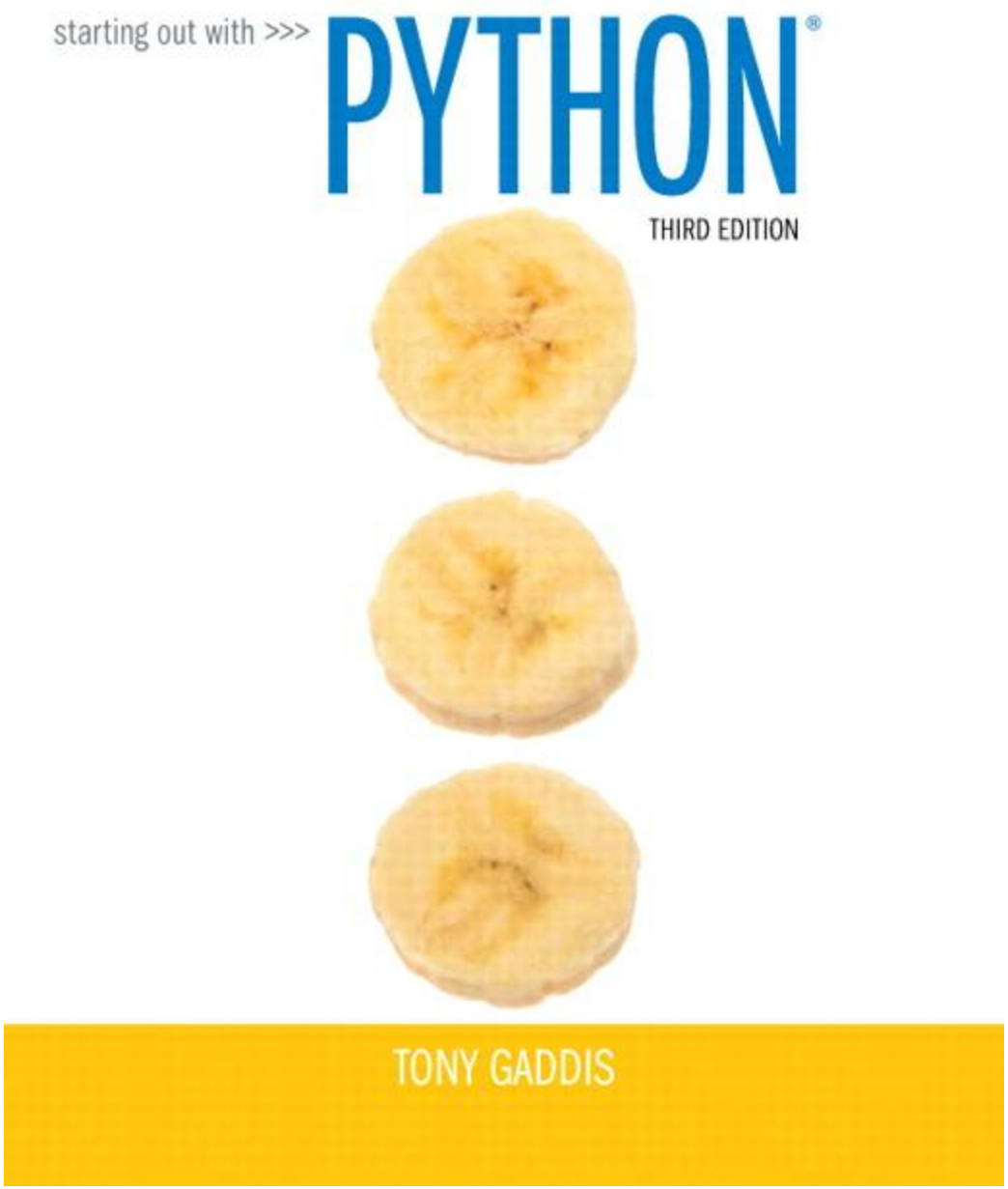
CHAPTER 2

Input, Processing, and Output



Topics

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

Fundamentals of Computer Science: Algorithms and Information Processing

- **Computer science focuses on a broad set of interrelated ideas**
  - Two of the most basic ones are:
    - Algorithms
    - Information processing

