


Software = Computer Programs


- ❖ **Program:** Set of step-by-step instructions that direct the computer to do a task and produce correct.
- ❖ **Programming Language:** Set of rules that instructs the computer what operations to perform.
- ❖ **Programming is Fun!**
 - ◆ Simpler than human language
 - ◆ Usually about 50 keywords
 - ◆ Automates structured tasks
 - ◆ Very logical without emotions
 - ◆ Computer becomes your slave



Copyright © 2019 R.M. Laurie 1

People = End Users & Programmers

- ❖ **End User's = IFSM201, IFSM300**
 - ◆ Utilize computer resources and applications
- ❖ **Programmers = CMIS102 or CMST385**
 - ◆ **Analyze** problem and create solution algorithm
 - ◆ **Code** the solution algorithm into a specific programming language
 - ◆ **Verify** program works using known test data
 - ◆ High demand career field with high pay
- ❖ **UMUC coding courses**
 - ◆ Programming: CMIS102 → CMIS141 → CMIS242
 - ◆ Web Design: IFSM201 → CMST385 → CMST386 → CMST388



Copyright © 2019 R.M. Laurie 2

Programming Language Generations

- ❖ **1st = Machine Language**
 - ◆ Actual bits that CPU processes
- ❖ **2nd = Assembly Language**
 - ◆ Each assembly instruction corresponds to one machine code instruction
 - ◆ Requires an **assembler** to convert assembly source code to machine code
- ❖ **3rd = High-level Language**
 - ◆ Uses human words for keywords
 - ◆ Abstract and general purpose
 - ◆ Requires a **compiler** or **interpreter** to run
 - ◆ Compiles for different CPU's
- ❖ **4rd = Application Language**
 - ◆ SQL for databases

Copyright © 2019 R.M. Laurie 3

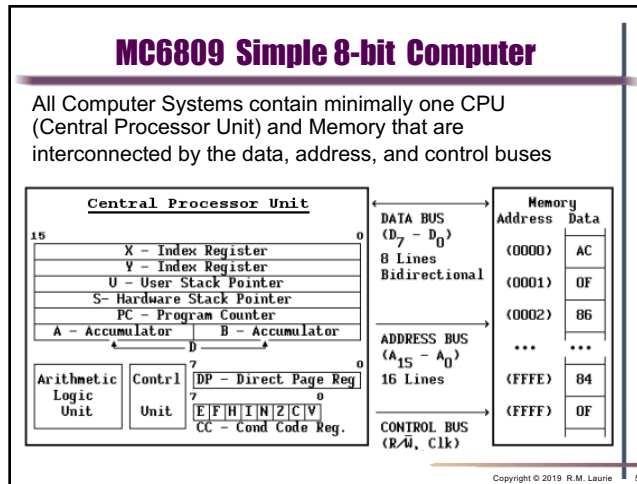
First Generation: Machine Language

- ❖ **Lowest level programming language because it represents data and program instructions as binary 0/1. Generally, hexadecimal is used for human interaction.**
- ❖ **All programming languages are eventually converted into machine language.**
- ❖ **Will be run on only one type of CPU**

0000	
...	
D000	86
D001	42
D002	8B
D003	0C
D004	B7
D005	D1
D006	00
D007	BB
D008	D1
D009	10
D00A	B7
D00B	D1
D00C	01
...	
FFFF	

Copyright © 2019 R.M. Laurie 4

Slide Set 8 - Programming Introduction



Second Generation: Assembly Language

Assembly Program is assembled to machine code by Assembler

Address	Instructions	Data	Assembly Language Program
D000	86	12	LDA #\$12
D002	8B	0C	ADDA #\$0C
D004	B7	D100	STA \$D100
D007	BB	D110	ADDA \$D110
D00A	B7	D101	STA \$D101
D00D	8B	1E	ADDA #\$1E
D00F	B7	D01B	BCC \$D019
D012	86	00	LDA #\$00
D014	B7	D110	STA \$D110
D017	23	D007	BRA \$D007
D01A	3F		SWI

