

12 – TIPOS PROPIOS

Jorge Muñoz
IIC1103 – Introducción a la Programación

RPG CHARACTER



INTERESES

“videojuegos”

“videojuegos, videojuegos, videojuegos”

“juegos. RPG”



Situación
Real





Situación
Real

J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25

Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25

Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25

Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14

Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1

Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###



Clase: **WARRIOR**
Vida: **50**
Ataque: **20-40**



Clase: **WIZARD**
Vida: **30**
Ataque: **22-42**



Python tiene el tipo **RPGCharacter** (Ni instalar ni import)

```
#Crear una lista vacia
listal = []

#Agregar un elemento al final
listal.append("cosa")

#Quita y retorna el elemento en la posicion 8
elem = listal.pop(8)
```

```
#Crear un personaje
#Necesita un nombre y un tipo (WARRIOR o WIZARD)
#Si es WARRIOR tendra 50 puntos de vida inicial
#Si es WIZARD tendra 30 puntos de vida inicial
personajel = RPGCharacter("Gandalf","WIZARD")

#Infligir dano al personaje
#Necesita los puntos de dano
personajel.damage(20)

#Retorna un entero con los puntos de vida
#actuales del personaje
vida1 = personajel.life()

#Retorna un entero con los puntos de un ataque
#Numero al azar entre el min y el max del poder
#Si es WARRIOR sera entre 20 y 40
#Si es WIZARD sera entre 22 y 42
at1 = personajel.attack()
```





I'LL WAIT
FOR YOU HERE



```
J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25
###
Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25
###
Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25
###
Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14
###
Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1
###
Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###
```

```
#Crear un personaje
#Necesita un nombre y un tipo (WARRIOR o WIZARD)
#Si es WARRIOR tendra 50 puntos de vida inicial
#Si es WIZARD tendra 30 puntos de vida inicial
personaje1 = RPGCharacter("Gandalf", "WIZARD")

#Infligir dano al personaje
#Necesita los puntos de dano
personaje1.damage(20)

#Retorna un entero con los puntos de vida
#actuales del personaje
vidal = personaje1.life()

#Retorna un entero con los puntos de un ataque
#Numero al azar entre el min y el max del poder
#Si es WARRIOR sera entre 20 y 40
#Si es WIZARD sera entre 22 y 42
atl1 = personaje1.attack()
```



```
#Creo Jugador 1
n1 = input("J1: Nombre? ")
t1 = input("J1: WARRIOR or WIZARD? ")
c1 = RPGCharacter(n1,t1)
```

... .

```
J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25
###
Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25
###
Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25
###
Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14
###
Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1
###
Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###
```



```
#Creo Jugador 1
n1 = input("J1: Nombre? ")
t1 = input("J1: WARRIOR or WIZARD? ")
c1 = RPGCharacter(n1,t1)

#Creo Jugador 2
n2 = input("J2: Nombre? ")
t2 = input("J2: WARRIOR or WIZARD? ")
c2 = RPGCharacter(n2,t2)

#Let them fight!
continuar = True
while continuar:
```

...
...

```
J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25
###
Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25
###
Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25
###
Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14
###
Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1
###
Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###
```



```
#Creo Jugador 1
n1 = input("J1: Nombre? ")
t1 = input("J1: WARRIOR or WIZARD? ")
c1 = RPGCharacter(n1,t1)

#Creo Jugador 2
n2 = input("J2: Nombre? ")
t2 = input("J2: WARRIOR or WIZARD? ")
c2 = RPGCharacter(n2,t2)

#Let them fight!
continuar = True
while continuar:
    a1 = c1.attack()
    a2 = c2.attack()
    print(n1,"con",a1,"y",n2,"con",a2)
```

...
...

```
J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25
###
Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25
###
Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25
###
Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14
###
Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1
###
Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###
```

```

#Creo Jugador 1
n1 = input("J1: Nombre? ")
t1 = input("J1: WARRIOR or WIZARD? ")
c1 = RPGCharacter(n1,t1)

#Creo Jugador 2
n2 = input("J2: Nombre? ")
t2 = input("J2: WARRIOR or WIZARD? ")
c2 = RPGCharacter(n2,t2)

#Let them fight!
continuar = True
while continuar:
    a1 = c1.attack()
    a2 = c2.attack()
    print(n1,"con",a1,"y",n2,"con",a2)

    #Jugador 1 Gana
    if a1 >= a2:
        d = a1-a2
        print(n2,"pierde",d,"puntos")
        c2.damage(d)
        print("--VIDA--",n1,c1.life(),n2,c2.life())
        if c2.life() <= 0:
            print(n2,"MUERE")
            continuar = False
    
```