Designing a Program

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

Designing a Program

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

Designing a Program

- **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:** a step-by-step solution to solve a problem using a finite amount of time and space

Pseudocode

- **Pseudocode:**
  - Informal high-level description of the operating principle of a computer program or other algorithm.
  - 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

Algorithm

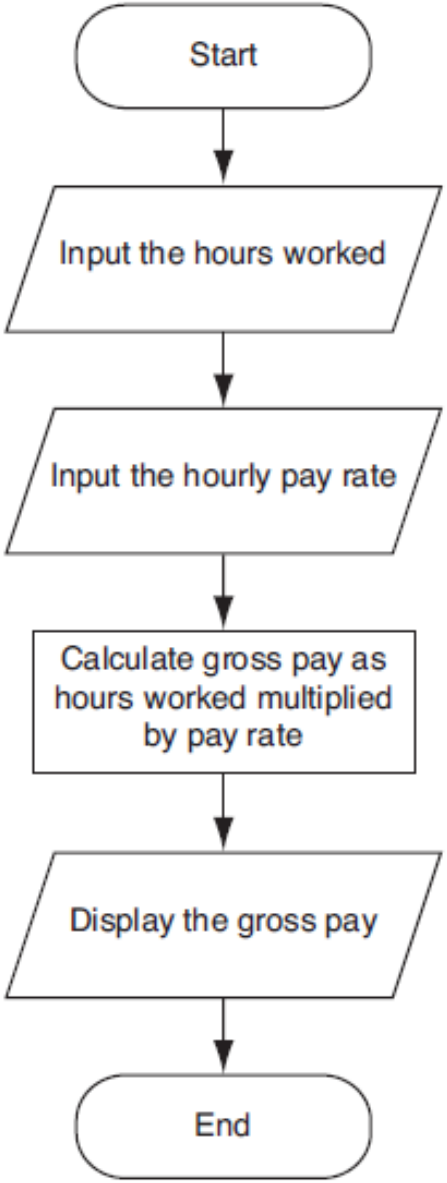
- Sequence of steps that describes each of these computational processes is called an algorithm
- Features of an algorithm:
  - Consists of a finite number of instructions
  - Each individual instruction is well defined
  - Describes a process that eventually halts after arriving at a solution to a problem
  - Solves a general class of problems

Algorithm

- Steps for subtracting two numbers:
  1. Write down the numbers, with larger number above smaller one, digits column-aligned from right
  2. Start with rightmost column of digits and work your way left through the various columns
  3. Write down difference between the digits in the current column of digits, borrowing a 1 from the top number's next column to the left if necessary
  4. If there is no next column to the left, stop
     - Otherwise, move to column to the left; go to Step 3
- The computing agent is a human being

Algorithm

- Add two numbers
  - Ask user to enter first integer
  - Ask user to enter second integer
  - Add two integers
  - Print result
- Calculate Area and Perimeter of a Right Triangle
  - Ask user to enter base
  - Ask user to enter height
  - Calculate hypotenuse using Pythagorean
  - Multiply base * height * 1/2 to get area
  - Add base + height + hypotenuse to get perimeter
  - Print results



