quotes at both ends are the bookends, defining the string. To make the string easier to understand, look at it in parts:



# Using Quotes With the Strings (Contd.)

## Inserting a Quote

- `\'The Great\'` prints as `'The Great'`
- Each `\'` sequence is printed as a single quote
- `\"can\'t\"` prints as `"can't"`
- Both `\"` sequences print as double quotes
- The lone `\'` sequence prints as a single quote

# Using Quotes With the Strings (Contd.)

## Sounding the System Bell

- The below statement sounds the system bell of your computer. It does this through the escape sequence, `\a` which represents the system bell character
- Every time you print it, the bell rings
- You can print a string with just this sequence, as I have, or you can put it inside a longer string. You can even use the sequence several times to ring the bell more than once

```
print("\a")
```

- If I run it through IDLE, I get a little square box printed on my screen—not what I wanted. But if I run that same program directly from Windows, by double-clicking the program file icon, my computer's system bell rings just as I intended



The Silly Strings program prints several strings to the screen. The results are shown in Figure

***Go, Change the World***

# Concatenating and Repeating Strings

## Introducing the Silly Strings Program

```
# Silly Strings
# Demonstrates string concatenation and repetition
print("You can concatenate two " + "strings with the '+'
operator.")

print("\nThis string " + "may not " + "seem terr" + "ibly
impressive. " \ + "But what " + "you don't know" + " is
that\n" + "it's one real" \ + "l" + "y" + " long string,
created from the concatenation " \ + "of " + "twenty-two\n"
+ "different strings, broken across " \ + "six lines." + "
Now are you" + " impressed? " + "Okay,\n" \ + "this " +
"one " + "long" + " string is now over!")

print("\n If you really like a string, you can repeat it.
For example,")
```



# Concatenating and Repeating Strings

## Introducing the Silly Strings Program

```
print("who doesn't like pie? That's right, nobody. But if  
you really")  
print("like it, you should say it like you mean it:")  
print("Pie" * 10)  
input("\n\n Press the enter key to exit.")
```

# Concatenating and Repeating Strings

## Concatenating Strings

- Concatenating strings means **joining them together to create a whole new string**. A simple example is in the first print statement:

```
print("You can concatenate two " + "strings with the '+'  
operator.")
```

- The **+** operator joins the two strings, "You can concatenate two " and "strings with the '+' operator.", together to form a new, larger string
- It's like adding the strings together using the same symbol you've always used for adding numbers

# Concatenating and Repeating Strings

## Using the Line Continuation Character

- Generally, you write one statement per line. But you don't have to. **Python allows to stretch a single statement across multiple lines**
- All you have to do is use the **line-continuation character, \** (which is just a backslash), as I did in the preceding code. Put it anywhere you'd normally use a space (but not inside a string) to continue your statement on the next line. The computer will act as if it sees one long line of code.

```
print("\n This string " + "may not " + "seem terr" + "ibly impressive. " \
      + "But what " + "you don't know" + " is that\n" + "it's one real" \
      + "l" + "y" + " long string, created from the concatenation " \
      + "of " + "twenty-two\n" + "different strings, broken across " \
      + "six lines." + " Now are you" + " impressed? " + "Okay,\n" \
      + "this " + "one " + "long" + " string is now over!")
```

# Concatenating and Repeating Strings

## Repeating Strings

- The next new idea presented in the program is illustrated in the following line:

```
print("Pie" * 10)
```

- This line creates a new string, "PiePiePiePiePiePiePiePiePiePie", and prints it out. That's the string "Pie" repeated 10 times, by the way.



# Working with Numbers

- Computers let you represent information in other ways, too
- One of the most basic but most important ways is as numbers. Numbers are used in almost every program
- Whether you're writing a space shooter game or home finance package, you need to represent numbers some way. You've got high scores or checking account balances to work with, after all.
- Fortunately, Python has several different types of numbers to fit all of your game or application programming needs.

