Introduction To Programming

MAD 102

Objectives

- Explain computer programming and what it includes
- Describe how data and instructions are stored in the computer
- Explain the difference between hardware and software, with examples of each
- Summarize the basic operating functions of the central processing unit, memory, and storage drives
- Describe an algorithm and name some tools for developing one
- Describe methods for testing programs

Objectives

- Name the two main data types a programmer uses and give an example of each
- Define and contrast input and output
- Explain the need for and use of program comments
- Describe the input-output-processing method

What is computer programming

- Computer programming is :

  “The process of formulating instructions to operate a digital computer, an electronic device that can receive, process, store and send data”

  - These instructions and data are represented as binary digits
- Bits: are binary digits – either zero and ones (0 and 1)
  - They are the building blocks of a digital computer

Types of Data

- **Data** is stored in digital form and is the raw information processed by computers
  - There are three ***general*** categories of data types (types of data are stored differently for reasons of compatibility and processing efficiency)
    - Numeric data – values to be used in ***mathematical*** calculations
    - Text data – letters, punctuation marks, that can be displayed and printed
    - Raw binary data – image, video and sound files

Base 10

- Humans work well with a base 10 system
  - The decimal system

  - For example – the number 535
    - 5 hundreds
    - 3 tens
    - 5 single units

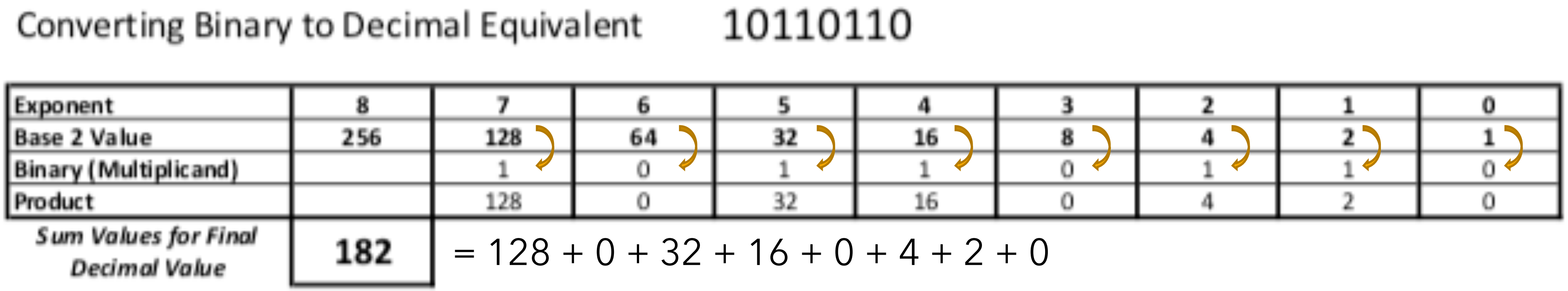
    - Put them together we get 535

Binary

- Computers work with base 2 – binary
- There are only two options for this a **1** and **0**
  - Voltage on 1
  - Voltage off 0
- In binary – numbers increase by a factor of 2: 1,2,4,8,16,32,64,128,256,…
- Each of the digits in binary is a **bit**
- Eight bits make a **byte**
- Four bits is a nibble
- Computer memory is made up of either **32bits** or **64bits** in length

CONVERSION

- CONVERTING 10110110 TO DECIMAL



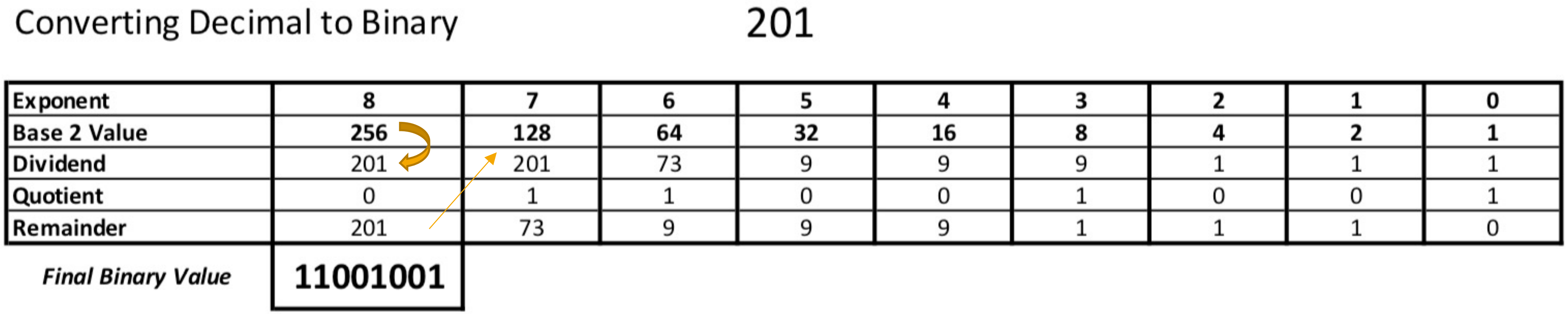
Converting Binary to Decimal Equivalent 10110110

Exponent	8	7	6	5	4	3	2	1	0
Base 2 Value	256	128	64	32	16	8	4	2	1
Binary (Multiplicand)		1	0	1	1	0	1	1	0
Product		128	0	32	16	0	4	2	0
Sum Values for Final Decimal Value	**182**								

