Proyecto 0: Tutorial de Unix/Python/Conda/Autograder y Herencia

Adaptación basada en el curso de Introducción a la Inteligencia Artificial de la universidad de Berkeley

---

Introducción

Los proyectos para este laboratorio asumen que usa Python 3.6 (en algunos otros se empleará Python 2.7). El calificador automático generará un error con un mensaje sobre cgi.escape si está utilizando Python 3.8+

El Proyecto 0 cubrirá lo siguiente:

- •Un tutorial mini-UNIX (particularmente importante si trabaja en máquinas instructivas),
- •Instrucciones sobre cómo configurar la versión correcta de Python,
- •Un tutorial de mini-Python,
- •Calificación de proyectos: la versión de cada proyecto incluye su calificador automático para que lo ejecutes tu mismo.
- •Ejercicios de calentamiento de Python
- •Mini-projecto de Python

Archivos para editar y trabajar:

Completarás partes de `addition.py` `buyLotsOfFruit.py` `shopSmart.py` del tutorial.zip durante la tarea. ¡CUIDADO! no cambies otros archivos en esta distribución. Encontrarás también un notebook donde están los ejercicios básicos de python.

Conceptos básicos de Unix

Aquí hay comandos básicos para navegar UNIX y editar archivos que ya deberías conocer, si no es el caso recupera los apuntes de ISO y ponte al día.

Manipulación de archivos/directorios

Cuando abres una ventana de terminal aparecerá el símbolo del sistema:

```
[xxxx ~]$
```

Copiar

El indicador muestra tu nombre de usuario, el host en el que ha iniciado sesión y tu ubicación actual en la estructura del directorio (su ruta). Ten en cuenta que tu mensaje puede verse ligeramente diferente. Para crear un directorio, emplea el `mkdir`. Emplea `cd` para cambiarte al directorio recién creado directorio:

```
[xxxx ~]$ mkdir foo

[xxxx ~]$ cd foo

[xxxx ~/foo]$
```

Copiar

Emplea `ls` para ver en modo lista del contenido del directorio y `touch` o `nano` para crear un archivos vacíos:

```
[xxxx ~/foo]$ ls

[xxxx~/foo]$ touch hello_world

[xxxx ~/foo]$ ls

hello_world

[xxxx ~/foo]$ cd ..

[xxxx ~]$
```

Copiar

Descarga python_basics.zip en tu directorio de inicio (nota: el nombre del archivo zip puede ser ligeramente diferente cuando lo descargues). Emplea `unzip` para extraer el contenido del archivo zip:

```
[xxxx ~]$ ls *.zip

python_basics.zip

[xxxx ~]$ unzip python_basics.zip

[xxxx ~]$ cd python_basics

[xxxx ~/python_basics]$ ls

foreach.py

helloWorld.py

listcomp.py

listcomp2.py

quickSort.py

shop.py

shopTest.py
```

Copiar

Algunos otros comandos útiles de Unix:

- `cp` copia un archivo o archivos
- `rm` elimina (borra) un archivo
- `mv` mueve un archivo (es decir, cortar/pegar en lugar de copiar/pegar)
- `man` muestra la documentación de un comando
- `pwd` imprime tu ruta actual
- `xterm` abre una nueva ventana de terminal
- `firefox` abre un navegador web
- •Presiona "Ctrl-c" para matar un proceso en ejecución
- •Añade `&` a un comando para ejecutarlo en segundo plano (en modo background)
- `fg` vuelve a traer un programa que se ejecuta en segundo plano al primer plano

El editor de texto de xEmacs

Emacs es un editor de texto personalizable que tiene algunas características agradables diseñadas específicamente para programadores. Sin embargo, puedes usar cualquier otro editor de texto que prefieras (como `vi`, `pico` o `joe` en Unix; o Notepad en Windows; o TextWrangler en OS X; o un IDE como `Pycharm`).

Instalación de Python

Muchos de vosotros habéis empleado Python en SAD a través de Anaconda y ya lo tendréis instalado en vuestros ordenadores, comprobad que tenéis un enviroment para cada versión ya instalada en vuestros ordenadores y en los ordenadores del laboratorio, de no ser así deberemos proceder a su creación. Para aquellos que no lo conozcáis del año pasado Conda y Anaconda proporcionan una manera fácil de administrar muchos entornos diferentes, cada uno con sus propias versiones y dependencias de Python. Esto nos permite evitar conflictos entre nuestra versión preferida de Python y la de otras clases o otros laboratorios. Veremos cómo configurar y usar un entorno conda y anaconda si no lo habéis hecho nunca.

Prerrequisito: Anaconda. Muchos lo tendréis instalado de SAD; si no lo tienes, instálalo siguiendo los pasos:

Creación de un entorno Conda

El comando para crear un entorno conda con Python 2.7 es:

```
conda create --name <env-name> python=2.7
```

Copiar

el 3.6 es:

```
conda create --name <env-name> python=3.6
```

Copiar

Para nosotros, decidimos nombrar nuestro entorno `tia27` y `tia36` por lo que ejecutamos el siguiente comando para confirmar la instalación de los paquetes faltantes.

```
[xxxx ~/python_basics]$ conda create --name tia27 python=2.7
```

Copiar

```
[xxxx ~/python_basics]$ conda create --name tia36 python=3.6
```

Copiar

Entrar en el entorno (*enviroment)*

Para entrar al entorno conda que acabamos de crear, haz lo siguiente. Tenga en cuenta que la versión de Python dentro del entorno es 3.6, justo lo que queremos.

```
[xxxx ~/python_basics]$ conda activate tia36

(tia36) [xxxx ~/python_basics]$ python -V

Python 3.6.6 :: Anaconda, Inc.
```

Copiar

Nota: la etiqueta `(<env-name>)` le muestra el nombre del entorno de conda que está activo. En nuestro caso, tenemos `(tia36)`, como esperábamos.

Saliendo del entorno (*enviroment*)

Salir del medio ambiente es igual de fácil.

```
(cs188) [xxxx ~/python_basics]$ conda deactivate
```

Copiar

¡Nuestra versión de python volverá a la configuración predeterminada del sistema!