# Working with Numbers (Contd.)

## Introducing the Word Problems Program

The Word Problems program is just an amusing (hopefully) way to explore working with numbers.

```
C:\Python31\python.exe

If a 2000 pound pregnant hippo gives birth to a 100 pound calf,
but then eats 50 pounds of food, how much does she weigh?
Press the enter key to find out.
2000 - 100 + 50 = 1950

If an adventurer returns from a successful quest and buys each of
6 companions 3 bottles of ale, how many bottles are purchased?
Press the enter key to find out.
6 * 3 = 18

If a restaurant check comes to 19 dollars with tip, and you and
your friends split it evenly 4 ways, how much do you each throw in?
Press the enter key to find out.
19 / 4 = 4.75

If a group of 4 pirates finds a chest full of 107 gold coins, and
they divide the booty evenly, how many whole coins does each get?
Press the enter key to find out.
107 // 4 = 26

If that same group of 4 pirates evenly divides the chest full
of 107 gold coins, how many coins are left over?
Press the enter key to find out.
107 % 4 = 3

Press the enter key to exit._
```





# Working with Numbers (Contd.)

## Introducing the Word Problems Program

```
# Word Problems
# Demonstrates numbers and math

print("If a 2000 pound pregnant hippo gives birth to a 100 pound calf,")
print("but then eats 50 pounds of food, how much does she weigh?")
input("Press the enter key to find out.")
print("2000 - 100 + 50 =", 2000 - 100 + 50)
```

```
print("\n If an adventurer returns from a successful quest and buys each
of")
print("6 companions 3 bottles of ale, how many bottles are purchased?")
input("Press the enter key to find out.")
print("6 * 3 =", 6 * 3)
```



# Working with Numbers (Contd.)

## Introducing the Word Problems Program

```
print("\n If a restaurant check comes to 19 dollars with tip, and you  
and")  
print("your friends split it evenly 4 ways, how much do you each throw  
in?")  
input("Press the enter key to find out.")  
print("19 / 4 =", 19 / 4)
```

```
print("\n If a group of 4 pirates finds a chest full of 107 gold coins,  
and")  
print("they divide the booty evenly, how many whole coins does each  
get?")  
input("Press the enter key to find out.")  
print("107 // 4 =", 107 // 4)
```



# Working with Numbers (Contd.)

## Introducing the Word Problems Program

```
print("\nIf that same group of 4 pirates evenly divides the chest full")
print("of 107 gold coins, how many coins are left over?")
input("Press the enter key to find out.")
print("107 % 4 =", 107 % 4)

input("\n\nPress the enter key to exit.")
```

# Working with Numbers (Contd.)

## Understanding Numeric Types

- Python allows programmers to use several different types of numbers
- The two types used in this program, and probably the most common, are **integers and floating-point numbers (or floats)**
- Integers are **whole numbers—numbers with no fractional part**. Or, another way to think about them is that they can be written **without a decimal point**
- Floats are the numbers with **fractional part/ decimal part**
- The numbers **1, 27, -100, and 0** are all examples of integers. Floats are numbers with a decimal point, like **2.376, -99.1, and 1.0**.



