Polymorphism

- A function is polymorphic if ???
  - It can be applied to several different types of data
- Why function polymorphism?
  - Avoid code duplication
- How is it implemented?
  - Depends
  - C uses the void * data type and function pointers to provide polymorphism (qsort is an example)
  - Java uses generic type (Stack.java)
  - How do you implement a generic stack in Python? Python is a dynamically typed language, it supports different data types naturally (Stack.py).
    - List in Python can be used as stack.
    - Its append() method can be used as push, adding to the top of a stack
    - Its pop() method is the same as stack pop, removing elements in a LIFO order.

```python
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
    def pop(self):
        if not self.isEmpty():
            return self.stack.pop()
        else:
            return None
    def peek(self):
        if not self.isEmpty():
            return self.stack[len(self.stack)-1]
        else:
            return None
    def isEmpty(self):
        return len(self.stack) == 0

def main():
    stk = Stack()
    print(stk.pop())
    for i in range(0, 5):
        stk.push(i)
        print("top is: ", stk.peek())
        for i in range(0, 5):
            print("pop: ", stk.pop())
            print(stk.peek())
if __name__ == '__main__':
    main()
```
/**
 * Generic Stack
 * 
 * Ying Li
 * 09/21/2020
 */
class Stack<T>{
    Node<T> top;

    T pop () {
        if (top != null) {
            T item = top.data;
            top = top.next;
            return item;
        }
        return null;
    }

    void push (T item) {
        Node<T> t = new Node<T> (item);
        t.next = top;
        top = t;
    }

    T peek () {
        return top.data;
    }

    boolean isEmpty () {
        if (top == null) {
            return true;
        } else {
            return false;
        }
    }
}

public static void main (String[] args) {
    Stack<Integer> s = new Stack<Integer>();
    s.push(3);
    s.push(4);
    s.push(5);
    while (!s.isEmpty()) {
        System.out.println(s.pop());
    }
}

class Node<T> {
    Node<T> next = null;
    T data;

    public Node (T d) {
        data = d;
    }
}
- Polymorphism in OO language is not just function polymorphism.
  - Inheritance
    - A class inherits characteristics from parent classes
  - **Override**
    - A subclass redefine the method inherited from a super class
    - Overriding vs overloading
      - **Overriding**: same method name, same parameters in different class (run-time concept)
      - **Overloading**: same method name, different parameters in one class (compile-time concept); Java supports overloading but Python doesn’t.
  - **Abstract class**
    - Can be inherited only, cannot have instances
    - Provide default function implementation for the subclasses.
    - Java: declared by using the keyword, abstract
      - Comparing with Interfaces, abstract classes have some implementation.
    - Python: a class with at least one abstract method
      - abstract method: a method that has declaration but not implementation
  - **Multiple inheritance**:
    - Inherit more than one class (Python class can inherit from multiple classes)
    - Java can allow one subclass extend from one super class, but it allows a class implement multiple interfaces.
Conversion.py

class Human:
    def setID (self, id):
        self.id = id

    # abstract method
    def isHappy (self):
        pass

    def printt (self):
        print(self.id, ", ", self.isHappy())

class Parent (Human):
    def __init__ (self, id):
        Human.setID(self, id)

    def isHappy (self):
        return False

    # python doesn't support overloading
    def isHappy (self, age):
        if age < 25:
            return True
        else:
            return False

    # isHappy invoked in printt is the most recent one
    # so the interpreter complains it doesn't have param
    def printt (self):
        print("ID %d is %s" % (self.id, self.isHappy()))

class Child (Human):
    def __init__ (self, id):
        Human.setID(self, id)

    def isHappy (self):
        return True

def main ():
    p = Parent(100)
    c = Child(101)

    p.printt()
    c.printt()

if __name__ == "__main__":
    main()
/**
 * File: ConversionII.java
 * Author: Ying Li
 * Date: 09/21/2020
 */

public class ConversionII {
    public static void main (String[] argv) {
        Parent p = new Parent(100);
        Child c = new Child(101);
        Human h1 = p;
        Human h2 = c;

        h1.print();
        h2.print();
    }
}

abstract class Human {
    protected int id;

    public void setID (int id) { this.id = id; }
    public abstract boolean isHappy ();
    public void print () { System.out.println(id + " isHappy()"); }
}

class Parent extends Human {
    public Parent (int id) {
        setID (id);
    }

    public boolean isHappy () { return false; }

    // overloading
    public boolean isHappy (int age) {
        if (age < 25) { return true; }
        else { return false; }
    }

    // overriding
    public void print () { System.out.printf("ID %d is %s\n", id, isHappy()); }
}

class Child extends Human {
    public Child (int id) {
        setID (id);
    }

    public boolean isHappy () { return true; }
}