# Welcome!

Making Game with Python (2)

Zhihong (John) Zeng & Andrew Zeng

# Today

- Course info
- Computer fundamentals
- Python basics
- Mathematical operations
- Python variables and types
- Install software

# Course Info

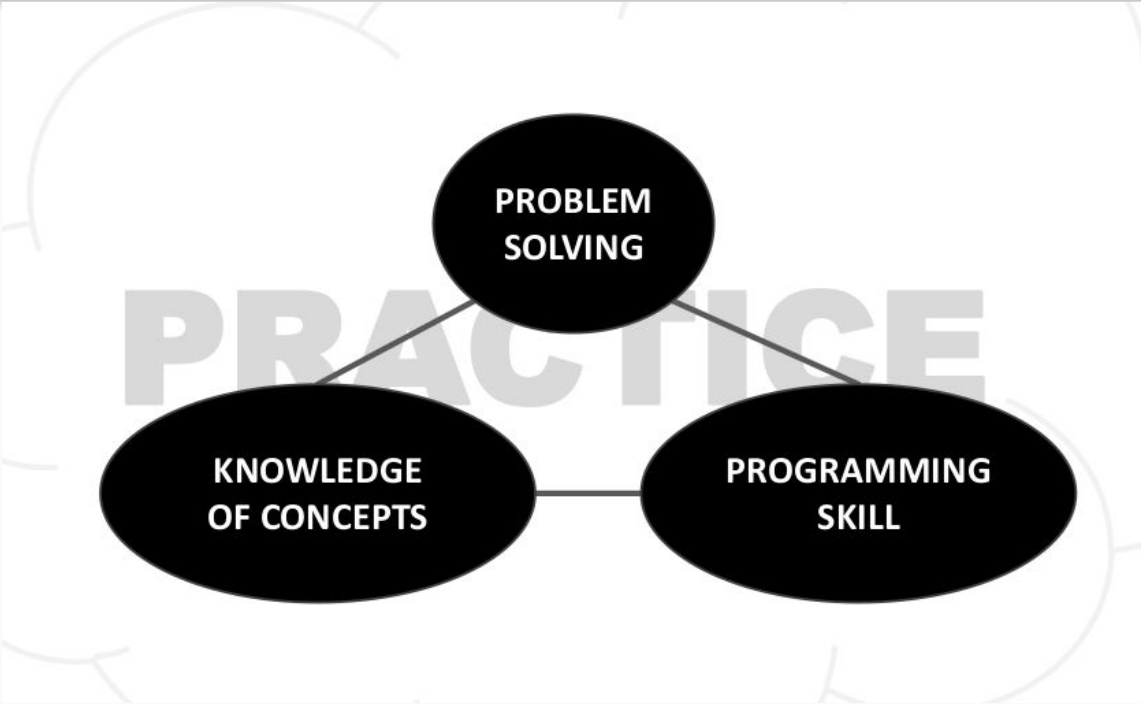
## Make Games with Graphics


*Making Games with Python & Pygame* covers the Pygame library with the source code for 11 games. Making Games was written as a sequel for the same age range as Invent with Python. Once you have an understanding of the basics of Python programming, you can now expand your abilities using the Pygame library to make games with graphics, animation, and sound.

The book features the source code to 11 games. The games are clones of classics such as Nibbles, Tetris, Simon, Bejeweled, Othello, Connect Four, Flood It, and others.