Conceptos básicos de Python

Archivos requeridos

Puede descargar todos los archivos asociados con el minitutorial de Python como un archivo zip: python_basics.zip. Si hizo el tutorial de Unix en la pestaña anterior, ya ha descargado y descomprimido este archivo.

Tabla de contenido

Las tareas de programación de este curso se escribirán en Python, un lenguaje interpretado orientado a objetos que comparte algunas características con Java. Este tutorial te guiará a través de las construcciones sintácticas primarias en Python, usando ejemplos breves.

Invocando al Intérprete

Python se puede ejecutar en uno de dos modos. Se puede utilizar de forma interactiva, a través de un intérprete, o se puede llamar desde la línea de comandos para ejecutar un script. Primero usaremos el intérprete de Python de forma interactiva.
Invocas al intérprete usando el comando `python` en el símbolo del sistema de Unix;

```
(tia36) [xxxx ~]$ python

Python 3.6.6 |Anaconda, Inc.| (default, Jun 28 2018, 11:07:29)

[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>
```

Copiar

Operadores

El intérprete de Python se puede utilizar para evaluar expresiones, por ejemplo, expresiones aritméticas simples. Si ingresa dichas expresiones en el indicador (`>>>`), se evaluarán y el resultado se devolverá en la línea siguiente.

```
>>> 1 + 1

2

>>> 2 * 3

6
```

Copiar

Los operadores booleanos también existen en Python para manipular los valores primitivos `True` y . `False`

```
>>> 1 == 0

False

>>> not (1 == 0)

True

>>> (2 == 2) and (2 == 3)

False

>>> (2 == 2) or (2 == 3)

True
```

Copiar

Instrucciones con Strings

Al igual que Java, Python tiene un tipo de cadena integrado. El `+` operador está sobrecargado para realizar la concatenación de cadenas en valores de cadena.

```
>>> 'artificial' + "intelligence"

'artificialintelligence'
```

Copiar

Hay muchos métodos incorporados que le permiten manipular cadenas.

```
>>> 'artificial'.upper()

'ARTIFICIAL'

>>> 'HELP'.lower()

'help'

>>> len('Help')

4
```

Copiar

Ten en cuenta que podemos usar comillas simples `''` o comillas dobles `""` para rodear la cadena. Esto permite anidar fácilmente los strings.

También podemos almacenar expresiones en variables.

```
>>> s = 'hello world'

>>> print(s)

hello world

>>> s.upper()

'HELLO WORLD'

>>> len(s.upper())

11

>>> num = 8.0

>>> num += 2.5

>>> print(num)

10.5
```

Copiar

En Python, no tiene que declarar variables antes de asignarlas.

Ejercicio: dir y Ayuda

Obtén información sobre los métodos que proporciona Python para las cadenas. Para ver qué métodos proporciona Python para un tipo de datos, emplea la documentación oficial de Python en internet o los comandos `dir` y `help`

```
>>> s = 'abc'



>>> dir(s)

['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__str__', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'index',
'isalnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'replace', 'rfind',
'rindex', 'rjust', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

```
>>> help(s.find)
Help on built-in function find:

find(...) method of builtins.str instance
    S.find(sub[, start[, end]]) -> int

    Return the lowest index in S where substring sub is found,
    such that sub is contained within S[start:end].  Optional
    arguments start and end are interpreted as in slice notation.

    Return -1 on failure.



>>> s.find('b')
1
```

Copiar

Prueba algunas de las funciones de cadena enumeradas en `dir` (ignore aquellas con guiones bajos '_' alrededor del nombre del método).

Estructuras de datos integradas

Python viene equipado con algunas estructuras de datos integradas útiles, muy similares al paquete de colecciones de Java.

Liza

Las listas almacenan una secuencia de elementos mutables, son los equivalentes a los arrays:

```
>>> fruits = ['apple', 'orange', 'pear', 'banana']
>>> fruits[0]
'apple'
```

Copiar

Podemos usar el `+` operador para hacer la concatenación de listas:

```
>>> otherFruits = ['kiwi', 'strawberry']
>>> fruits + otherFruits
>>> ['apple', 'orange', 'pear', 'banana', 'kiwi', 'strawberry']
```

Copiar

Python también permite la indexación negativa desde el final de la lista. Por ejemplo, `fruits[-1]` accederá al último elemento `'banana'`:

```
>>> fruits[-2]
'pear'
```

```
>>> fruits.pop()
'banana'
>>> fruits
['apple', 'orange', 'pear']
>>> fruits.append('grapefruit')
>>> fruits
['apple', 'orange', 'pear', 'grapefruit']
>>> fruits[-1] = 'pineapple'
>>> fruits
['apple', 'orange', 'pear', 'pineapple']
```

Copiar

También podemos indexar múltiples elementos adyacentes usando el operador de división. Por ejemplo, `fruits[1:3]` devuelve una lista que contiene los elementos en la posición 1 y 2. En general `fruits[start:stop]` obtendrá los elementos en `start, start+1, ..., stop-1`. También podemos hacer `fruits[start:]` lo que devuelve todos los elementos a partir del `start` índice. También `fruits[:end]` devolverá todos los elementos antes del elemento en la posición `end`:

```
>>> fruits[0:2]
['apple', 'orange']
>>> fruits[:3]
['apple', 'orange', 'pear']
>>> fruits[2:]
['pear', 'pineapple']
>>> len(fruits)
4
```

Copiar

Los elementos almacenados en las listas pueden ser de cualquier tipo de datos de Python. Entonces, por ejemplo, podemos tener listas de listas:

```
>>> lstOfLsts = [['a', 'b', 'c'], [1, 2, 3], ['one', 'two', 'three']]
>>> lstOfLsts[1][2]
3
>>> lstOfLsts[0].pop()
'c'
>>> lstOfLsts
[['a', 'b'], [1, 2, 3], ['one', 'two', 'three']]
```

Copiar

Ejercicio: Listas

Juega con algunas de las funciones de la lista. Puedes encontrar los métodos a los que puedes llamar en un objeto a través de `dir` y obtener información sobre ellos a través del `help`:

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
'__delslice__', '__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__setslice__', '__str__',
'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

Copiar

```
>>> help(list.reverse)
    Help on built-in function reverse:

    reverse(...)
    L.reverse() -- reverse \*IN PLACE\*
```

Copiar

```
>>> lst = ['a', 'b', 'c']
>>> lst.reverse()
>>> ['c', 'b', 'a']`
```

Copiar

Nota: Ignora las funciones con guiones bajos “_” alrededor de los nombres; estos son métodos de ayuda privados. Pulse 'q' para salir de una pantalla de ayuda.

Tuplas

Una estructura de datos similar a la lista es la tupla, que es como una lista excepto que es **inmutable** una vez creada (es decir, no puede cambiar su contenido una vez creada). Sería algo más similar a un enumerado. Ten en cuenta que las **tuplas están entre paréntesis** mientras que las **listas tienen corchetes**.