Flowcharts

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

Input, Processing, and Output

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

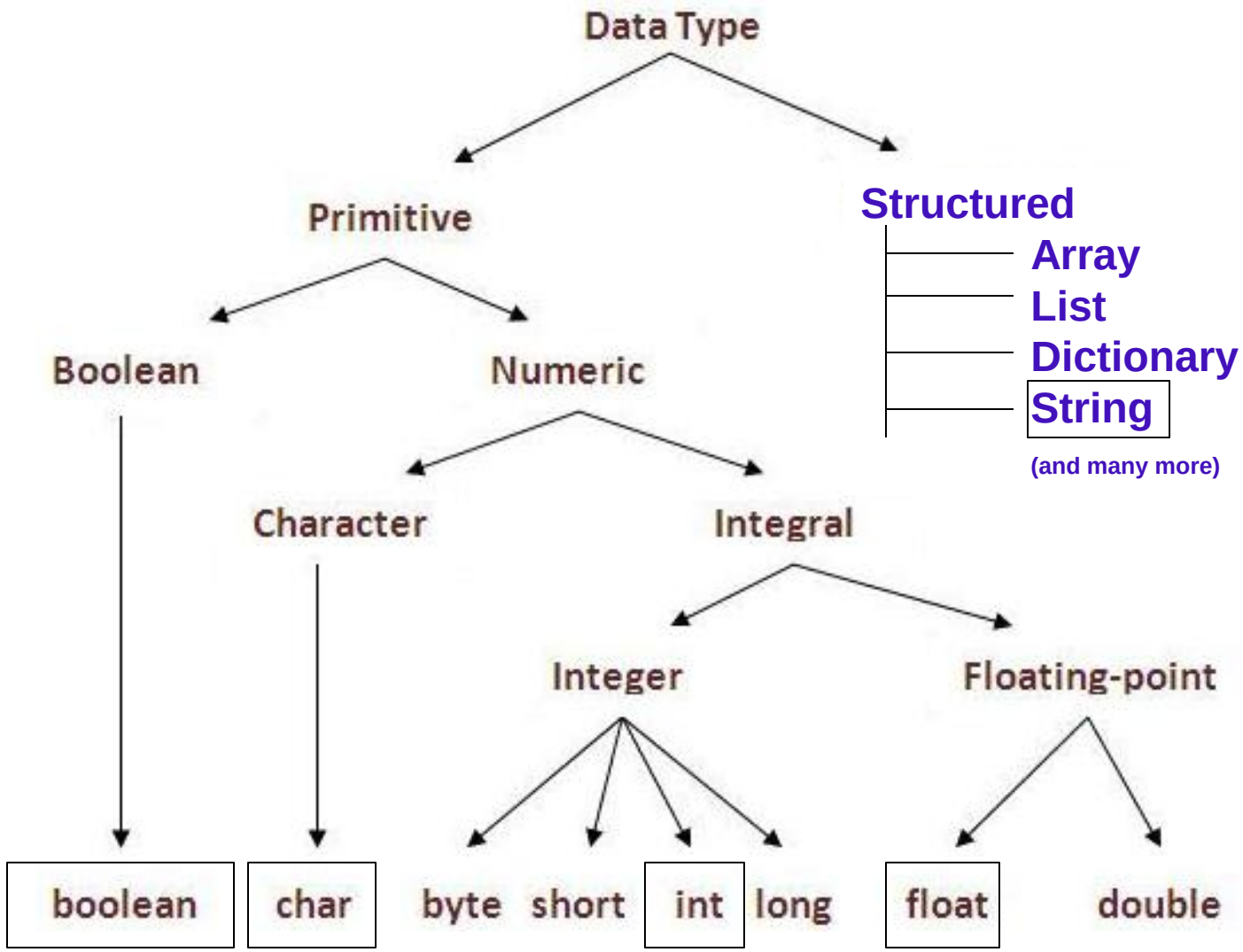
Variables

- **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
  - Makes it easy to remember and use later in program
- **Assignment statement: used to create a variable and make it reference data**
  - General format is `variable = expression`

    `variable name = value to hold`
  - Assignment operator: the equal sign (=)

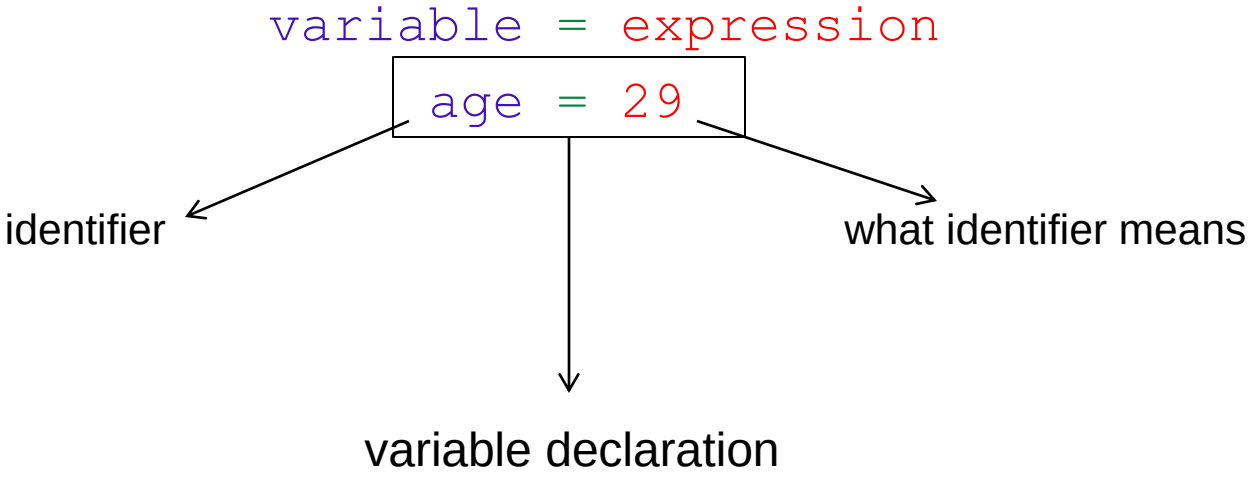
```
age = 29          - integer (whole number)
length = 23.6     - float (decimal number, real number)
initial = 'H'     - character (single letter,, single quote)
name = "Python"   - string (whole word, double quotes)
contd = True      - Boolean holds only True or False
TAX = 0.08        - CONSTANT (unchangeable once set)
```