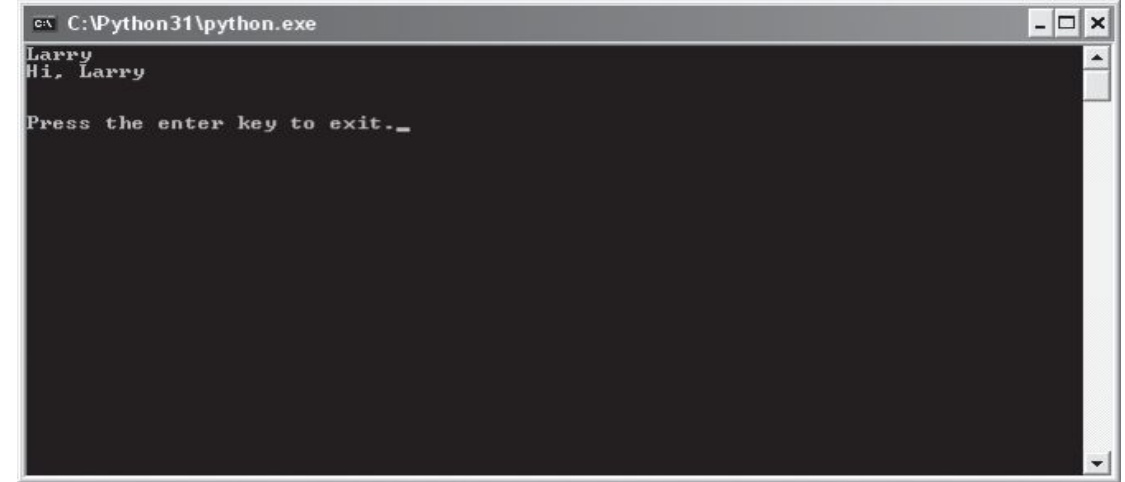
# Understanding Variables

Through variables, you can store and manipulate information, a fundamental aspect of programming. Python lets you create variables to organize and access this information.

# Understanding Variables (Contd.)

## Introducing the Greeter Program

```
# Greeter
# Demonstrates the use of a
variable
name = "Larry"
print(name)
print("Hi,", name)
input("\n\nPress the enter key
to exit.")
```

A screenshot of a Windows command prompt window titled "C:\Python31\python.exe". The window shows the output of the Greeter program: "Larry" on the first line, "Hi, Larry" on the second line, and "Press the enter key to exit.\_" on the third line. The cursor is positioned at the end of the third line, indicating that the program is waiting for user input to exit.

# Understanding Variables (Contd.)

## Creating Variables

- A **variable** provides a way to **label and access information**
- Instead of having to know exactly where in the **computer's memory some information is stored, you use a variable to get at it**
- It's kind of like calling your friend on his cell phone. You don't have to know where in the city your friend is to reach him. You just press a button and you get him.
- But, before you use a variable, you have to create it, as in the following line:

```
name = "Larry"
```

- This line is called an assignment statement. It creates a variable called name and assigns it a value so that it references the string "Larry".
- In general, assignment statements assign a value to a variable. If the variable doesn't already exist, as was the case with name, it's created, then assigned a value.

# Understanding Variables (Contd.)

## Using Variables

- Once a variable has been created, it refers to some value
- The convenience of a variable is that it can be used just like the value to which it refers.  
`print(name)`
- So the line prints the string "Larry" just like the statement `print("Larry")` does.  
`print("Hi, ", name)`
- The above line prints the string "Hi," followed by a space, followed by "Larry". In this case, I can use `name` instead of "Larry" with the same results.





# Understanding Variables (Contd.)

## Naming Variables

- For this program, I chose to call my variable name, but I could just as easily have used person, guy, or alpha7345690876, and the program would have run exactly the same
- There are only a few rules that you have to follow to create legal variable names. Create an illegal one and Python will let you know about it with an error
- The following are the two most important rules:
  1. A variable name can contain only numbers, letters, and underscores
  2. A variable name can't start with a number

# Understanding Variables (Contd.)

## Naming Variables

- The following are some guidelines that more experienced programmers follow for creating good variable names
  - **Choose Descriptive Names:** Variable names should be clear enough so that another programmer could look at the name and have a good idea what it represents.
  - **Be Consistent:** There are different schools of thought about how to write multiword variable names. Is it `high_score` or `highScore`? I use the underscore style. But it's not important which method you use, as long as you're consistent.
  - **Follow the Traditions of the Language:** Some naming conventions are just traditions. For example, in most languages (Python included) variable names start with a lowercase letter. Another tradition is to avoid using an underscore as the first character of your variable names. Names that begin with an underscore have special meaning in Python.
  - **Keep the length in check:** This may seem to go against the first guideline: Choose descriptive names. Isn't `personal_checking_account_balance` a great variable name? Maybe not. Long variable names can lead to problems. They can make statements hard to read. Plus, the longer the variable name, the greater the chance of a typo. As a guideline, try to keep your variable names under 15 characters.