```
>>> pair = (3, 5)
>>> pair[0]
3
>>> x, y = pair
>>> x
```

```
3
>>> y
5
>>> pair[1] = 6
TypeError: object does not support item assignment
```

Copiar

El intento de modificar una estructura inmutable provoca una excepción.

Conjuntos

Un conjunto es otra estructura de datos que sirve como una lista desordenada sin elementos duplicados. A continuación, mostramos cómo crear un conjunto:

```
>>> shapes = ['circle', 'square', 'triangle', 'circle']
>>> setOfShapes = set(shapes)
```

Copiar

A continuación se muestra otra forma de crear un conjunto:

```
>>> setOfShapes = {‘circle’, ‘square’, ‘triangle’, ‘circle’}
```

A continuación, mostramos cómo agregar cosas al conjunto, probar si un elemento está en el conjunto y realizar operaciones de conjuntos comunes (diferencia, intersección, unión):

```
>>> setOfShapes
set(['circle', 'square', 'triangle'])
>>> setOfShapes.add('polygon')
>>> setOfShapes
set(['circle', 'square', 'triangle', 'polygon'])
>>> 'circle' in setOfShapes
True
>>> 'rhombus' in setOfShapes
False
>>> favoriteShapes = ['circle', 'triangle', 'hexagon']
>>> setOfFavoriteShapes = set(favoriteShapes)
>>> setOfShapes - setOfFavoriteShapes
set(['square', 'polygon'])
>>> setOfShapes & setOfFavoriteShapes
set(['circle', 'triangle'])
>>> setOfShapes | setOfFavoriteShapes
set(['circle', 'square', 'triangle', 'polygon', 'hexagon'])
```

Copiar

Ten en cuenta que los objetos del conjunto están desordenados; ¡no puede suponer que su orden de recorrido o de impresión será el mismo en todas las máquinas!

Diccionarios

La última estructura de datos incorporada es el diccionario que almacena un mapa de un tipo de objeto (la clave) a otro (el valor). La clave debe ser de un tipo inmutable (cadena, número o tupla). El valor puede ser cualquier tipo de datos de Python.
Nota: En el siguiente ejemplo, el orden de impresión de las claves devueltas por Python podría ser diferente al que se muestra a continuación. La razón es que, a diferencia de las listas que tienen un orden fijo, un diccionario es simplemente una tabla hash para la que no hay un orden fijo de las claves (como HashMaps en Java). El orden de las claves depende de la forma exacta en que el algoritmo hash asigna las claves a los cubos y, por lo general, parecerá arbitrario. Su código no debe basarse en el orden de las claves, y no debería sorprenderse si incluso una pequeña modificación en la forma en que su código usa un diccionario da como resultado un nuevo orden de claves.

```
>>> studentIds = {'knuth': 42.0, 'turing': 56.0, 'nash': 92.0}
>>> studentIds['turing']
56.0
>>> studentIds['nash'] = 'ninety-two'
>>> studentIds
{'knuth': 42.0, 'turing': 56.0, 'nash': 'ninety-two'}
>>> del studentIds['knuth']
>>> studentIds
{'turing': 56.0, 'nash': 'ninety-two'}
>>> studentIds['knuth'] = [42.0, 'forty-two']
>>> studentIds
{'knuth': [42.0, 'forty-two'], 'turing': 56.0, 'nash': 'ninety-two'}
>>> studentIds.keys()
['knuth', 'turing', 'nash']
>>> studentIds.values()
[[42.0, 'forty-two'], 56.0, 'ninety-two']
>>> studentIds.items()
[('knuth', [42.0, 'forty-two']), ('turing',56.0), ('nash', 'ninety-two')]
>>> len(studentIds)
3
```

Copiar

Al igual que con las listas anidadas, también puede crear diccionarios de diccionarios.

Ejercicio: Diccionarios

Use `dir` y `help` para conocer las funciones que puede llamar en los diccionarios.

Escritura de bucles

Ahora escribamos un script de Python simple que demuestre el `for` bucle de Python. Abre el archivo llamado `foreach.py`, que debe contener el siguiente código:

```
# This is what a comment looks like

fruits = ['apples', 'oranges', 'pears', 'bananas']

for fruit in fruits:

    print(fruit + ' for sale')


fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75}

for fruit, price in fruitPrices.items():

    if price < 2.00:

        print('%s cost %f a pound' % (fruit, price))

    else:

        print(fruit + ' are too expensive!')
```

Copiar

En la línea de comando, emplea el siguiente comando en el directorio que contiene `foreach.py`:

```
[xxxx ~/tutorial]$ python foreach.py

apples for sale

oranges for sale

pears for sale

bananas for sale

apples are too expensive!

oranges cost 1.500000 a pound

pears cost 1.750000 a pound
```

Copiar

Recuerda que los estados impresos que enumeran los costos pueden estar en un orden diferente en su pantalla que en este tutorial; eso se debe al hecho de que estamos recorriendo las claves del diccionario, que no están ordenadas. Para obtener más información sobre las estructuras de control (p. ej., `if` y `else`) en Python, consulta la sección del tutorial oficial de Python sobre este tema.
Si te gusta la programación funcional también te puede gustar `map` y `filter`:

```
>>> list(map(lambda x: x * x, [1, 2, 3]))

[1, 4, 9]

>>> list(filter(lambda x: x > 3, [1, 2, 3, 4, 5, 4, 3, 2, 1]))

[4, 5, 4]
```

Copiar

El siguiente fragmento de código demuestra la construcción de un list comprenhension de Python :