Variables


Data Type
Primitive
Structured
Array
List
Dictionary
String
(and many more)
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Variables

- **Variable declaration**:
  - Declaration specifies properties of an identifier: it declares what a word (identifier) means. For every variable:



Variables in Memory
(conceptual)

- When a variable is declared, enough memory to hold a value of that type is allocated for it at an unused memory location. This is the address of the variable.

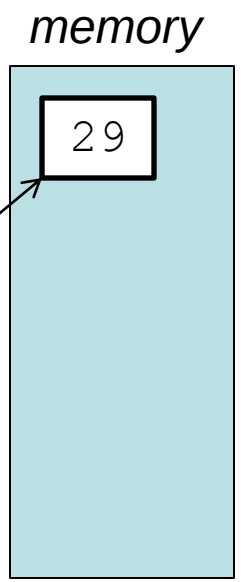
1. Variable declaration

```
age = 29
```

2. Space allocation in memory (enough to hold variable value)

3. Value stored to memory

4. Address:
variable name: `age`
memory location: `33965280`



Variable/Function Naming Rules

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

Variable/Function Naming Rules

BAD	GOOD	
print, list, sum, str, if	myList	Variable names cannot be a Python keywords (purple color, orange color)
first Name	firstName	Variable names cannot contain spaces
1age &age	age _age	First character must be a letter or an underscore
	f_Name _2011Date	After first character may use letters, digits, or underscores
	name Name NAME	Variable names are case sensitive
x y	height length	Variable name should reflect its use i.e. be DESCRIPTIVE
	firstName first_name	Compound variable names 1. lowerCamelCase (I mostly use) 2. snake_case: use _ (underscore) between words Do not mix snake_case and lowerCamelCase
main() getSpeed()		Do not use function names

Python Style

- Style Guide for Python Code

http://legacy.python.org/dev/peps/pep-0008/#code-lay-out

- Google Style

https://google-styleguide.googlecode.com/svn/trunk/pyguide.html

- C99 standard (now called C11)

http://www.open-std.org/jtc1/sc22/wg14/www/standards

Variables

- **In assignment statement, variable receiving value must be on left side**

```
herAge = 29          29 = herAge
```