= 128 + 0 + 32 + 16 + 0 + 4 + 2 + 0

CONVERSION

- CONVERTING DECIMAL 201 TO BINARY EQUIVALENT



Converting Decimal to Binary 201

Exponent	8	7	6	5	4	3	2	1	0
Base 2 Value	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Dividend	201	201	73	9	9	9	1	1	1
Quotient	0	1	1	0	0	1	0	0	1
Remainder	201	73	9	9	9	1	1	1	0

Final Binary Value **11001001**

Hexadecimal

- Also called hex is base 16
- Because we only have 10 digits to use - the remaining 6 values are the letters a-f
- Each hexadecimal digit can represent 4 binary digits
  - 1111 is equivalent to the hexadecimal f
  - This allows for a more compact and clear representation of values
  - The number 4,294,967,295
  - In binary is 11111111111111111111111111111111 (32-bits)
  - In hexadecimal it is ffffffff

What is computer programming

- Binary instructions are known as machine instructions
- Binary is difficult to comprehend, programs called assemblers were created to translate instructions for us
  - Assembly language instructions are converted to machine instructions
- High-level languages allow programmers to write formulas which more closely matched how people think
  - Compilers translate these high-level language programs into executable programs

What is computer programming

- Programming languages (C#, Java, Swift) or Scripting languages (JavaScript or Python) – are understandable to people but must be ***translated*** to the computer’s machine language so that they can be run

What is computer programming

- Data and information are often used interchangeably
  - Data is considered as ***unprocessed*** and ***unorganized*** facts, names, and numbers
  - Information is considered the ***useful*** results of the computer's *processing* and organization
- Software
  - A digital representation of instructions on the computer – the computer programs
- Hardware
  - The computer and its related equipment

What is computer programming

- The processor is an integrated circuit that contains the CPU (Central Processing Unit) which performs the processing activity. It has two functional parts:
  - **Control Unit** – performs the following functions:
    - Fetch – gets the next instruction from system memory
    - Decode – get any data required by the instruction and find the address of the next instruction
    - Execute – perform required actions, which might involve sending data and instructions to the ALU
    - Store – write results to main memory or send them to an output device
  - **Arithmetic Logic Unit (ALU)** –performs like a calculator
    - Compares two values and returns one of three results – the values are equal, the first value is greater than the second, the second value is greater than the first

Memory

- Memory is the other main component of computer processing
- Two common types:
    - **Random Access Memory (RAM)** – also called main memory
        - It is the temporary storage place for instructions and data while the computer is running
        - It is erased when the computer is shut down – this is why it is known as volatile
        - Data is not being fetched in a predetermined order
    - **Read-Only Memory (ROM)** – data can be read, or accessed but not changed
        - Contains instructions for the system to perform a self-test as it powers up and loads the OS into main memory
        - It is persistent – it is not erased when the computer is shut down
- Long-term storage is handled with disk drives, flash drives and other storage media

Character-coding systems

- Character-coding systems
  - American standard code for information interchange (ASCII)
    - Eight bits used to represent each character
    - Covered 128 specified characters – numbers 0-9, letters a-z and A-Z and basic punctuations
    - Was the most commonly used character encoding on WWW until Dec 2007 when it was surpassed by UTF-8
    - http://www.ascii-code.com/
  - Unicode
    - Sixteen bits are used (first eight match ASCII) capable of representing 65,536 characters (compared to 256 for ASCII)
    - Developed to address needs of non-English language alphabets
    - http://unicode-table.com/en/

Input and output

- Input to the program is gathered in two ways:
    - Information is entered while interacting with the program
    - Information is retrieved from a file or database
- Information is sent from the computer and is called output. There are two kinds of output:
    - To the user's screen (soft copy)
    - To a printer (hard copy)
- A prompt tells the user what information is required for input and is displayed on the screen

Program Logic

- For a computer to understand what you want it to do – you must provide it with instructions
- The instructions must be provided in a specific sequence
- The instructions must be complete
- The instructions must be definitive and free of errors

Program Logic

- Programs are like recipes
- Start at the beginning and follow a sequence
  - Can’t start cooking before you have done some prep work
- All the steps are related to the task at hand
  - Are not going to include ”Call to make an appointment for an oil change” as this is not a step that would be required for any recipe

Program logic

- A simple program follows the sequence structure:
- Sequence structure
  - Steps meant to be performed ***in order***
  - Statements are performed **without** any conditions
  - May be grouped into sections and commented