```
nums = [1, 2, 3, 4, 5, 6]

plusOneNums = [x + 1 for x in nums]

oddNums = [x for x in nums if x % 2 == 1]

print(oddNums)

oddNumsPlusOne = [x + 1 for x in nums if x % 2 == 1]

print(oddNumsPlusOne)
```

Copiar

Este código está en un archivo llamado `listcomp.py`, que puede ejecutar:

```
[xxxx ~]$ python listcomp.py

[1, 3, 5]

[2, 4, 6]
```

Copiar

Ejercicio: List comprenhension

Escribe una list comprenhension que, a partir de una lista, genere una versión en minúsculas de cada cadena que tenga una longitud superior a cinco. Puedes encontrar la solución en `listcomp2.py`.

¡Cuidado con la sangría!

A diferencia de muchos otros lenguajes, Python usa la sangría en el código fuente para la interpretación. Entonces, por ejemplo, para el siguiente script:

```
if 0 == 1:

    print('We are in a world of arithmetic pain')

print('Thank you for playing')
```

Copiar

dará salida: `Thank you for playing`
Pero si hubiéramos escrito el guión como

```
if 0 == 1:

    print('We are in a world of arithmetic pain')

    print('Thank you for playing')
```

Copiar

no habría salida. La moraleja de la historia: ¡cuidado con la sangría! Lo mejor es usar cuatro espacios para la sangría; eso es lo que usa el código del curso.

Tabulaciones vs espacios

Debido a que Python usa la sangría para la evaluación del código, necesita realizar un seguimiento del nivel de sangría en los bloques de código. Esto significa que si tu programa de Python pasa de usar tabulaciones como sangría a espacios como sangría, el intérprete de Python no podrá resolver la

ambigüedad del nivel de sangría y lanzar una excepción. Aunque el código se puede alinear visualmente en su editor de texto, Python "ve" un cambio en la sangría y lo más probable es que genere una excepción (o, en raras ocasiones, produzca un comportamiento inesperado).
Esto sucede más comúnmente cuando se abre un archivo de Python que usa un esquema de sangría que es opuesto al que usa su editor de texto (es decir, su editor de texto usa espacios y el archivo usa tabulaciones). Cuando escribes nuevas líneas en un bloque de código, habrá una combinación de tabulaciones y espacios, aunque el espacio en blanco esté alineado. Para una discusión más larga sobre tabulaciones versus espacios, visita esta discusión en StackOverflow.

Funciones de escritura

Al igual que en Java, en Python puedes definir tus propias funciones:

```
fruitPrices = {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75}


def buyFruit(fruit, numPounds):
    if fruit not in fruitPrices:
        print("Sorry we don't have %s" % (fruit))
    else:
        cost = fruitPrices[fruit] * numPounds
        print("That'll be %f please" % (cost))


# Main Function
if __name__ == '__main__':
    buyFruit('apples', 2.4)
    buyFruit('coconuts', 2)
```

Copiar

En lugar de tener una `main` función como en Java, la `__name__ == '__main__'` verificación se usa para delimitar expresiones que se ejecutan cuando se llama al archivo como un script desde la línea de comando. El código después de la verificación principal es, por lo tanto, el mismo tipo de código que pondría en una `main` función en Java.
Guarde este script como fruit.py y ejecútelo:

```
(cs188) [xxxx ~]$ python fruit.py
That'll be 4.800000 please
Sorry we don't have coconuts
```

Copiar

Ejercicio avanzado

Escribe una `quickSort` función en Python usando listas de comprensión. Utilice el primer elemento como pivote. Puedes encontrar la solución en `quickSort.py`.

P0-F1: Primera Parte

Conceptos básicos de objetos

Aunque esta no es una clase de programación orientada a objetos, tendrá que usar algunos objetos en los proyectos de programación, por lo que vale la pena cubrir los conceptos básicos de los objetos en Python. Un objeto encapsula datos y proporciona funciones para interactuar con esos datos.

Definición de clases

Aquí hay un ejemplo de cómo definir una clase llamada `FruitShop`:

```
class FruitShop:


  def __init__(self, name, fruitPrices):
    """
      name: Name of the fruit shop


      fruitPrices: Dictionary with keys as fruit
      strings and prices for values e.g.
      {'apples': 2.00, 'oranges': 1.50, 'pears': 1.75}
    """
    self.fruitPrices = fruitPrices
    self.name = name
    print('Welcome to %s fruit shop' % (name))


  def getCostPerPound(self, fruit):
    """
      fruit: Fruit string
    Returns cost of 'fruit', assuming 'fruit'
    is in our inventory or None otherwise
    """
    if fruit not in self.fruitPrices:
      return None
    return self.fruitPrices[fruit]


  def getPriceOfOrder(self, orderList):
    """
```

```
        orderList: List of (fruit, numPounds) tuples

    Returns cost of orderList, only including the values of
    fruits that this fruit shop has.
    """
    totalCost = 0.0
    for fruit, numPounds in orderList:
        costPerPound = self.getCostPerPound(fruit)
        if costPerPound != None:
            totalCost += numPounds * costPerPound
    return totalCost

def getName(self):
    return self.name
```

Copiar

La `FruitShop` clase tiene algunos datos, el nombre de la tienda y los precios por libra de alguna fruta, y proporciona funciones, o métodos, sobre estos datos. ¿Qué ventaja hay en envolver estos datos en una clase?

1. Encapsular los datos evita que se alteren o utilicen de manera inapropiada,
2. La abstracción que proporcionan los objetos facilita la escritura de código de propósito general.

Uso de objetos

Entonces, ¿cómo hacemos un objeto y lo usamos? Asegúrate de tener la `FruitShop` implementación en `shop.py`. Luego importamos el código de este archivo (haciéndolo accesible a otros scripts) usando `import shop`, ya que `shop.py` es el nombre del archivo. Entonces, podemos crear `FruitShop` objetos de la siguiente manera:

```
import shop


shopName = 'the Berkeley Bowl'
fruitPrices = {'apples': 1.00, 'oranges': 1.50, 'pears': 1.75}
berkeleyShop = shop.FruitShop(shopName, fruitPrices)
applePrice = berkeleyShop.getCostPerPound('apples')
print(applePrice)
print('Apples cost $%.2f at %s.' % (applePrice, shopName))


otherName = 'the Stanford Mall'
otherFruitPrices = {'kiwis': 6.00, 'apples': 4.50, 'peaches': 8.75}
```