Copyright © 2019 R.M. Laurie 6

3rd Generation: High-Level Language Python

Line # Source code written in Python syntax

<https://youtu.be/lilIW-ovx0Y>

```

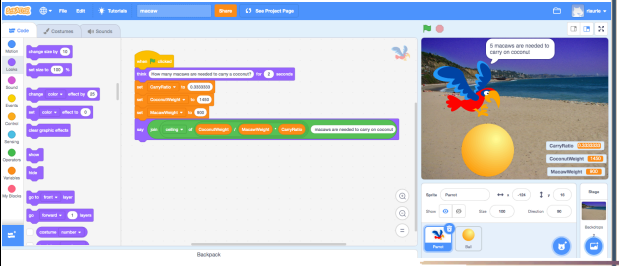
1. # How many Macaws are needed to carry a coconut?
2. # Author: Robert Laurie
3. import math #Required to use math.ceil function
4. fCarryRatio = 1/3 #float Carry ratio is 1/3 of weight
5. nCoconutWt = 1450 #int Coconut weight = 1450 grams
6. nMacawWt = 900 #int Macaw bird weight = 900 grams
7. fCarryWt = nMacawWt * fCarryRatio #float Calculation
8. # help(math.ceil) can get info about a function
9. nMacawQty = math.ceil(nCoconutWt/fCarryWt)
10. # Display results with description string
11. print(nMacawQty, "macaws needed to carry one coconut")
    
```

Copyright © 2019 R.M. Laurie 7

- ### Historical Development of HLL
- ❖ **FORTRAN**: 1957, Engineering and science applications.
 - ❖ **COBOL**: 1959, Developed for business applications.
 - ❖ **BASIC**: 1965, Interpreted language, Easy to program with DOS.
 - ❖ **C**: 1975, **Procedural Oriented** (verbs), efficient structure and fast
 - ❖ **C++**: 1985, Added several keywords to C so that could be used as an **Object Oriented Programming** language, **OOP** focuses on object (nouns) rather than tasks (verbs).
 - ❖ **Java**: 1993, Pseudo-Compiled language generates **bytecode** which runs on any **Java Virtual Machine** to achieve **OS and CPU Independence**; Developed as a simplified **OOP** language that supports **Networks, Security, and Multithreaded** for multitasking.
 - ❖ **JavaScript**: 1995, Interpreted Language, utilizes JS interpreter in web browser; **Object-based**; Similar syntax to Java and C++.
 - ❖ **Python**: 2005, Interpreted, **Procedural or OOP**, **Good 4 beginners**.
 - ❖ **Ruby**: 2007, Interpreted Language, **Purely OOP**
 - ❖ **Go**: 2009, Compiled, **Functional Language**
- Copyright © 2019 R.M. Laurie 8

Scratch Programming Environment

- ❖ Purely graphical yet powerful <https://scratch.mit.edu>
- ❖ A graphical programming environment based on flowchart algorithm design concepts
- ❖ <https://scratch.mit.edu/projects/327376223>



Copyright © 2019 R.M. Laurie 9

Implementing Program - Mathematics


- ❖ Computer programming applies mathematics
- ❖ **Variables** are containers similar to mathematics
 - ◆ Variables can contain strings of text (ASCII or Unicode)
 - ◆ Variables can contain numbers
 - ◆ Some variables are predefined in **Scratch**
 - ◆ You can create more variables to store data
 - ◆ **Local variables** in Scratch affect **single sprite**
 - ◆ **Global variables** in Scratch affect **all sprites**
- ❖ **Operators** process data

Arithmetic	Functional	Comparison	Relational

Copyright © 2019 R.M. Laurie 10

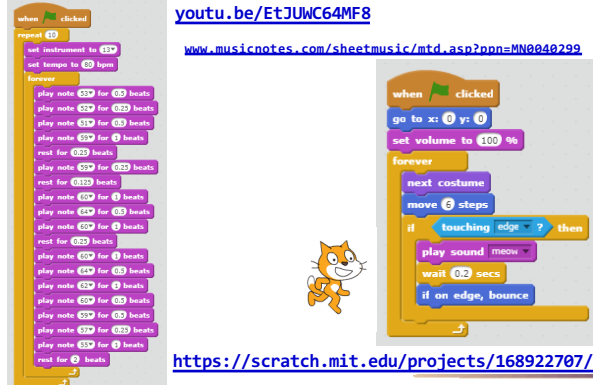
Implementing Program – Sequencing

- ❖ Computer programming creates a sequencing of events using control structures
 - ◆ Sequential Control Structure (Top-Bottom)
 - ◆ Selection Control Structure (Branching)
 - ◆ Decision making control
 - ◆ Tests an Assertion (T/F) to branch or not
 - ◆ Assertion uses Relational and Logical Operators
 - ◆ Repetition Control Structure (Looping)
 - ◆ Loop back and repeats process execution
 - ◆ Tests an Assertion (T/F) to loop again or exit loop
 - ◆ Assertion uses Relational and Logical Operators
 - ◆ Counter controlled or Sentinel controlled loops
 - ◆ Computers Never Get Bored
 - ▶ Best for iterative well structured processing
 - ▶ Not well suited for creative problem solving