Buy on Amazon | Read Online for Free

# Course Info

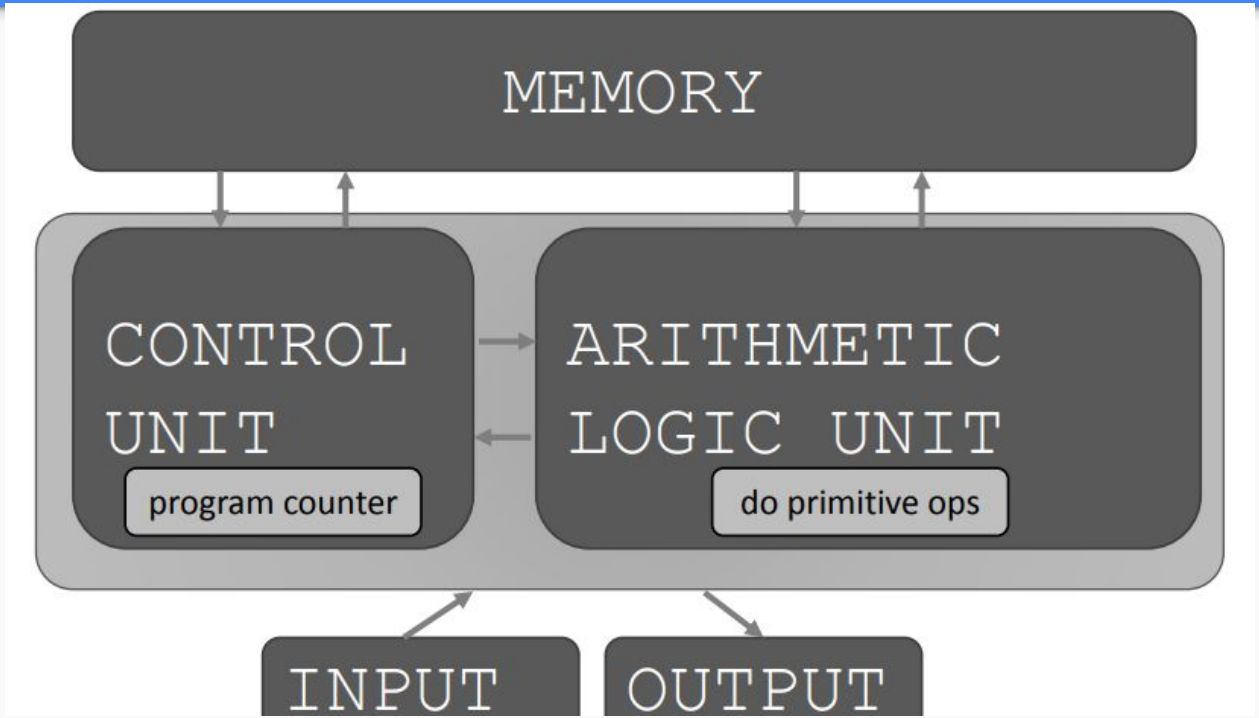

PROBLEM SOLVING
PRACTICE
KNOWLEDGE OF CONCEPTS
PROGRAMMING SKILL

# Computer Fundamentals

# What does a Computer do?

- Fundamentally:
  - Performs calculations
  - Remembers results
- What kinds of calculations:
  - Built-in to the language
  - Ones that you define as the programmer
- Computers only know what you tell them

# Basic Computer Architecture



# A Numerical Example

1. Positive integer multiplication by using addition: A = x*y
2. Recipe for deducing multiplication of x and y (sequence of simple steps)
   - 2.1. Start with integers x and y, and A=0
   - 2.2. If y is 0, stop and return A
   - 2.3. Otherwise y = y-1 and A = A+x, and continue Step 2.2

| x | y | A |
|---|---|---|
| 2 | 3 | 2 |
| 2 | 2 | 2+2=4 |
| 2 | 1 | 4+2=6 |
| 2 | 0 | 6 |

# What is a programming recipe

- Sequence of simple steps
- Flow of control process that specifies when each step is executed
- A means of determining when to stop

```
1+2+3 = an algorithm
```

# Stored Program Computer

- Sequence of instructions stored inside computer
  - Built from predefined set of primitive instructions
    - Arithmetic and logic
    - Simple tests
    - Moving data
- Special program (interpreter) executes each instruction in order
  - Change flow of control though sequence
  - Stop when done

# Basic Primitives

- Tuning showed that you can compute anything using 6 primitives (left, right, scan, print, erase, and do nothing)
- Modern programming languages have more convenient set of primitives
- Use abstract methods to create new primitives
- Anything computable in one language is computable in any other programming language

# Create Recipes

- A programming language provides a set of primitive operations
- Expressions are complex combination of primitives in a programming language
- Expressions and computations have values and meanings in a programming language

# Aspects of Languages

- Primitive constructs
  - English: words
  - Programming language: numbers, strings, simple operators


Word Cloud copyright Michael Twardos, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.