...

```

J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25
###
Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25
###
Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25
###
Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14
###
Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1
###
Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###
```



```
#Creo Jugador 1
n1 = input("J1: Nombre? ")
t1 = input("J1: WARRIOR or WIZARD? ")
c1 = RPGCharacter(n1,t1)

#Creo Jugador 2
n2 = input("J2: Nombre? ")
t2 = input("J2: WARRIOR or WIZARD? ")
c2 = RPGCharacter(n2,t2)

#Let them fight!
continuar = True
while continuar:
    a1 = c1.attack()
    a2 = c2.attack()
    print(n1,"con",a1,"y",n2,"con",a2)

    #Jugador 1 Gana
    if a1 >= a2:
        d = a1-a2
        print(n2,"pierde",d,"puntos")
        c2.damage(d)
        print("--VIDA--",n1,c1.life(),n2,c2.life())
        if c2.life() <= 0:
            print(n2,"MUERE")
            continuar = False

    #Jugador 2 gana
    if a1 < a2:
        d = a2 - a1
        print(n1,"pierde",d,"puntos")
        c1.damage(d)
        print("--VIDA--",n1,c1.life(),n2,c2.life())
        if c1.life() <= 0:
            print(n1,"MUERE")
            continuar = False

    print("###")
```

```
J1: Nombre? Martigan
J1: WARRIOR or WIZARD? WARRIOR
J2: Nombre? Willow
J2: WARRIOR or WIZARD? WIZARD
Martigan con 37 y Willow con 32
Willow pierde 5 puntos
--VIDA-- Martigan 50 Willow 25
###
Martigan con 31 y Willow con 41
Martigan pierde 10 puntos
--VIDA-- Martigan 40 Willow 25
###
Martigan con 38 y Willow con 41
Martigan pierde 3 puntos
--VIDA-- Martigan 37 Willow 25
###
Martigan con 37 y Willow con 26
Willow pierde 11 puntos
--VIDA-- Martigan 37 Willow 14
###
Martigan con 40 y Willow con 27
Willow pierde 13 puntos
--VIDA-- Martigan 37 Willow 1
###
Martigan con 29 y Willow con 25
Willow pierde 4 puntos
--VIDA-- Martigan 37 Willow -3
Willow MUERE
###
```



Situación
Real



Tengo que confesaros algo ...

El tipo RPGCharacter
no existe :S



Pero esta semana
veremos como crearlo



12 – TIPOS PROPIOS

Jorge Muñoz
IIC1103 – Introducción a la Programación

TIPOS PROPIOS

int

bool

str

list





- Crearlos nosotros mismos
- Técnica:
Programación
Orientada a
Objetos (OO)

¿Qué necesitamos?

- Usar tipos propios
- Crear una nueva variable de ese tipo
- Definir tipos propios

USAR TIPOS PROPIOS



¿Dame tu 3r elemento?



¿Dame la nota de su I2?

list

```
x = lista1[3]
```

¿Dame tu 3r elemento?



i1
(float)

i2
(float)

ex
(float)

- Se llaman **ATRIBUTOS**
- Se accede `x.atributo`
 - Sin paréntesis, no es una función



i1
(float)

i2
(float)

ex
(float)

```
bob = # ... bob es una variable de tipo Estudiante
nota = (bob.i1 * 0.25) + (bob.i2 * 0.25) + (bob.ex * 0.5)
print("La nota de Bob es", nota)
```

- Se llaman **ATRIBUTOS**
- Se accede `x.atributo`
 - Sin paréntesis, no es una función

```
def calcular_nota(e):
    n = (e.i1 * 0.25) + (e.i2 * 0.25) + (e.ex * 0.5)
    return n
```



i1
(float)

i2
(float)

ex
(float)

- Se llaman **ATRIBUTOS**
- Se accede `x.atributo`
 - Sin paréntesis, no es una función

CARTAS



INTERESES

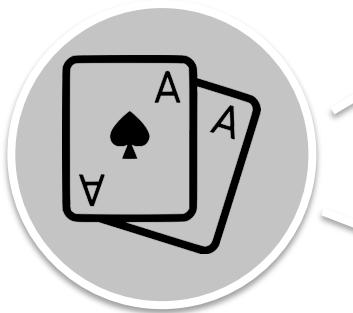
“cartas”

“Juegos de cartas”

“Poker, cartas, casinos”

“Magic, cartas”

```
def quien_gana(c1,c2):
```



num
(int)

pinta
(str)