```
otherFruitShop = shop.FruitShop(otherName, otherFruitPrices)
otherPrice = otherFruitShop.getCostPerPound('apples')
print(otherPrice)
print('Apples cost $%.2f at %s.' % (otherPrice, otherName))
print("My, that's expensive!")
```

Copiar

Este código está en `shopTest.py`; puedes ejecutarlo así:

```
[xxxx ~]$ python shopTest.py
Welcome to the Berkeley Bowl fruit shop
1.0
Apples cost $1.00 at the Berkeley Bowl.
Welcome to the Stanford Mall fruit shop
4.5
Apples cost $4.50 at the Stanford Mall.
My, that's expensive!
```

Copiar

Entonces, ¿qué acaba de pasar? La `import shop` declaración le dijo a Python que cargara todas las funciones y clases en `shop.py`. La línea `berkeleyShop = shop.FruitShop(shopName, fruitPrices)` construye una instancia de la `FruitShop` clase definida en shop.py llamando a la `__init__` función en esa clase. Ten en cuenta que solo pasamos dos argumentos, mientras que `__init__` parece tomar tres argumentos: `(self, name, fruitPrices)`. La razón de esto es que todos los métodos de una clase tienen `self` como primer argumento. El `self` valor de la variable se establece automáticamente en el objeto mismo; al llamar a un método, solo proporciona los argumentos restantes. La `self` variable contiene todos los datos (`name` y `fruitPrices`) para la instancia específica actual (similar a `this` en Java). Las declaraciones de impresión usan el operador de sustitución (descrito en los documentos de Python si tiene curiosidad).

Variables estáticas frente a instancias

El siguiente ejemplo ilustra cómo usar variables estáticas y de instancia en Python.
Crea el `person_class.py` que contiene el siguiente código:

```
class Person:
    population = 0

    def __init__(self, myAge):
        self.age = myAge
        Person.population += 1

    def get_population(self):
        return Person.population
```

```
    def get_age(self):
        return self.age
```

Copiar

Primero compilamos el script:

`[xxxx ~]$ python person_class.py`

Ahora usa la clase de la siguiente manera:

```
>>> import person_class
>>> p1 = person_class.Person(12)
>>> p1.get_population()
1
>>> p2 = person_class.Person(63)
>>> p1.get_population()
2
>>> p2.get_population()
2
>>> p1.get_age()
12
>>> p2.get_age()
63
```

Copiar

En el código anterior, `age` es una variable de instancia y `population` es una variable estática. `population` es compartido por todas las instancias de la `Person` clase, mientras que cada instancia tiene su propia `age` variable.

Más consejos y trucos de Python

Este tutorial se ha referido brevemente a algunos aspectos importantes de Python que serán relevantes para el curso. Aquí hay algunos datos más útiles:

- Emplea `range` para generar una secuencia de enteros, útil para generar `for` bucles indexados tradicionales:

```
for index in range(3):
    print(lst[index])
```

Copiar

- Después de importar un archivo, si editas un archivo fuente, los cambios no se propagarán inmediatamente en el intérprete. Para esto, emplea el `reload` comando:

```
>>> reload(shop)
```

Copiar

Solución de problemas

Estos son algunos problemas (y sus soluciones) que suelen encontrar los nuevos estudiantes de Python.

•**Problema:** ImportError: ningún módulo llamado py
Solución: para declaraciones de importación con `import <package-name>`, no incluyas la extensión del archivo (es decir, la `.py` cadena). Por ejemplo, debes usar: `import shop` NO: `import shop.py`

•**Problema:** NameError: el nombre 'MI VARIABLE' no está definido Incluso después de importar, puede ver esto.
Solución: para acceder a un miembro de un módulo, debe escribir `MODULE NAME.MEMBER NAME`, donde `MODULE NAME` es el nombre del `.py` archivo y `MEMBER NAME` es el nombre de la variable (o función) a la que intenta acceder.

•**Problema:** TypeError: el objeto 'dict' no se puede llamar
Solución: Las búsquedas en el diccionario se realizan usando corchetes: [y]. NO paréntesis: (y).

•**Problema:** ValueError: demasiados valores para desempaquetar
Solución: aseguraté de que la cantidad de variables que está asignando en un `for` ciclo coincida con la cantidad de elementos en cada elemento de la lista. Del mismo modo para trabajar con tuplas.
Por ejemplo, si `pair` es una tupla de dos elementos (p. ej `pair =('apple', 2.0)`.), el siguiente código provocaría el error "demasiados valores para desempaquetar":
`(a, b, c) = pair`
Aquí hay un escenario problemático que involucra un `for` bucle:

```
pairList = [('apples', 2.00), ('oranges', 1.50), ('pears', 1.75)]

for fruit, price, color in pairList:

  print('%s fruit costs %f and is the color %s' % (fruit, price, color))
```

Copiar

•**Problema:** AttributeError: el objeto 'lista' no tiene atributo 'longitud' (o algo similar)
Solución: Encontrar la longitud de las listas se hace usando `len(NAME OF LIST)`.

•**Problema:** los cambios en un archivo no surten efecto.
Solución:

1. Aseguraté de guardar todos sus archivos después de cualquier cambio.
2. Si está editando un archivo en una ventana diferente a la que está usando para ejecutar python, asegúrese `reload(_YOUR_MODULE_)` de garantizar que sus cambios se reflejen. `reload` funciona de manera similar a `import`.

Más referencias

•El lugar para obtener más información sobre Python: www.python.org
•Un buen libro de referencia: Aprendiendo Python

Autograduación

Para que se familiarice con el evaluador automático, le pediremos que codifique, pruebe y envíe soluciones para tres preguntas.
Puede descargar todos los archivos asociados con el tutorial de autograder como archivo zip: tutorial.zip (ten en cuenta que es **diferente** del archivo zip utilizado en los minitutoriales de UNIX y Python, python_basics.zip). Descomprime este archivo y examina su contenido:

```
[xxxx ~]$ unzip tutorial.zip
[xxxx ~]$ cd tutorial
[xxxx ~/tutorial]$ ls
addition.py
autograder.py
buyLotsOfFruit.py
grading.py
projectParams.py
shop.py
shopSmart.py
testClasses.py
testParser.py
test_cases
tutorialTestClasses.py
```

Copiar

Esto contiene una serie de archivos que editará o ejecutará:

- `addition.py`: archivo fuente para la pregunta 1
- `buyLotsOfFruit.py`: archivo fuente para la pregunta 2
- `shop.py`: archivo fuente para la pregunta 3
- `shopSmart.py`: archivo fuente para la pregunta 3
- `autograder.py`: script de autoevaluación (ver más abajo)

y otros que puedes ignorar:

- `test_cases`: el directorio contiene los casos de prueba para cada pregunta
- `grading.py`: código del calificador automático
- `testClasses.py`: código del calificador automático
- `tutorialTestClasses.py`: clases de prueba para este proyecto en particular
- `projectParams.py`: parámetros del proyecto

El comando `python autograder.py` califica su solución a los tres problemas. Si lo ejecutamos antes de editar cualquier archivo, obtenemos una página o dos de salida:

```
[xxxx ~/tutorial]$ python autograder.py
Starting on 1-21 at 23:39:51


Question q1
===========
*** FAIL: test_cases/q1/addition1.test
***     add(a, b) must return the sum of a and b
***     student result: "0"
```

```
***     correct result: "2"
*** FAIL: test_cases/q1/addition2.test
***     add(a, b) must return the sum of a and b
***     student result: "0"
***     correct result: "5"
*** FAIL: test_cases/q1/addition3.test
***     add(a, b) must return the sum of a and b
***     student result: "0"
***     correct result: "7.9"
*** Tests failed.

### Question q1: 0/1 ###


Question q2
===========
*** FAIL: test_cases/q2/food_price1.test
***     buyLotsOfFruit must compute the correct cost of the order
***     student result: "0.0"
***     correct result: "12.25"
*** FAIL: test_cases/q2/food_price2.test
***     buyLotsOfFruit must compute the correct cost of the order
***     student result: "0.0"
***     correct result: "14.75"
*** FAIL: test_cases/q2/food_price3.test
***     buyLotsOfFruit must compute the correct cost of the order
***     student result: "0.0"
***     correct result: "6.4375"
*** Tests failed.

### Question q2: 0/1 ###



Question q3
===========
```

```
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
*** FAIL: test_cases/q3/select_shop1.test
***     shopSmart(order, shops) must select the cheapest shop
***     student result: "None"
***     correct result: "<FruitShop: shop1>"
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
*** FAIL: test_cases/q3/select_shop2.test
***     shopSmart(order, shops) must select the cheapest shop
***     student result: "None"
***     correct result: "<FruitShop: shop2>"
Welcome to shop1 fruit shop
Welcome to shop2 fruit shop
Welcome to shop3 fruit shop
*** FAIL: test_cases/q3/select_shop3.test
***     shopSmart(order, shops) must select the cheapest shop
***     student result: "None"
***     correct result: "<FruitShop: shop3>"
*** Tests failed.

### Question q3: 0/1 ###

Finished at 23:39:51

Provisional grades
==================
Question q1: 0/1
Question q2: 0/1
Question q3: 0/1
------------------
Total: 0/3

Your grades are NOT yet registered.  To register your grades, make sure
```

```
to follow your instructor's guidelines to receive credit on your project.
```

Copiar

Para cada una de las tres preguntas, esto muestra los resultados de las pruebas de esa pregunta, la calificación de las preguntas y un resumen final al final. Debido a que aún no ha resuelto las preguntas, todas las pruebas fallan. A medida que resuelve cada pregunta, es posible que algunas pruebas pasen mientras que otras fallan. Cuando se aprueban todas las pruebas para una pregunta, se obtiene la máxima puntuación.
Si observa los resultados de la pregunta 1, puede ver que ha fallado tres pruebas con el mensaje de error "sumar (a, b) debe devolver la suma de a y b". La respuesta que da su código es siempre 0, pero la respuesta correcta es diferente. Lo arreglaremos en la siguiente pestaña.

Pregunta 1: Suma

Abre `addition.py` y mira la definición de `add`:

```
def add(a, b):
    "Return the sum of a and b"
    "*** YOUR CODE HERE ***"
    return 0
```

Copiar

Las pruebas llamaron a esto con `a` y `b` establecieron diferentes valores, pero el código siempre devolvió cero. Modifique esta definición para que diga:

```
def add(a, b):
    "Return the sum of a and b"
    print("Passed a = %s and b = %s, returning a + b = %s" % (a, b, a + b))
    return a + b
```

Copiar

Ahora vuelve a ejecutar el calificador automático (omitiendo los resultados de las preguntas 2 y 3):

```
[xxxx ~/tutorial]$ python autograder.py -q q1
Starting on 1-21 at 23:52:05


Question q1
===========
Passed a = 1 and b = 1, returning a + b = 2
*** PASS: test_cases/q1/addition1.test
***     add(a, b) returns the sum of a and b
Passed a = 2 and b = 3, returning a + b=5
*** PASS: test_cases/q1/addition2.test
***     add(a, b) returns the sum of a and b
```