- **You can only use a varaible is a value is assigne to it**

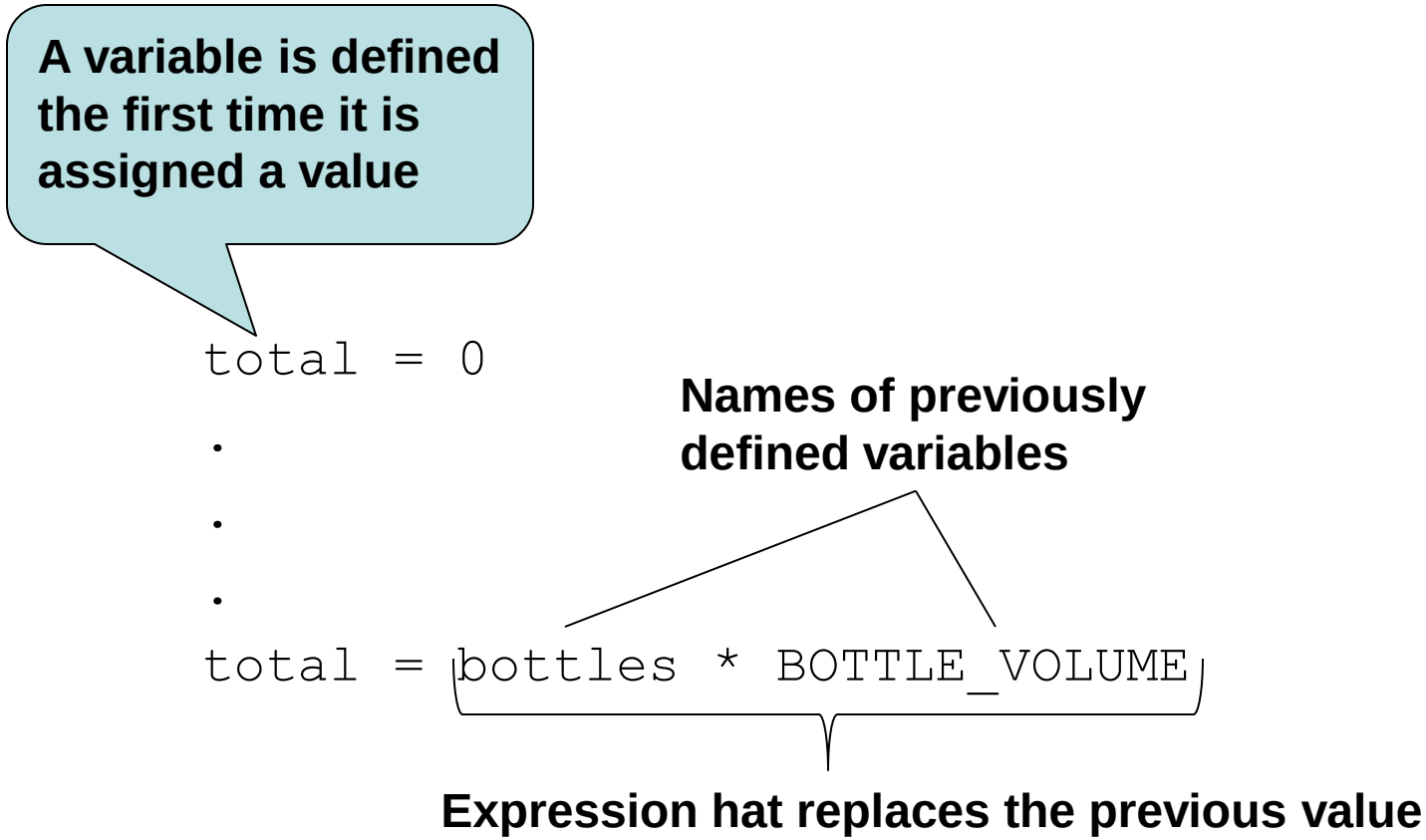
```
hisAge = herAge      myAge = someAge
```

- **A variable can be passed as an argument to a function**
  - Variable name <u>should not</u> be enclosed in quote marks

```
print(herAge)        print("herAge")
output: 29           output: herAge
```

- **<u>You can only use a variable if some value is assigned to it first</u>**

Variables


A variable is defined the first time it is assigned a value
total = 0
.
.
.
total = bottles * BOTTLE_VOLUME
Names of previously defined variables
Expression hat replaces the previous value

Variables

A variable is defined the first time it is assigned a value

```
total = 0
.
.
.
total = bottles * BOTTLE_VOLUME
.
.
.
total = total + cans * CAN_VOLUME
```

Names of previously defined variables

Expression hat replaces the previous value

The name can occur on both sides

Variables

- **Common error: Undefined Variable**
  - You try to use a variable that has not been defined
  - 2 most common reasons:
    - You did not define a variable but try to use it

```
total = 0
result = result + total
```

    - Misspelling

```
volume = 0
totalVolume = 0
totalVolume = totalvolume + volume
```

Variable Reassignment

- **Variables can reference different values while program is running**

  result = 29 …

  result = 31 …

- **A variable can refer to item of any type**
  - Python specific
  - Variable that has been assigned to one type can be reassigned to another type

  type = 29

  STRONGLY DISCOURAGED …

  type = “Python” 

IN THIS CLASS YOU WILL NOT DO THIS. Once you declare a variable to be of certain type, it MUST stay that type … no exception and guaranteed points deduction.