Copyright © 2019 R.M. Laurie 11

Music and Motion are Sequential Processes



youtu.be/EtJUWC64MF8

www.musicnotes.com/sheetmusic/mtd.asp?ppn=MN0040299

<https://scratch.mit.edu/projects/168922707/>

Copyright © 2019 R.M. Laurie 12

Implementing Program – Input/Output

- ❖ Computer programs often require *input* data to be *processed* into information for *output*
- ❖ Scratch Input functions

ask [What's your name?] and wait

answer

key space pressed?

mouse down?

mouse x

mouse y
- ❖ Scratch Output functions

say Hello! for 2 secs

say Hello!

think Hmm... for 2 secs

think Hmm...

show

hide

switch costume to costume2

next costume

switch backdrop to backdrop1

Copyright © 2019 R.M. Laurie 13

Planning Phase

- ❖ Planning Phase
 - ◆ Write Program Specifications document
 - ◆ Analysis of requirements of program
 - ◆ Program specifications description
 - ◆ Describe the goals of the program
 - ◆ Describe appearance of input and output
- ❖ Design Phase
 - ◆ Algorithm Design
 - ◆ Mathematical Analysis and Algorithm
 - ◆ Flow Chart to describe event sequencing
 - ◆ Verify algorithm
 - ◆ Test with known data and Solve manually

Copyright © 2019 R.M. Laurie 14

Algorithm Design - Mathematical

- ❖ Create a program to convert Celsius temperature to Fahrenheit temperature
- ❖ Mathematical Description and test data
 - ◆ Boiling point
 - F = 212
 - C = 100
 - ◆ Freezing point
 - F = 32
 - C = 0

$F = (180 / 100) C + 32$
 $= (9/5) C + 32$
 $= 1.8 C + 32$

$y = mx + b$

Copyright © 2019 R.M. Laurie 15

Algorithm Design - Flowchart

- ❖ Flowcharts are an excellent way to plan the sequence of operations needed for the program to run

Common Flowchart Symbols

Terminator

Input/Output

↓ Connector

Process

Selection

Scratch Program

<https://scratch.mit.edu/projects/168924483/>

Copyright © 2019 R.M. Laurie 16

Verify Algorithm

- ❖ Testing with known data
 - ◆ Boiling point
F = 212 C = 100
 - ◆ Freezing point
F = 32 C = 0
 - ◆ Collect Data
 - ◆ Bank thermometer
 - ◆ Radio weather report
 - ◆ Internet
- ❖ Solve manually by hand using calculator

```

graph TD
    Start([Start]) --> DisplayIntro[/Display Introduction/]
    DisplayIntro --> PromptC[/Prompt and Input Celsius Temperature/]
    PromptC --> ConvertF[Convert to Fahrenheit Temperature]
    ConvertF --> DisplayBoth[/Display both Celsius Temperature And Fahrenheit Temperature/]
    DisplayBoth --> End([End])
    
```

Copyright © 2019 R.M. Laurie 17

Implementation Phase

- ❖ Translate Algorithm into Code
 - ◆ Run to detect *syntax errors*
- ❖ Test Program
 - ◆ Test with known data
 - ◆ Detects program *logic errors*
 - ◆ Often requires several iterations
 - ◆ May require re-evaluation of design specifications and algorithms

Copyright © 2019 R.M. Laurie 18

Enhanced Temperature Converter

- ❖ Specifications
 - ◆ Program prompts for input units and value then converts and display temperatures
 - ◆ Mathematical Algorithms
 - ◆ C → F
 $F = 1.8 C + 32$
 - ◆ F → C
 $C = (F - 32) / 1.8$
 - ◆ Test Data:
0°C = 32°F
100°C = 212°F
-12°C = 10.4°F

```

graph TD
    Start([Start]) --> PromptWhich[/Prompt Which Conversion/]
    PromptWhich --> CEntered{C entered?}
    CEntered -- True --> PromptC[/Prompt Celsius/]
    PromptC --> ConvertCF[Convert C to F]
    ConvertCF --> DisplayCF[/Display Temperature xC = xF/]
    CEntered -- False --> PromptF[/Prompt Fahrenheit/]
    PromptF --> ConvertFC[Convert F to C]
    ConvertFC --> DisplayFC[/Display Temperature xF = xC/]
    DisplayCF --> End([End])
    DisplayFC --> End
    
```