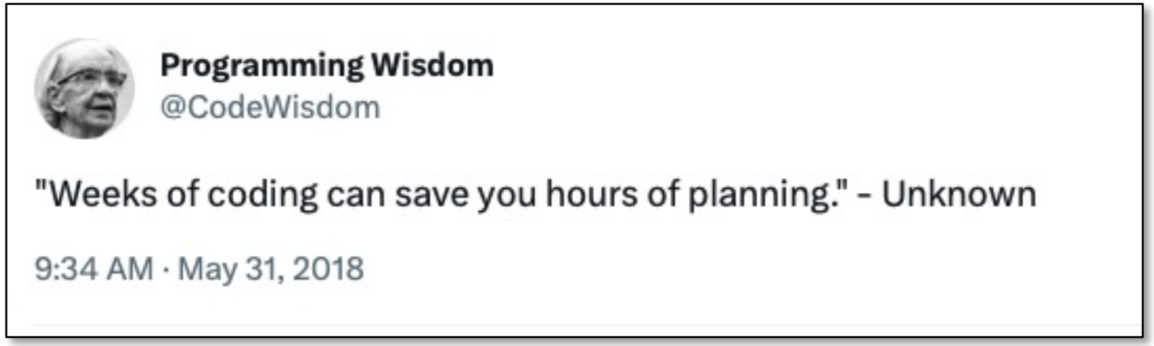
Algorithms

- Creating an algorithm is the logic of problem solving
  - First understand the problem
  - Formulate the steps used (the algorithm)
  - Characteristics of algorithms
    - Correct – Provide a *satisfactory* solution to the problem
    - Efficient – uses suitable programming tools *without* wasting time and resources
    - Easy to understand – can be explained in ordinary language

Program Development Cycle

- You don’t just sit down and start writing code – this leads to errors, re-writes and bloated code
- Follows a series of steps

1. Understand the problem
2. Plan the logic
3. Write the code
4. Test the code
5. Deploy your code
6. Maintain your code



Programming Wisdom
@CodeWisdom

"Weeks of coding can save you hours of planning." - Unknown

9:34 AM · May 31, 2018

Understand the problem

- Your program is to meet the needs of someone
    - Print the total amount owing for a purchase
    - Determine how many bottles of item X are required
    - Did the user correctly guess the secret number
    - Record the users running speed for the current day
    - Log a new favourite restaurant

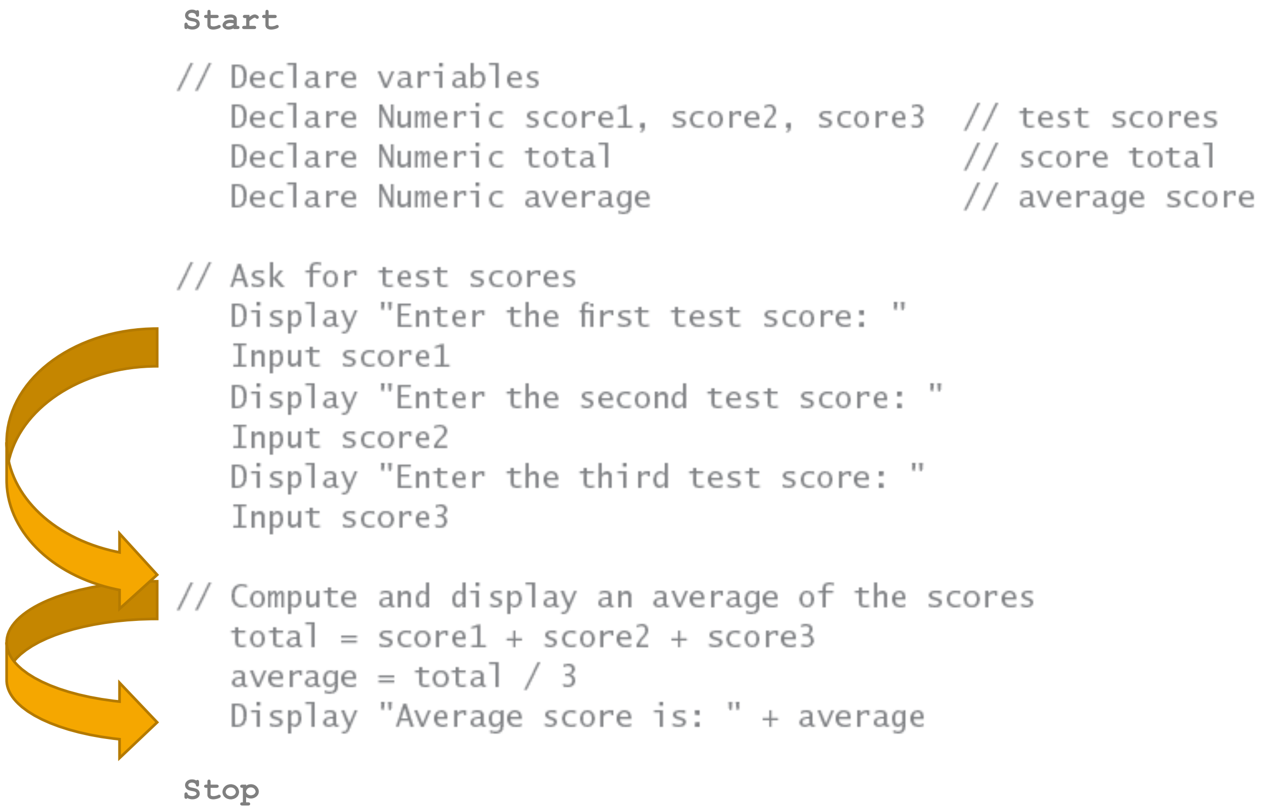
Plan the logic

- Develop the algorithm – the sequence of steps to solve the problem
- You will be developing a plan in three steps:
    - Get some information from the user (input some data)
    - Process the information (calculations)
    - Provide the solutions (output some information)
- Often called **desk-checking** – this is the process of walking through your steps on paper
    - Popular methods are pseudocode and flowcharting

Pseudocode

- An English like representation of the steps required to solve a problem
- Has a loose set of rules to ensure that a program can be easily converted from it into any computer language
- Meant to be easy for anyone to understand and be able to “read” what the program does