Strings and Characters

- **String: sequence of characters**
  - In python, it may be enclosed in single (') or double (") quote marks
    - Enclosed string can contain both single and double quotes and can have multiple lines

```
"Python"   'This is Python'   "G"   'G'   "#%$@"
```

- **Character: single character**

IMPORTANT: use " " for string and ' ' for character

```
String          language = "Python"
Character       init = 'g'
```

IN THIS CLASS YOU WILL use single quotes '' for characters and double quotes "" for strings… no exception and guaranteed points deduction if you fail to follow

Numeric Data Types

- **Data types: categorize value in memory**
  - int (integer) 1 2 3 4 5 6 …

    32bit number: −2,147,483,648 to 2,147,483,647
  - long (integer)

    64bit number: −9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
  - float (rational number) 2.3 1.0056

    32bit number: precision 24 numbers after dot
  - double (rational number) same as float

    64Bit number: precision 53 numbers after dot

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

variablesDemo.py

Numeric Operators

- An arithmetic expression consists of operands and operators combined in a manner that is already familiar to you from learning algebra

OPERATOR	MEANING	SYNTAX
-	Negation	-a
**	Exponentiation	a ** b
*	Multiplication	a * b
/	Division	a / b
//	Quotient (Integer division)	a // b
%	Remainder or modulus	a % b
+	Addition	a + b
-	Subtraction	a - b

[TABLE 2.6] Arithmetic operators

Numeric Operators

```
a = 21
b = 10
c = 0

c = a + b
c = a - b
c = a * b

c = a / b
```

/ operator performs floating point division
returns a float 25/5 = 5.0

```
a = 10
b = 6
c = a % b
```

10%6 = 4 so 10/6 = 1
then you do 10-(6*1) = 4

```
a = 2
b = 3
c = a**b
```

```
a = 10
b = 5
c = a//b
```

// operator performs integer division
returns integer 36//9 = 4

Operator Precedence and Grouping with Parentheses

- Precedence rules:

**	Exponentiation (raise to the power)
+ -	Unary plus and minus (+x and -x)
* / % //	Multiply, divide, modulo and integer division
+ -	Addition and subtraction
=	Assignment operators

  - With two exceptions, operations of equal precedence are left associative, so they are evaluated from left to right
    - ** and = are right associative
  - You can use () to change the order of evaluation

Operator Precedence and Grouping with Parentheses

EXPRESSION	EVALUATION	VALUE
`5 + 3 * 2`	`5 + 6`	11
`(5 + 3) * 2`	`8 * 2`	16
`6 % 2`	`0`	0
`2 * 3 ** 2`	`2 * 9`	18
`-3 ** 2`	`-(3 ** 2)`	-9
`-(3) ** 2`	`9`	9
`2 ** 3 ** 2`	`2 ** 9`	512
`(2 ** 3) ** 2`	`8 ** 2`	64
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

Mixed-Type Expressions and Data Type Conversion

- int * int

```
5 * 4 = 20
```

- float * float

```
6.4 * 2.1 = 13.44
```

- int * float

```
4 * 2.1 = 8.4
```

- Type conversion of `float` to `int` causes truncation of fractional part

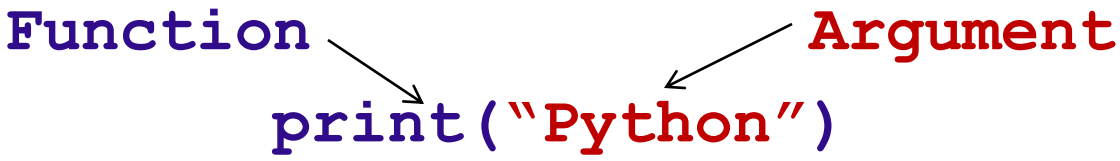
```
result = int(4 * 2.1)
```

- Variables can be used as operands

```
result = set // subCount
```

Displaying Output with the `print` Function

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



Parameters vs Arguments

```
Function          Argument
print("Python")
```

A **function parameter** is a variable declared in the prototype or declaration of a function:

```
foo(x) // declaration -- x is a parameter
{
}
```