Word Cloud copyright unknown, All Right Reserved. This content is excluded from our Creative Commons license. For more information, see https://ocw.mit.edu/help/faq-fair-use/.

# Aspects of Languages

- Syntax
  - English:
    - “Cat dog boy” -> not syntactically valid
    - “Cat hugs boy” -> syntactically valid but
  - Programming language
    - “hi” 5 -> not syntactically valid
    - 3.2*5 -> syntactically valid

# Aspects of Languages

- Language meaning
  - English: Can have many meaning
    - “Flying planes can be dangerous”
  - Programming language: can have only one meaning but may not be what a programmer intend

# Where Things Go Wrong

- Syntactic error
  - Common and easily caught
- Static semantic errors
  - Some languages check for these before running program
- No semantic errors but different meaning from what a programmer intended
  - Program crashes, stop running
  - Program run forever
  - Program gives an answer but different from expected

# Python Basics

# Python Programs

- A program is a sequence of definitions and commands
  - Definitions evaluated
  - Commands executed by Python interpreter in a shell
- Commands (statements) instruct interpreter to do something
- Can be typed directly in a shell or stored in a file that is read into the shell and evaluated intended

# Objects

- Programs manipulate data objects
- Objects have a type that defines the kinds of things program can do to them
  - Andrew is a human so he can walk, speak English, etc.
  - Chewbacca (Star wars) is a wookie so he can walk, “mwaaarhrhh”, etc.
- Objects are
  - Scalar (cannot be subdivided)
  - Non-scalar (have internal structure that can be accessed)

# Scalar Objects

- int -- represent integers, ex. 5
- float -- represent real numbers, ex. 3.27
- bool -- represent Boolean values True and False
- NoneType -- special and has no value, None
- Can use type() to see the type of an object
  - >>>type(5) -->int
  - >>>type(3.0) -->float

# Type Conversions (Cast)

- Can convert object of one type to another
  - float(3) converts integer 3 to float 3.0
  - int(3.9) truncates float 3.9 to integer 3
  - int(‘321’) converts string ‘321’ to integer 321
  - str(123) converts integer 123 to string ‘123’

# Printing to Console

- To show output from code to a user, use print command
  - print(3)
  - print(3+2)
  - print('ABC')

# Expressions

- Combine objects and operators to form expressions
- An expression has a value, which has a type
- Syntax for a simple expression
  - <object> <operator> <object>

# Operators on ints and floats

- i + j → sum
- i - j → difference
- i * j → product

  Note: if both are ints, result is int. If either or both are floats, result is float

- i / j → division. result is float
- i % j → remainder when i is divided by j
- i ** j → i to the power of j

# Operation precedence (order)

- Parentheses used to tell Python to do these operations first
- Operator precedence without parentheses
  - **
  - *
  - /
  - + and - executed left to right, as appear in expression

# Binding Variables and Values

- Equal sign is an assignment of a value to a variable name
  - Pi = 3.14
  - Pi_approx = 22/7
- Value stored in computer memory
- An assignment binds name to value
- Retrieve value associated with name or variable by invoking the name

# Abstracting Expressions

- Why give names to values of expressions
  - To reuse names instead of values
  - Easier to change code later

```
pi = 3.14
radius = 2.2
area = pi * (radius ** 2)
```

# Programming vs Math

- In programming, you do not “solve for x”
- Programming:
  - Expression on the right
  - Variable name on the left
  - Bonus: Equivalent expression to radius = radius + 1 is radius += 1

Python:

```
pi = 3.14
radius = 2.2
area = pi * (radius ** 2)
```

Math:

```
pi = 3.14
radius = 2.2
pi * (radius ** 2) = area
```

# Changing Bindings

- Can re-bind variable names using new assignment statements
- Previous value may still stored in memory but lost the handle for it
- Value for area does not change until you tell the computer to do the calculation again

```
pi = 3.14
radius = 2.2
area = pi * (radius ** 2)
radius = radius + 1
```

# Install Software

- Install python 3.6.1 or greater (latest 3.7.2)
  - www.python.org
- Install pygame:
  - `python3 -m pip install -U pygame --user`
  - `python3 -m pygame.examples.aliens`