Pseudocode



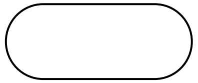
```
Start

// Declare variables
   Declare Numeric score1, score2, score3  // test scores
   Declare Numeric total                   // score total
   Declare Numeric average                 // average score

// Ask for test scores
   Display "Enter the first test score: "
   Input score1
   Display "Enter the second test score: "
   Input score2
   Display "Enter the third test score: "
   Input score3

// Compute and display an average of the scores
   total = score1 + score2 + score3
   average = total / 3
   Display "Average score is: " + average

Stop
```

Representing control structures with flowcharts

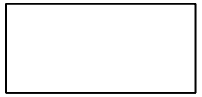
- A flowchart is a graphical tool for expressing an algorithm’s logic that convey the same information as pseudocode
- It can represent a single module or an entire program

Flowchart components

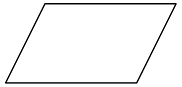
- **Terminal symbols** (ovals) that mark the beginning and ending of a flowchart



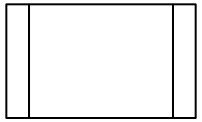
- **Process symbols** (rectangles) for the variable declarations or assignment statements



- **Input/output symbols** (parallelograms) for display statements, prompts and input statements

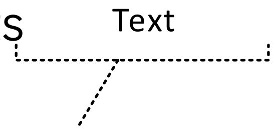


- **Module symbols** (rectangles with stripes) used to call a module or function, with the module definition in a separate flowchart section

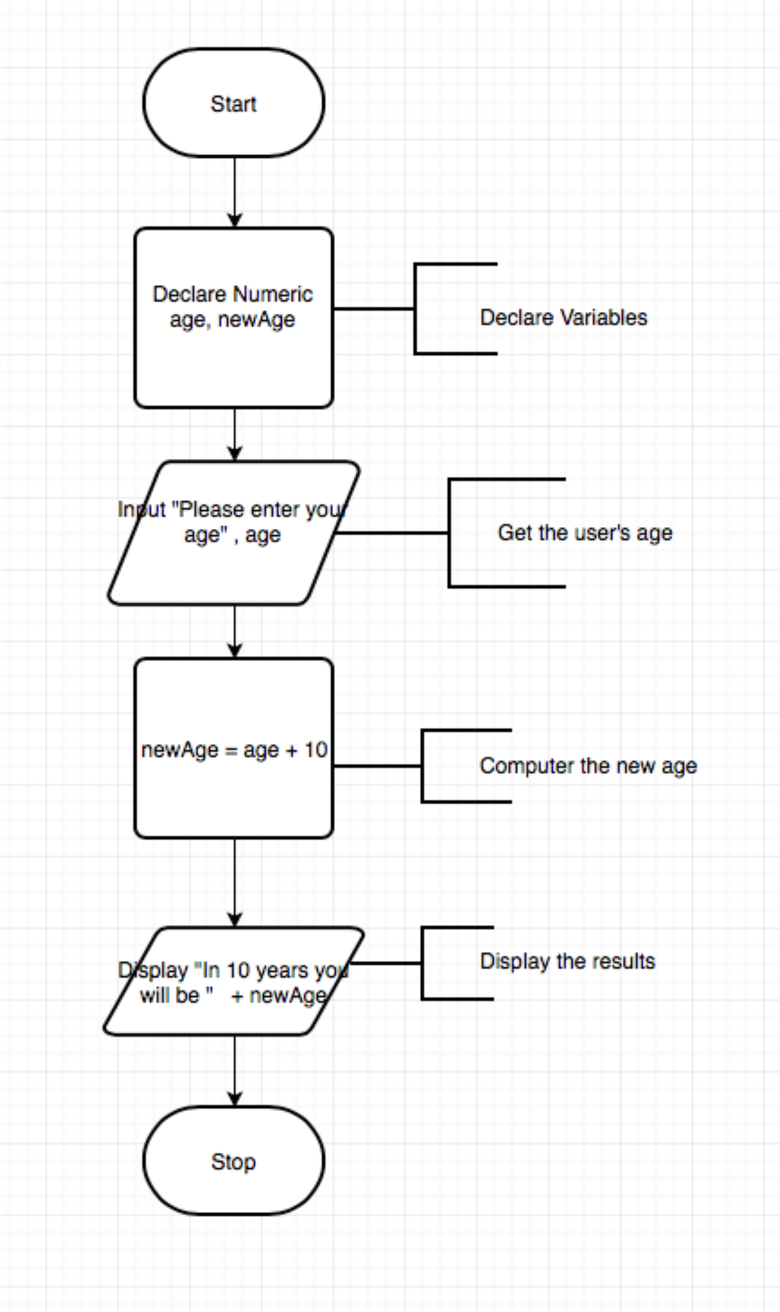


- **Flowlines** (lines with arrowheads) for connecting other symbols

- **Annotation boxes** (open-sided boxes) for comments

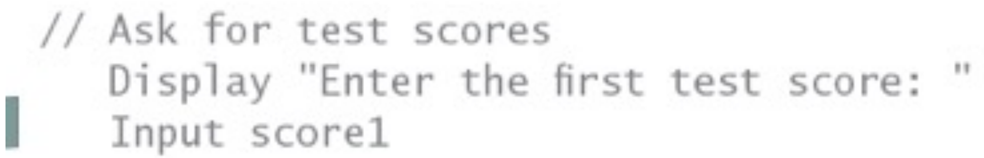


Sample flowchart


Start
Declare Numeric age, newAge
Declare Variables
Input "Please enter your age" , age
Get the user's age
newAge = age + 10
Computer the new age
Display "In 10 years you will be " + newAge
Display the results
Stop

Write the code

- Convert your plan into a working algorithm
- If you planned the details – it is ***easier*** work to create a working algorithm
- Not every step is a step for step conversion.