An **argument** is the value that is passed to the function in place of a parameter:

```
foo(6); // 6 is the argument passed to parameter x
foo(y+1); // the value of y+1 is the argument passed to parameter x
```

Displaying Multiple Items with the `print` Function

- **Python allows one to display multiple arguments with a single call to `print`**
  - Items are separated by commas when passed as arguments
  - Arguments displayed in the order they are passed to the function
  - Variables should not be enclosed in quote marks
  - Items are automatically separated by a space when displayed on screen

```
hisAge = 29
print("He is ", hisAge, " years old.")
output: He is 29 years old.
```

Escape Characters

The newline character **\n** is called an escape sequence.

Backslash notation	Description
\a	Bell or alert
\b	Backspace
\cx	Control-x
\C-x	Control-x
\e	Escape
\f	Formfeed
\M-\C-x	Meta-Control-x
\n	Newline
\nnn	Octal notation, where n is in the range 0.7
\r	Carriage return
\s	Space
\t	Tab
\v	Vertical tab

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

```
#this is a comment
print("Python")
```

printDemo.py

Reading Input from Keyboard

- **Most programs need to read input from the user**
- **Built-in `input` function reads input from keyboard**
  - **ALWAYS returns input 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

```
variable    function    argument
firstName = input("Enter your name: ")
```

Reading Numbers with `input`

- **`input` function always returns a string**
- **Built-in functions convert between data types**
  - **`int(`*`item`*`)`** converts *`item`* to an `int`
  - **`float(`*`item`*`)`** converts *`item`* to a `float`
  - Type conversion only works if item is valid numeric value, otherwise, throws exception

```
name = input("What is your name? ")
age = int(input("What is your age? "))
income = float(input("What is your income? "))
```

- 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 an error

inputDemo.py

Mixed-Mode Arithmetic and Type Conversions

CONVERSION FUNCTION	EXAMPLE USE	VALUE RETURNED
`int(<a number or a string>)`	`int(3.77)`	`3`
	`int("33")`	`33`
`float(<a number or a string>)`	`float(22)`	`22.0`
`str(<any value>)`	`str(99)`	`'99'`

[TABLE 2.8] Type conversion functions

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

Breaking Long Statements into Multiple Lines

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

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

More About Data Output

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

Formatting Numbers

- **Can format display of numbers on screen using built-in `format` function**
  - Two arguments:
    - Numeric value to be formatted
    - Format specifier

```
print(5000/12)                  416.666666667

print(format(5000/12, '.2f'))   416.67
```

mathDemo.py

Summary

- Algorithms can be compiled and executed even in Python because they are written in Psudocode
- print function takes 1 or more arguments
- bookName = “Starting out with Python”
- symbol = ‘g’
- A comment starts with # and needs to be the first item on each line that has a comment

```
# comment
# comment
print(“python”)
```

Summary

- newEntry = 2.4

- passToFunction = "7/11/11"
- 45 = sideA

- time = "12:37"

  print(time) 12:37 print("time") time

- int
- print
- 123Side
- _Name
- variable names should be descriptive

Summary

- **input function takes only 1 argument**

- **by default input returns a string and to make the input an integer is has to be converted using int() function**

```
age = int(input('What is your age? '))
```

- **Variable at input is expecting a float but the user enters a string. What happens?**

Summary

- **What is 5**3?**
- **Is this correct? and what does it return?**
  - 5.3/4
  - 16/4
  - 16//4
  - 16.4//4.4
  - 5*6
  - 4.4*6
  - 4.4*6.7
  - int(10.2/2)
  - int(5*2.5)
  - float(5*6)

Practice

A customer in a store is purchasing 5 items. Write a program that asks for the price of each item and then displays the subtotal of the sale, the amount of sales tax, and the total. Assume sales tax is 8%.

Algorithm:

1. **Prompt to enter the cost of 5 items**
2. **Calculate subtotal of 5 items without tax**
3. **Calculate tax on subtotal**
4. **Calculate total cost of 5 items including tax**
5. **Print formatted result**

Summary

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