Copyright © 2019 R.M. Laurie 19

Enhanced Temperature Converter

<https://scratch.mit.edu/projects/168954282/#player>

Copyright © 2019 R.M. Laurie 20

Enhanced Temperature Converter

Specifications

- Program prompts for input units and value then converts and display temperatures
- Mathematical Algorithms**
 - $C \rightarrow F$
 $F = 1.8 C + 32$
 - $F \rightarrow C$
 $C = (F - 32)/1.8$
- Test Data:**
 $0^{\circ}C = 32^{\circ}F$
 $100^{\circ}C = 212^{\circ}F$
 $-40^{\circ}C = -40^{\circ}F$

```

1. # Temperature Converter
2. # Author: Robert Laurie
3. print("Temperature Converter")
4. sUnit = input("Enter Unit (C/F): ")
5. if sUnit == 'c' or sUnit == 'C':
6.     slnp = input("Enter °C: ")
7.     fCels = float(slnp)
8.     fFahr = 1.8 * fCels + 32
9.     print(fCels, "°C = ", fFahr, "°F")
10. elif sUnit == 'f' or sUnit == 'F':
11.     slnp = input("Enter °F: ")
12.     fFahr = float(slnp)
13.     fCels = (fFahr - 32)/1.8
14.     print(fFahr, "°F = ", fCels, "°C")
15. else:
16.     print("You must enter F or C")
17. print("Done")
    
```

Copyright © 2019 R.M. Laurie 21

Temperature Converter with Loop

```

when clicked
set Loop to 0
repeat until not Loop = 0
ask "What unit is your input temperature? c=Celsius or f=Fahrenheit" and wait
if answer = c then
ask "What is the Celsius temperature?" and wait
set TempC to answer
set TempF to (1.8 * TempC + 32)
say join TempC join "°C" join TempF 1 s
else
ask "What is the Fahrenheit temperature?" and wait
set TempF to answer
set TempC to (TempF - 32) / 1.8
say join TempF join "°F" join TempC 1 s
else
think [Error: Only c or f]
wait 2 secs
ask "Another temperature conversion? y or n" and wait
set Loop to answer
    
```

Copyright © 2019 R.M. Laurie 22

What unit is your input temperature?
c=Celsius or f=Fahrenheit

What is the Celsius temperature?

25°C = 77°F

Another temperature conversion? y or n

<https://scratch.mit.edu/projects/168959348/>

Dice Roller Loop Example

```

when clicked
set Execution Time to 0
set count to 0
repeat until count > 100
replace item count of RollCounters with 0
change count by 1
set count to 0
ask "How many times would you like to roll two dice?" and wait
set rolls to answer
reset timer
repeat until count = rolls
set die1 to pick random 1 to 6
set die2 to pick random 1 to 6
set dice to die1 + die2
replace item dice of RollCounters with item dice of RollCounters + 1
change count by 1
set Execution Time to timer
say join [Total] join Execution Time join seconds to roll two dice join rolls times
    
```

Copyright © 2019 R.M. Laurie 23

<https://scratch.mit.edu/projects/168963386/>

Loop Control Structures - for:

```

1. # Dice Roller
2. # Author: Robert Laurie
3. import random as rd, time as tm #Modules used
4. slnp = input("How many rolls? ")
5. nCnt=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] #Roll counters list
6. nTimeStart = tm.time() #Execution timer start time
7. for nl in range(int(slnp)): #for loop structure
8.     nSm = rd.randint(1,6) + rd.randint(1,6) #Roll 2 Dice
9.     nCnt[nSm] = nCnt[nSm] + 1 #Increment list element
10. print("Roll two dice", slnp, "times results")
11. nScale = max(nCnt) // 15 # Scale bar graph
12. if nScale == 0: nScale = 1
13. print("Each + represents", nScale, "rolls", "\n"+"-"*35)
14. for nl in range(2, 13, 1): #for loop structure
15.     sBr = '+'*(nCnt[nl]/nScale) # Create bar
16.     print("Roll{:3d} ={:7d} {}".format(nl, nCnt[nl], sBr))
17. nTime = (tm.time() - nTimeStart)*1000 # Runtime
18. print("-"*35, "\nRun time = {:.6f} mSec".format(nTime))
    
```

Copyright © 2019 R.M. Laurie 24

Loops processing

- Assertion (T/F) loop back or exit loop

Sentinel controlled

- while :

Counter controlled

- for in :
- Roll dice in 16 lines