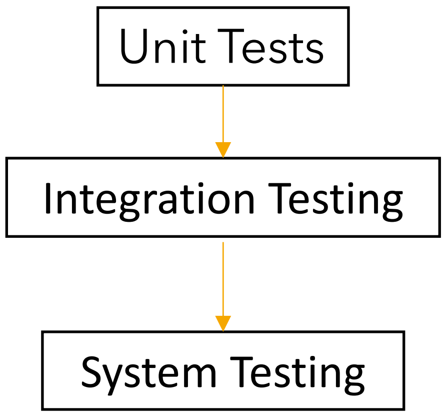
```
// Ask for test scores
   Display "Enter the first test score: "
   Input score1
```

  - Getting the user to enter some information and then storing that information may required several lines of actual code.

*** There is no perfect language for solving every problem – use the language that is best suited for the problem at hand ***

Testing the code

- Testing can be performed on different levels:
  - Small sections of program code called snippets
    - Do the last couple lines produce the result I want?
  - Task-oriented modules
    - Does this add to numbers correctly?
  - Interaction between modules
    - Does this code pass the correct information to the next module to run?
  - Complete programs
    - Does this program run from start to finish correctly?



Errors

- There are three main categories of errors:
  - Syntax errors – are violations of language rules
    - Prevents your program from being compiled
  - Logic errors – incorrect instructions
    - Assigning *instead* of comparing
    - Incorrect calculations
  - Runtime errors – errors that are not know until a program is run
    - Identified with an abrupt unintended termination of your code – or crash

Datasets

- Datasets can be created to test programs
- Can include data that test around the boundaries of a specified value
  - Testing if a number is less than 10?
    - Check 0-9, 10, 11-20, -ve numbers, decimal numbers, letters, nothing
- Used to check valid **and** invalid data

Deploy the code

- Let people use it
- Congratulations – you have successfully solved a problem!

Maintain the code

- Maintaining the code may entail
  - Fixing small problems
  - Making changes to values (tax amount changed)
  - Adding new features

- Program maintenance is the development phase where you appreciate the effort that went into keeping programs simple, easy to understand
  - Proper naming
  - Following conventions
  - Clear commenting

Programming with Python

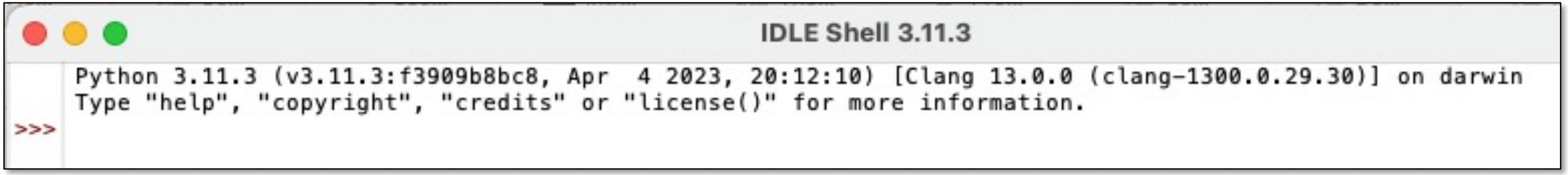
- Python is an object-oriented programming language
- Python is a **scripting** language and is interpreted
    - A script is a program whose instructions are executed by another program called an **interpreter**
    - Does not require a compiler allowing for fast edit-test-debug cycles
- Emphasizes readability

Programming with Python

- Python is open-source
    - User community helps to define the language and creating new interpreters
- Current version 3.x
    - not backwards compatible with older versions

Python Interpreter

- This is a program that executes code written in Python
- IDLE (Integrated Development and Learning Environment) allows us to type and run in a GUI (Graphical User Interface) window



```
Python 3.11.3 (v3.11.3:f3909b8bc8, Apr  4 2023, 20:12:10) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Python Interpreter

- The characters >>> indicate a prompt
  - Python is waiting to work with the code we enter after the prompt
- Basic environment for editing and running programs
- An IDE (Integrated Development Environment) is a better choice for more sophisticated programs
  - Enjoy benefits of indentation, code completion, debugging, etc.

Variables

- Programming languages use memory locations to store information. Keeping track of memory locations via their binary address is difficult. Programming languages make use of variables for this.
- Variables are ***programmer designated*** names for memory locations
- Memory locations are used to store values (numeric or text data) that can change ← vary

Variables

- Variables can be **declared** (informs the computer that you want a specific name to represent a specific value) and they can be **assigned** values
    - The **identifier** is the word used for your variable – it identifies a value
    - The assignment process involves an **assignment statement** where the variable name is followed by a ***single*** equals sign and the value to be assigned
    - = is the assignment operator

Assignment , not equality

- One of the common errors for new programmers is the = operator
  - This is the assignment operator – assigns the item on the right to the item on the left of it.
  - We often use the words “equals” when discussing our code – but it is not equality as we know from mathematics

Variables

- Depending on the programming language, you may have to declare the type of data that the variable represents. (***Weakly typed*** languages do not require a type – ***Strongly typed*** do *)
- String variables can store any text data – single characters, collection of characters (including numeric) and an “empty string”

