Recall from 221…

programming is the process of designing/implementing/debugging algorithms in a format that computers can understand & execute

- high-level languages (e.g., Python, Java, C++, C#, PHP, JavaScript) enable the programmer to use abstract constructs (e.g., variables, while loop)
- a compiler and/or interpreter translates the high-level code into executable machine code

variables

- are names that correspond to values, which can be set/updated via assignment statements

  >>> name = 'Guido'

- languages provide built-in operators for manipulating/combining values, expressions can appear on the right-hand side of assignments

  >>> average = (num1 + num2 + num3)/3.0
Recall from 221...

control statements provide for conditional execution and repetition
- in Python: if, if-else, if-elif-else, while, for

```python
if average >= 60:
    print('Nice job, you passed."
else:
    print('You failed. Get to work!"

num = 10
while num > 0:
    print(num)
    num = num - 1

sum = 0
for i in range(1, 11):
    sum = sum + i
print(sum)
```

Recall from 221...

functions enable the programmer to group statements together under a single name
- a function is a unit of "computational abstraction"
- parameters make the function general, can be called on different values
- return statements make it possible to use the result in an expression

```python
def sumToN(N):
    sum = 0
    for i in range(1, N+1):
        sum = sum + i
    return sum

>>> sumToN(10)
55
>>> sumToN(100)
5050
>>> sumToN(100)/100
50.5
```
Recall from 221…

strings and lists are simple, indexable data structures
- a string is a sequence of characters, with built-in operations (e.g., len, [], upper)
- a list is a sequence of arbitrary items, with built-in operations (e.g., len, [], reverse)

```python
def acronym(phrase):
    words = phrase.split()
    str = ''
    for w in words:
        str = str + w[0]
    return str.upper()

>>> acronym("What you see is what you get")
'WYSIWYG'

>>> acronym("Fouled up beyond all recognition")
'FUBAR'
```

Recall from 221…

221 covered other language features & programming techniques
- list comprehensions
- library/module use
- I/O & file processing
- simple OO

my last 221: Skip-3 Solitaire game
- DeckOfCards class captured properties/behavior of a deck of cards
- RowOfCards class captures properties/behaviors of a row of cards
- Skip3 function could create objects to model these parts of the game, then focus on the game interface
Object-oriented programming

the *object-oriented* approach to programming:
- solve problems by modeling real-world objects
e.g., if designing a banking system, model clients, accounts, deposits, …
- a program is a collection of interacting objects

- in software, objects are created from classes
  *the class describes the kind of object (its properties and behaviors)*
  *the objects represent individual instantiations of the class*

REAL WORLD CLASS: automobiles
REAL WORLD OBJECTS: my 2011 Subaru Outback, the batmobile, …

- the class encompasses all automobiles
  they all have common properties: color, seats, wheels, engine, brakes, …
  they all have common behaviors: can sit in them, start them, accelerate, steer, …
- each car object has its own specific characteristics and ways of producing behaviors
  my car is white & seats 5; the batmobile is black & seats 2
  accelerating with V-8 is different than accelerating with jet engine

Die example

Die properties?
- # sides
- # of rolls

 behaviors?
- __init__ (to initialize)
- roll
- numberOfSides
- numberOfRolls

```python
>>> d6 = Die(6)
>>> d8 = Die(8)
>>> d6.roll()
3
>>> d6.roll()
6
>>> d6.numberOfRolls()
2
>>> d8.numberOfRolls()
0
```
Skip-3 solitaire example

**DeckOfCards**

properties?
- list of cards (e.g., "QH")

behaviors?
- `__init__` (to initialize)
- `shuffle`
- `dealCard`
- `addCard`
- `numCards`
- `__str__` (convert to string)

**RowOfCards**

properties?
- list of cards (front == left)

behaviors?
- `__init__` (to initialize)
- `addAtEnd`
- `moveCard`
- `numCards`
- `__str__` (convert to string)

Shape classes and objects

A slightly more abstract example involves shapes

- **class:** circles
  - what properties do all circles share?
  - what behaviors do all circles exhibit?