```
Passed a = 10 and b = -2.1, returning a + b = 7.9

*** PASS: test_cases/q1/addition3.test

***     add(a, b) returns the sum of a and b



### Question q1: 1/1 ###


Finished at 23:41:01


Provisional grades

==================

Question q1: 1/1

Question q2: 0/1

Question q3: 0/1

------------------

Total: 1/3
```

Copiar

Ahora aprueba todas las pruebas y obtiene la máxima puntuación en la pregunta 1. Observe las nuevas líneas "Passed a=..." que aparecen antes de "*** PASS: ...". Estos son producidos por la declaración de impresión en `add`. Puede usar declaraciones de impresión como esa para generar información útil para la depuración.

Pregunta 2: función comprarMuchasFrutas

Agregua una función `buyLotsOfFruit(orderList)` a `buyLotsOfFruit.py` que tome una lista de `(fruit,pound)` tuplas y devuelva el costo de su lista. Si hay alguno `fruit` en la lista que no aparece en `fruitPrices` ella, debe imprimir un mensaje de error y devolver `None`. Por favor, no cambies la variable `fruitPrices`.

Continúa `python autograder.py` hasta que la pregunta 2 pase todas las pruebas y obtenga la máxima puntuación. Cada prueba confirmará que `buyLotsOfFruit(orderList)` devuelve la respuesta correcta dadas varias entradas posibles. Por ejemplo, `test_cases/q2/food_price1.test` prueba si:
`Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25`

Pregunta 3: función shopSmart

Completa la función `shopSmart(orders,shops)` e `shopSmart.py`, que toma un `orderList` (como el tipo pasado a `FruitShop.getPriceOfOrder`) y una lista de `FruitShop` y devuelve `FruitShop` donde su pedido

cuesta la menor cantidad en total. No cambies el nombre del archivo o los nombres de las variables, por favor. Ten en cuenta que proporcionaremos la `shop.py` implementación como un archivo de "soporte". Continúa `python autograder.py` hasta que la pregunta 3 pase todas las pruebas y obtengas la máxima puntuación. Cada prueba confirmará que `shopSmart(orders,shops)` devuelve la respuesta correcta dadas varias entradas posibles. Por ejemplo, con las siguientes definiciones de variables:

```
orders1 = [('apples', 1.0), ('oranges', 3.0)]

orders2 = [('apples', 3.0)]

dir1 = {'apples': 2.0, 'oranges': 1.0}

shop1 =  shop.FruitShop('shop1',dir1)

dir2 = {'apples': 1.0, 'oranges': 5.0}

shop2 = shop.FruitShop('shop2', dir2)

shops = [shop1, shop2]
```

Copiar

`test_cases/q3/select_shop1.test` prueba si: `shopSmart.shopSmart(orders1, shops) == shop1`
y `test_cases/q3/select_shop2.test` comprueba si: `shopSmart.shopSmart(orders2, shops) == shop2`

P0-F2: Segunda Parte

reglas adaptadas de http://acanomas.com/Reglamentos-Juegos-de-Naipes/951/Mona.html

1.Introducción