```
firstName = 'Luke'
```

Variables and named constants

- Or if you wanted to store a person’s age you would declare that the variable age would hold numeric information (number)
  - Numeric variables can store any type of number – with or without decimal places
  - Numeric values are values that you may perform mathematical calculations on

```
age = 24
```

Declaring strings vs. numeric

- All string values must be wrapped in quotes
  - Can be double quotes or single quotes

```
firstName = 'Luke'
lastName = "Skywalker"
```

- All numeric values are **not** wrapped in quotes

```
age = 24
speed = 65.3
```

Variable Naming

- Variable names must comply with language rules
  - Most languages follow these rules:
    - Can include letters, digits, underscores and hyphens
      - Dog, dog, d0g d_o_g ✓
    - Can NOT begin with a digit
      - 1dog
    - Can NOT contain spaces
      - one person
    - Can NOT contain keywords – words or phrases that are part of the language itself.
      - For example – var decimal: Double = 5.65

🚫

Variable naming

- Names should be easy to read and understand
- Early languages limited to max of six characters or digits – modern languages allow an almost unlimited length
- Camel casing is commonly used for naming – the first letter of each word is capitalized (not the first word for variables – more on this later)
- Names are case sensitive – so firstname is different from firstName and is different from FirstName
- Remember – what you write is not going to be read/modified by only you. It is important that names that convey meaning

Variable naming

	✓	🚫
Purpose of Variable	**Good Names / Good Descriptors**	**Bad Names/ Bad Descriptors**
Running total of checks written to date	runningTotal, checkTotal	written, checks
Velocity of a bullet train	velocity, trainVelocity, velocityInMph	velt, v, train
Current date	currentDate, todaysDate	current, cd, date
Lines per page	linesPerPage	lpp, lines

Variable naming

- Variable Name Length – although name length is almost unlimited, it does not mean that you should take full advantage of it. Use the *Goldilocks approach*

Length	**Variable Names**
Too Long	numberOfPeopleOnTheOlympicTeam, numberOfSeatsInTheStadium, maximumNumberOfPointsInModernOlympics
Too Short	n, numP n, ns, nosits m, maxPoints, max
Just Right	numTeamMembers, teamMemberCount numSeatsInStadium, seatCount teamPointsMax, pointsRecord

Objects

- When Python interpreter runs, it creates an object when executing the lines of code
    - Once the objects are no longer needed, they are automatically deleted from memory and thrown away
    - This process is called **garbage collection** and frees memory space

Objects

- Each Python object has three properties
    - Value –
        - The data associated with the object
    - Type - helps determine what behaviour it can support
        - Adding or concatenating
        - This can be accessed by running a function called **type()**
    - Identity
        - Each object has a unique identifier (the memory address where it is stored)
        - Can be accessed using a function called **id()**

Objects

- The type of an object determines its **mutability**
  - This is whether it can change or not
- String and Integers are **immutable** – they cannot be changed
  - Changing the values with new assignment statements results in new objects

Numeric Types

- There are two main numeric types – integers and floating-point numbers
  - Integer are whole numbers – they do not have a decimal component
  - Floating-point numbers have a decimal component.
    - The position of the decimal can “float” to different locations

Numeric Types

- Large numbers can be expressed using scientific notation

```
power = 1.21e9                 1210000000.0
```

Arithmetic Expressions

- Expressions are combination of items – **variables**, **operators**, **literals**, etc.
- Literals are a specific value
  - For example the name ‘Luke’ or the number 24
- Operators are symbols that perform specific operations
  - For example – the assignment operator (=) which we use to assign a literal to a variable

Numeric Operations

- Numeric calculations use arithmetic operators
- + for addition
- - for subtraction
- * for multiplication
- / for division
- % for modulus (also known as the remainder operator)
- ** the exponent operator

Order of Operations

- In most languages operations are carried out in this order:
  - Exponential operations performed before any basic arithmetic operation
  - Multiplication and division
  - Addition and subtraction
  - **BEDMAS**

Increasing and Decreasing Numbers

- **Incrementing** (increasing) or **decrementing** (decreasing) a numeric value by a set amount is a common programming practice
- For example – you want to increase a numeric variable by 1