- **objects:**
  - ![Red Circle](Image1)
  - ![Yellow Circle](Image2)
  - ![Blue Circle](Image3)

Similarly, could define classes and object instances for other shapes

- **squares:**
  - ![Green Square](Image4)
- **triangles:**
  - ![Pink Triangle](Image5)
BlueJ and software shapes

the BlueJ interactive development environment (IDE) is a tool for developing, visualizing, and debugging Java programs

- BlueJ was developed by researchers at Deakin University (Australia), Maersk Institute (Denmark), and University of Kent (UK)
- supported by Oracle (previously Sun Microsystems), the developers of Java
- BlueJ includes an editor, debugger, visualizer, documentation viewer, ...

we will start with a visual example in BlueJ: drawing shapes

Starting up BlueJ

to start up the BlueJ IDE, double-click on the BlueJ desktop icon

this opens the BlueJ main window

- in order to create and execute a program, must first create or load a project
- a project groups together all the files needed to produce a working program
- e.g., could create a project for the entire semester, or HW by HW

to create a new BlueJ project

- click on the Project heading at the top left & select New Project
- enter the project name and location

to open an existing BlueJ project

- click on the Project heading at the top left & select Open Project
- browse to locate and select the project
Loading the figures project

download figures.zip from the class Code directory
- save it on the Desktop and double-click to unzip
- in BlueJ, select Open Project
- browse to select figures

when a project loads, its classes are shown in a diagram
- here, there are 5 classes
- Canvas represents a painting area
- Circle, Square, Triangle, and Person represent shapes
- the arrows show that the shapes depend upon the Canvas class

Editing and compiling classes

you can view/edit a class definition by double-clicking on its box
- this opens the associated file in the BlueJ editor

before anything can be executed, the classes must be compiled
- recall, the Java compiler translates Java source code into Java byte code
- to compile all classes in a project, click on the Compile button
  (note: non-compiled classes are shaded, compiled classes are not)

IMPORTANT: classes don’t act, objects do!
- you can’t drive the class of all automobiles
- but you can drive a particular instance of an automobile

in order to draw a circle, must create a circle object
- then, can specify properties of that instance (radius, color, position, …)
Example: creating a circle

right-click on a class to see all the actions that can be applied
- select `new Circle()` to create a new object
- you will be prompted to specify a name for that object (circle1 by default)

the new Circle object appears as a box at the bottom of the screen
- note: classes and objects look different

EXERCISE: create 2 circles, a square, and a triangle

Applying object methods

can cause objects to act by right-clicking on the object box, then selecting the action
- the actions that objects can perform are called methods
  (same as in Python: a method is a function that belongs to an object.)

- here, `void makeVisible()` opens a Canvas in which the shape is displayed

EXERCISE: make the other shapes visible

EXERCISE: select other methods to change the color and size of objects

EXERCISE: play
Methods and parameters

sometimes an action (i.e., method) requires information to do its job

- the `changeColor` method requires a color ("red", "green", "black", ...)
- the `moveHorizontal` method requires a number (# of pixels to move)

- data values provided to a method are called *parameters*

Java provides for different types of values

- `String` is a sequence of characters, enclosed in double-quotes (e.g., "red")
- `int` is an integer value (e.g., 40)
- `double` is a real value (e.g., 3.14159)
- `char` is a character value (e.g., 'A')

- the parameter to `changeColor` is a `String` representing the new color
- the parameter to `moveHorizontal` is an `int` representing the # of pixels to move

Objects and state

recall that each object has properties and methods associated with it

- when you create a Circle, it has an initial size, color, position, ...
- those values are stored internally as part of the object
- as methods are called, the values may change

- at any given point, the property values of an object define its state

BlueJ enables you to inspect state of an object

- right-click on the object
- select Inspect to see the values of object properties

note: objects of the same class have the same properties, but may have different values
IN-CLASS EXERCISE

create objects and call the appropriate methods to produce a picture like this

Another example: Die class

can define a Die class to model different (numeric) dice

- properties shared by all dice: number of sides, number of times rolled
- behaviors/methods shared by all dice: roll it, get # of sides, get # of rolls
- the roll method generates a random roll and returns it
- the return value is displayed by BlueJ in a Method Result window
Another example: SequenceGenerator

there are two options for creating a SequenceGenerator object
  - can specify an alphabet to choose from (e.g., "etainshrd")
  - if nothing specified, will assume "abcdefghijklmnopqrstuvwxyz"