```
Origen: España
Duración de la partida: 10-15 minutos
Baraja: Española
N° de naipes: 40
Jugadores: 2 o más
Dificultad: Fácil
Tipo de juego: emparejar cartas
```

La mona es uno de los primeros juegos que aprenden los niños, ya que su sencillez y la claridad de su objetivo lo hacen apto para que todos los miembros de la familia puedan divertirse juntos.

2.Objetivo

No ser el último en desembarazarse de todas las cartas, formando parejas del mismo índice.

3.Número de Jugadores

En una partida de mona puede participar cualquier número de jugadores a partir de dos.

4.Tipo de Baraja

Para jugar a la mona puede servir cualquier baraja que tenga, como máximo, una carta desparejada. Lo más corriente es tomar una baraja española y separar una de sus cartas para que quede desparejada.

5.Valor de las Cartas
En el juego de la mona las cartas no tienen ningún valor. Si se juega con una baraja, se consideran pareja una **copa** y un **oro** que tengan el mismo índice o una **espada** y un **basto** que tengan el mismo índice.

6.Vocabulario Específico

- Mona: Carta que se ha retirado. Eso implica que habrá una carta sin pareja al finalizar el juego.
- Pareja: Par de cartas del mismo índice donde coincidan oros y copas o espadas y bastos.

7.Las Jugadas de la Mona
En la mona sólo hay una jugada: formar parejas de cartas del mismo índice entre oros-copas o espada-bastos.

Cuando un jugador forma una pareja de cartas elimina de su mano esas cartas.

8.Inicio
En la mona se reparten las cartas entre todos los jugadores, y no tiene ninguna importancia quién es el dador ni cuántas cartas recibe cada uno, ya que lo importante son las cartas que le quedan a cada jugador después de haberse desprendido de todas las parejas que ha formado con las cartas del reparto inicial.

1.- Para iniciar el juego es necesario preparar una baraja de modo que una de sus cartas quede desparejada. Por eso después de barajar se suele retirar la primera carta de la baraja antes de su reparto.

2.- Una vez repartidas todas las cartas entre los jugadores, se comenzará por una ronda inicial (el método jugar_primera_ronda de la clase ListaJugadores) especial en la que cada jugador se descartará de aquellas parejas que le hayan tocado en suerte.

3.- Se establece de manera aleatoria el turno, es decir, quién comenzará a jugar.

4.- Repetir salir si solo queda un jugador:

4.1 El jugador con el turno decide a quien le quiere solicitar una carta y de manera aleatoria se la roba (si el jugador es un jugador IA seleccionará el jugador al azar)
4.2 El jugador con el turno comprueba si ha podido hacer pareja.
4.2.1 si es el caso se descarta de la misma y sin pasar el turno se juega otra ronda (en el caso del jugadorIA la formación de parejas se hará de forma automática, para un jugadorPersona, el jugador deberá de indicar a través de los indices de las cartas, cuales forman pareja si la hubiera)
4.2.2 sino se actualiza el turno y se juega otra ronda

Se pide:

0.- Leer y entender el código
1.- Ahora mismo el juego está implementado tal que cuando un jugador se quede sin cartas el juego finalice. En el juego original, se sigue jugando hasta que solo queda un jugador, que será aquel que posea la pareja de la Mona, es decir, la única carta que no se puede emparejar. Modificar el método principal y/o el método def jugar_primera_ronda(self): de la clase ListaJugadores para llevar a cabo la nueva versión.

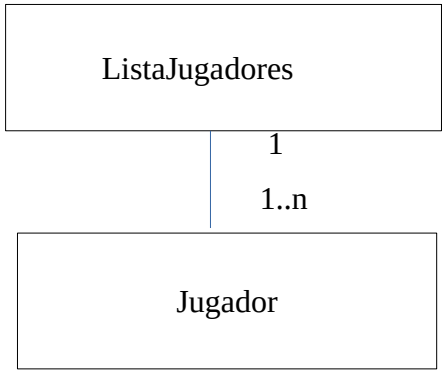
2.- Implementar el método, def solicitar_a_que_jugador(self,idxJugActual,numJugadores): de la clase JugadorIA, tal que de manera random obtenga el jugador al cual la máquina le querrá robar una carta

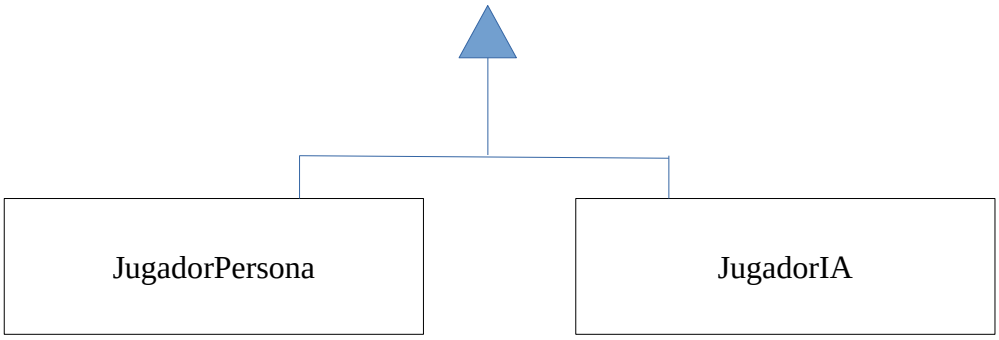
3.- Implementar los métodos de ManoIA y ManoPersona el método eliminar_parejas

---

Se dispone de dos programas: JuegoCartas.py y Cartas.py

Clases en JuegoCartas.py:





Clase ListaJugadores

atributos:

jugadores: [] #contendrá la lista de los jugadores

numJug: 0 #contendrá el número de jugadores

turno: 0 #el índice del jugador actual de 0 a numJug-1

métodos:

def __init__(self):

def pasar_turno(self):

def imprimir_nombre(self,idx):

def inicializar_partida(self,n_jugadores=2):

def jugar_primera_ronda(self):

def inicializar_turno(self):

def jugar_ronda(self):

def repartir_cartas(self, baraja):

Clase Jugador

atributos:

jugador: String #contendrá el nombre del jugador

métodos:

def __init__(self, jugador='Ordenador'):

def imprimir_nombre(self):

def solicitar_carta_al_jugadorX(self,numCartas):

def eliminar_parejas(self):

def eliminar_carta(self,carta):

def tiene_cartas(self):

def cuantas_cartas_quedan(self):

def anadir_carta(self,carta):

Clase JugadorIA(Jugador)

atributos:

mano: ManoIA

métodos:

def __init__(self, jugador='Ordenador'):

def solicitar_a_que_jugador(self,idxJugActual,numJugadores):

def imprimir(self):

Clase JugadorPersona(Jugador)
atributos:

mano: ManoPersona

métodos:

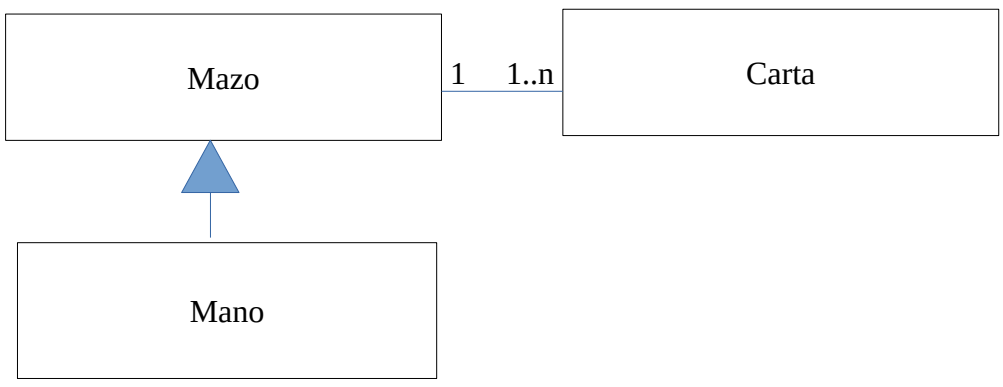
def __init__(self, jugador='Ordenador'):

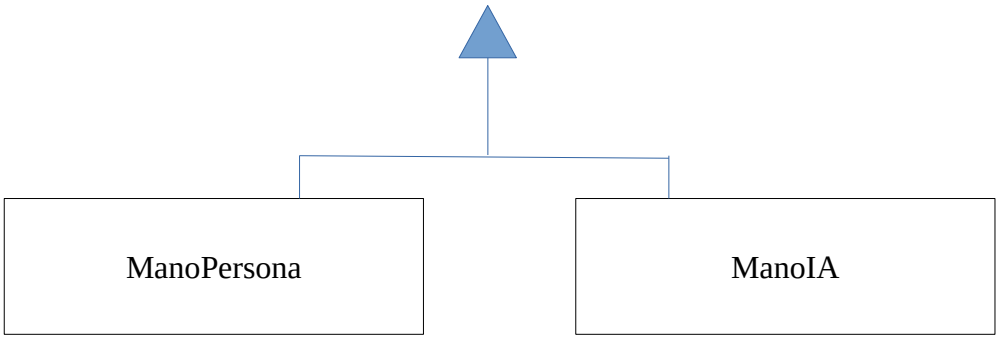
def solicitar_a_que_jugador(self,idxJugActual,numJugadores):

def imprimir(self):

Clases en Cartas.py

Clases en Cartas.py:





Clase Carta

atributos:

palo: integer 0-3

rango: integer 1-13

métodos:

def __init__(self, palo=0, rango=2):

def __str__(self):

def __cmp__(self, otra):

def __eq__(self, otra):

def __ne__(self, otra):

Clase Mazo

atributos:

cartas: #una lista de cartas

métodos:

def __init__(self):

def __str__(self):

def inicializar_baraja(self):

def solicitar_carta(self, idx):

def anadir_carta(self, carta):

def eliminar_carta(self, carta):

def esta_carta(self, carta):

def pop_carta(self, i=-1):

def barajar(self):

def ordenarAsc(self):

def mover_cartas(self, mano, num):

def imprimir(self):

def esta_vacio(self):

def apartar_mona(self):

def dar_carta(self):

def cuantas_cartas_quedan(self):

def eliminar_cartaIdx(self, idx):

Clase Mano(Mazo)

métodos:

def __init__(self):

def imprimir(self):

Clase ManoPersona(Mano):

def __init__(self):

def eliminar_parejas(self):

Class ManoIA(Mano):

```
#####################################
# Forman pareja:
# numX de bastos y numX de espadas
# numX de copas  y numx de oros
#####################################
```

def __init__(self):

def eliminar_parejas(self):