# Getting User Input

- Instead of working with a predefined value, the computer lets the user enter his or her name and then uses it to say hi

```
# Personal Greeter
```

```
# Demonstrates getting user input
```

```
name = input("Hi.  What's your name ? \n")
```

```
print(name)
```

```
print("Hi,", name)
```

```
input("\n\n Press the enter key to exit.")
```

# Getting User Input (Contd.)

## Using the input() Function

```
name = input("Hi.  What's your name? ")
```

- Variable 'name' is created and a value is assigned to it, just like before. But this time, the right side of the assignment statement is a call to the function input().
- The input() function gets some text from the user. It takes a string argument that it uses to prompt the user for this text.
- In this case, the argument passed to input() is the string "Hi. What's your name? ". input() waits for the user to enter something. Once the user presses the Enter key, input() returns whatever the user typed as a string.
- To help visualize how this works, imagine that in the assignment statement, the call to input() is replaced with the string the user typed. Now of course, the code in the program doesn't change, but imagining the return value of a function in place of the call to that function helps drive home how you can use return values.



# Using String Methods

- Python has a rich set of tools for working with strings. One type of these tools is string methods
- String methods allow you to create new strings from old ones. You can do everything from the simple, such as create a string that's just an all-capital-letters version of the original, to the complex, such as create a new string that's the result of a series of intricate letter substitutions

# Using String Methods (Contd.)

- “I think there is a world market for maybe five computers.” This was made by then IBM chairman, Thomas Watson, in 1943

```
# Quotation Manipulation
# Demonstrates string methods
# quote from IBM Chairman, Thomas
Watson,
in 1943
quote = "I think there is a world
market for maybe five computers."

print("Original quote:")
print(quote)

print("\nIn uppercase:")
print(quote.upper())

print("\nIn lowercase:")
print(quote.lower())

print("\nAs a title:")
print(quote.title())

print("\nWith a minor replacement:")
print(quote.replace("five",
"millions of"))

print("\nOriginal quote is still:")
print(quote)

input("\n\nPress the enter key to
exit.")
```

# Using String Methods (Contd.)

Method	Description
<code>upper()</code>	Returns the uppercase version of the string.
<code>lower()</code>	Returns the lowercase version of the string.
<code>swapcase()</code>	Returns a new string where the case of each letter is switched. Uppercase becomes lowercase and lowercase becomes uppercase.
<code>capitalize()</code>	Returns a new string where the first letter is capitalized and the rest are lowercase.
<code>title()</code>	Returns a new string where the first letter of each word is capitalized and all others are lowercase.
<code>strip()</code>	Returns a string where all the white space (tabs, spaces, and newlines) at the beginning and end is removed.
<code>replace(<i>old</i>, <i>new</i> [,<i>max</i>])</code>	Returns a string where occurrences of the string <i>old</i> are replaced with the string <i>new</i> . The optional <i>max</i> limits the number of replacements.



# Using The Right Types

- You've used three different types so far: strings, integers, and floating-point numbers.
- It's important to know not only which data types are available to you, but how to work with them.
- If you don't, you might end up with programs that produce unintended results



# Using The Right Types (Contd.)

## Introducing the Trust Fund Buddy-Bad Program

- The program is supposed to calculate a grand total for monthly expenditures based on user input
- This grand total is meant to help those living beyond any reasonable means to stay within budget so they don't ever have to think about getting a real job
- Trust Fund Buddy—Bad doesn't work as the programmer intended
- The program produces unintended results but doesn't crash, it has a logical error
- Based on what you already know, you might be able to figure out what's happening by looking at the code



# Using The Right Types (Contd.)

## Introducing the Trust Fund Buddy-Bad Program

```
# Trust Fund Buddy - Bad
# Demonstrates a logical error
print("""
            Trust Fund Buddy

Totals your monthly spending so
that your trust fund doesn't run
out (and you're forced to get a
real job).

Please enter the requested, monthly
costs.  Since you're rich, ignore
pennies and use only dollar
amounts.

""")
```

```
car = input("Lamborghini Tune-Ups: ")
rent = input("Manhattan Apartment: ")
jet = input("Private Jet Rental: ")
gifts = input("Gifts: ")
food = input("Dining Out: ")
staff = input("Staff (butlers, chef,
              driver, assistant): ")
guru = input("Personal Guru and Coach: ")
games = input("Computer Games: ")
total = car + rent + jet + gifts + food +
        staff + guru + games

print("\n Grand Total:", total)
input("\n\n Press the enter key to exit.")
```