```
firstNumericalValue = 1

firstNumericalValue = firstNumericalValue + 1

print(firstNumericalValue)
```

Compound Operators

- Compound operators work as short-hand ways for updating variables

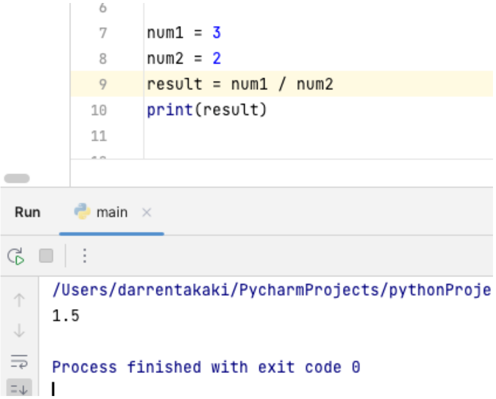
```
firstNumericalValue = 1

firstNumericalValue += 1

print(firstNumericalValue)
```

Compound Operators

- Addition +=
- Subtraction -=
- Multiplication *=
- Division /=
- Modulo %=

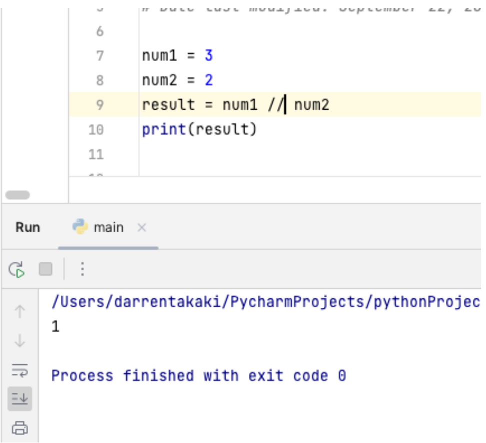
Floor Division Operator

- The operator / will divide two numbers
  - It returns a floating point number
- The floor division operator //, returns the integer value minus decimal values



```
num1 = 3
num2 = 2
result = num1 / num2
print(result)
```

```
/Users/darrentakaki/PycharmProjects/pythonProje
1.5

Process finished with exit code 0
```



```
# Date last modified: September 22, 20

num1 = 3
num2 = 2
result = num1 // num2
print(result)
```

```
/Users/darrentakaki/PycharmProjects/pythonProjec
1

Process finished with exit code 0
```

Modules

- Code is usually written in files called **scripts**
- The script is then passed to the Python interpreter in order for it to run (**execute**) the code
- These files are **modules** and can be used by other modules or scripts
- The module is accessed by an **import** statement

Modules

- Using modules makes management of larger programs easier
- The Python Standard Library is a collection of pre-installed modules

Modules

- The objects defined in a module are accessed using dot notation.
  - Use the name of the module, followed by a dot (.), followed by the object you want to access.

Math Module

- The math module supports advanced math operations beyond the basic math operators
- **import math** will add it
- It contains a series of **functions**
  - Blocks of code that are executed by **calling** (asking it to run) it
  - Some functions just run – other require additional information for them to run
  - The additional information provided to a function are **arguments**

Random Numbers

- The **random module** provides methods for generating random numbers
- The random method - random() – returns a random floating point number in the range of 0 (***inclusive***) to 1(***exclusive***)
  - This means any value from 0 to 1, including 0 but not including 1

Random Range

- The randrange method – randrange() – generates integers within a range
  - The single positive argument returns values from 0 (inclusive) to that number – 1 (exclusive)
  - For example – random.randrange(3) would return 0,1,2 - but not 3

Defined ranges

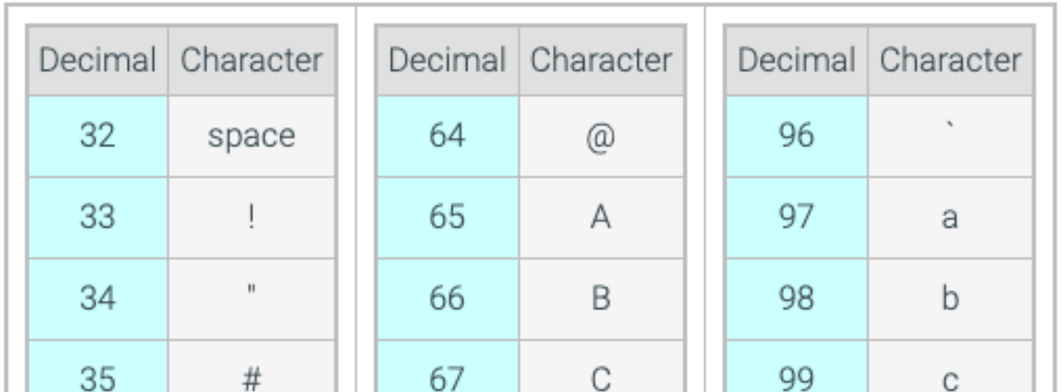
- Providing a min and max value will set a range
- randrange(minValue, maxValue) will provide from the min(including) to one less than the max
- randint(minValue, maxValue) will provide from the min (including) to the max(including)

Generating the random number

- The random method makes use of a seed – in this case, an integer based on the current time – to help generate a random number
  - Set the seed using the **seed** method to reproduce ‘random’ numbers

Unicode

- All characters are represented by a unique number (code point)
- Python uses Unicode to represent these characters
- Unicode allows for over 1 million code points



Decimal	Character	Decimal	Character	Decimal	Character
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c

Unicode

- The ord() method can be used to convert a specific character to its Unicode encoded integer value