# Using The Right Types (Contd.)

## Tracking Down Logical Errors

- Logical errors can be the toughest bugs to fix. Since the program doesn't crash, you don't get the benefit of an error message to offer a clue. You have to observe the behavior of the program and investigate the code
- In this case, the huge number is clearly not the sum of all the numbers the user entered. But, by looking at the numbers, you can see that the grand total printed is a concatenation of all the numbers
- How did that happen? Well, if you remember, the `input()` function returns a string. So each "number" the user enters is treated like a string. Which means that each variable in the program has a string value associated with it.
- So, the line is not adding numbers. It's concatenating strings!

```
total = car + rent + jet + gifts + food + staff + guru + games
```

# Using The Right Types (Contd.)

## Converting Values

- The solution to the Trust Fund Buddy—Bad program is to convert the string values returned by input() to numeric ones. Since the program works with whole dollar amounts, it makes sense to convert each string to an integer before working with it

```
# Trust Fund Buddy - Good
# Demonstrates type conversion
print(
    """
        Trust Fund Buddy

Totals your monthly spending so that your trust fund doesn't run out
(and you're forced to get a real job).

Please enter the requested, monthly costs.  Since you're rich, ignore
pennies and use only dollar amounts.

    """)
```

# Using The Right Types (Contd.)

## Converting Values

```
car = input("Lamborghini Tune-Ups: ")
car = int(car)

rent = int(input("Manhattan Apartment: "))
jet = int(input("Private Jet Rental: "))
gifts = int(input("Gifts: "))
food = int(input("Dining Out: "))
staff = int(input("Staff (butlers, chef, driver, assistant): "))
guru = int(input("Personal Guru and Coach: ") )
games = int(input("Computer Games: "))

total = car + rent + jet + gifts + food + staff + guru + games

print("\nGrand Total:", total)
input("\n\nPress the enter key to exit.")
```

# Using The Right Types (Contd.)

## Converting Values

- Notice that the assignments are done in just one line. That's because the two function calls, `input()` and `int()`, are nested
- Nesting function calls means putting one inside the other. This is perfectly fine as long as the return values of the inner function can be used as argument values by the outer function. Here, the return value of `input()` is a string, and a string is a perfectly acceptable type for `int()` to convert.

Converting from one type to another type is called Type Conversion

Function	Description	Example	Returns
<code>float(x)</code>	Returns a floating-point value by converting <i>x</i>	<code>float("10.0")</code>	10.0
<code>int(x)</code>	Returns an integer value by converting <i>x</i>	<code>int("10")</code>	10
<code>str(x)</code>	Returns a string value by converting <i>x</i>	<code>str(10)</code>	'10'

# Using Augmented Assignment Operators

- Let's say you want to know the yearly amount the user spends on food. To calculate and assign the yearly amount, you could use the line

```
food = food * 52
```

- This line multiplies the value of food by 52 and then assigns the result back to food. You could accomplish the same thing with the following line:

```
food *= 52
```

- `*=` is an augmented assignment operator. It also multiplies the value of food by 52 and then assigns the result back to food, but it's shorter than the first version
- Since assigning a new value to a variable based on its original value is something that happens a lot in programming, these operators provide a nice shortcut to a common task.

# Using Augmented Assignment Operators

- Useful Augment Assignment Operators

Operator	Example	Is Equivalent To
<code>*=</code>	<code>x *= 5</code>	<code>x = x * 5</code>
<code>/=</code>	<code>x /= 5</code>	<code>x = x / 5</code>
<code>%=</code>	<code>x %= 5</code>	<code>x = x % 5</code>
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>x -= 5</code>	<code>x = x - 5</code>