```
number = ord("!")
print(number)               33
```

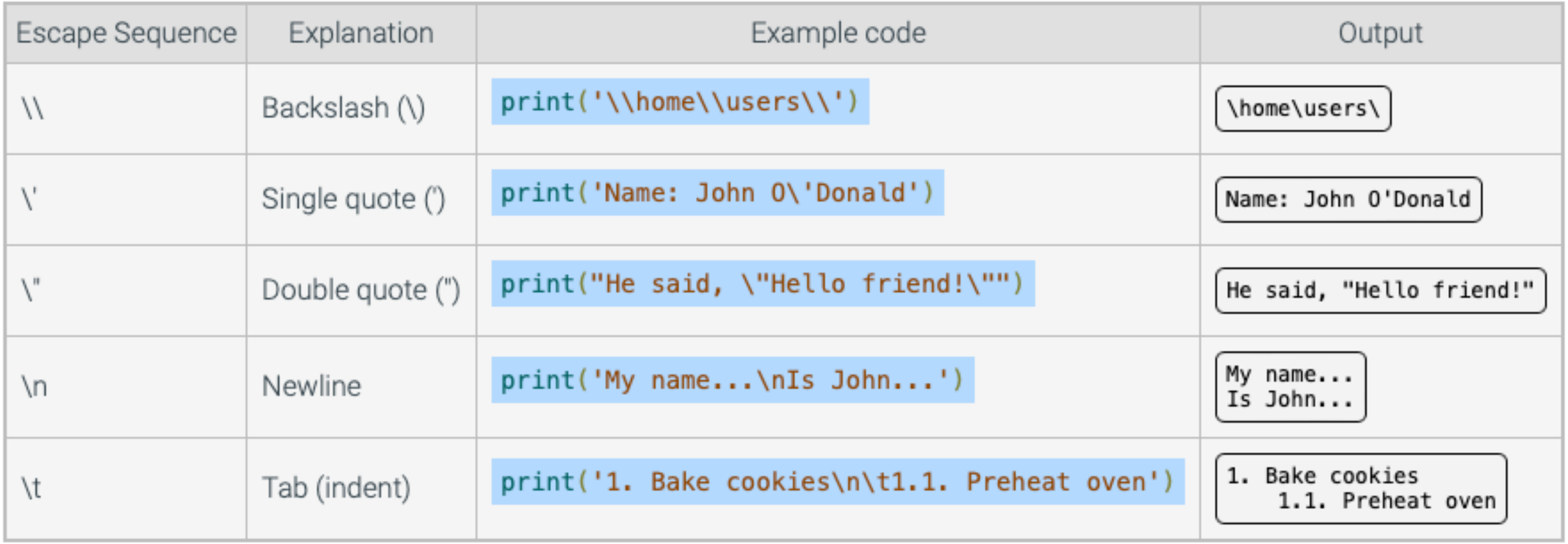
- The chr() will convert the encoded integer into the character

```
character = chr(number)
print(character)                  !
```

Escape Sequences

- There a times when a ‘ or a “ is part of your string value, not the end of the string
- To get the interpreter to ignore the characters, we escape them using a / (backslash)

Escape Sequences



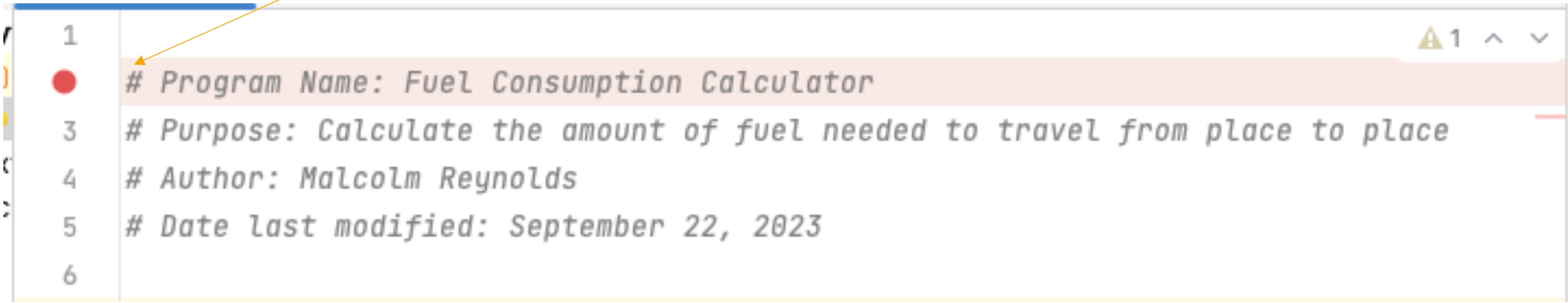
Escape Sequence	Explanation	Example code	Output
\\	Backslash (\)	print('\\home\\users\\')	\home\users\
\'	Single quote (')	print('Name: John O\'Donald')	Name: John O'Donald
\"	Double quote (")	print("He said, \"Hello friend!\"")	He said, "Hello friend!"
\n	Newline	print('My name...\nIs John...')	My name... Is John...
\t	Tab (indent)	print('1. Bake cookies\n\t1.1. Preheat oven')	1. Bake cookies 1.1. Preheat oven

Program comments

- Comments are ways to explain how your program operates and basic information about it
  - A non-programmer should be able to look at your code and see what it is doing
- Every program should have consistent basic documentation that includes the program’s name and purpose, who wrote it, and the date it was written and modified.
- They are used to describe what a section of code does
- They are written in plain language
- Languages differ in how they are written - // at the beginning of a line

Comments

Indicates a single line comment



```
# Program Name: Fuel Consumption Calculator
# Purpose: Calculate the amount of fuel needed to travel from place to place
# Author: Malcolm Reynolds
# Date last modified: September 22, 2023
```