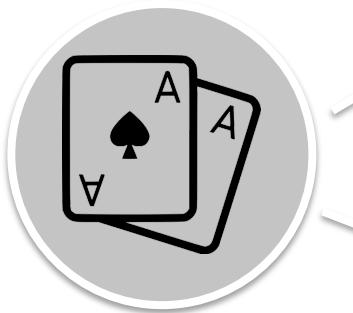
- Retorna **1, 2 o 0**, si gana c1, c2, o empatan, respectivamente
- Gana si tiene el número es **más grande**
- Si tienen el mismo número, gana el que sea pinta “**rojo**” (solo hay “verde” y “rojo”)
- Si tienen el mismo número y misma pinta, **empatan**





I'LL WAIT
FOR YOU HERE

```
def quien_gana(c1,c2):
```



num
(int)

pinta
(str)

- Retorna **1, 2 o 0**, si gana c1, c2, o empatan, respectivamente
- Gana si tiene el número es **más grande**
- Si tienen el mismo número, gana el que sea pinta “**rojo**” (solo hay “verde” y “rojo”)
- Si tienen el mismo número y misma pinta, **empatan**



```
def quien_gana(c1,c2):
    if c1.num > c2.num:
        r = 1
    elif c1.num < c2.num:
        r = 2
    elif c1.pinta == c2.pinta:
        r = 0
    elif c1.pinta == "rojo":
        r = 1
    else:
        r = 2
    return r
```



num
(int)

pinta
(str)

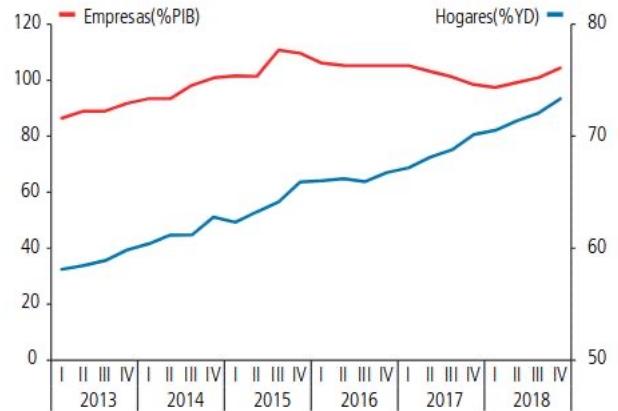
DESIGUALDAD

Jueves 18 abril de 2019 | Publicado a las 10:26

Riqueza de hogares chilenos disminuye y endeudamiento de las familias registra un máximo histórico

GRÁFICO 3

Deuda sector Hogares y Empresas no financieras
(porcentaje del PIB y del ingreso disponible)



Fuente: Banco Central de Chile.

```
# Informacion de hogares sacada  
# de alguna base de datos publica  
# como datos.gov.cl
```

```
hogares = # list de Hogar
```



```
print(num, "de", len(hogares), "hogares tienen superior a 500")
```

- Cuantos hogares tienen un endeudamiento por miembro igual o superior a 500
- El endeudamiento total del hogar se divide por su numero de miembros (ms).
- Endeudamiento es la diferencia entre los gastos (gas) y los ingresos (ing)



ms
(int)

ing
(int)

gas
(int)







I'LL WAIT
FOR YOU HERE

```
# Informacion de hogares sacada  
# de alguna base de datos publica  
# como datos.gov.cl
```

```
hogares = # list de Hogar
```



```
print(num, "de", len(hogares), "hogares tienen superior a 500")
```

- Cuantos hogares tienen un endeudamiento por miembro igual o superior a 500
- El endeudamiento total del hogar se divide por su numero de miembros (ms).
- Endeudamiento es la diferencia entre los gastos (gas) y los ingresos (ing)



ms
(int)

ing
(int)

gas
(int)





Situación
Real

```
# Informacion de hogares sacada
# de alguna base de datos publica
# como datos.gov.cl

hogares = # list de Hogar

num = 0
for h in hogares:
    ep = (h.gas-h.ing) / h.ms
    if ep >= 500:
        num += 1

print(num, "de", len(hogares), "hogares tienen superior a 500")
```



ms
(int)

ing
(int)

gas
(int)

- Cuantos hogares tienen un endeudamiento por miembro igual o superior a 500
- El endeudamiento total del hogar se divide por su numero de miembros (ms).
- Endeudamiento es la diferencia entre los gastos (gas) y los ingresos (ing)



• Open Data

- https://en.wikipedia.org/wiki/Open_data
- <http://datos.gob.cl/> (Gobierno de Chile)
- <http://datos.bcn.cl/> (Congreso Nacional de Chile)
- <http://opendata.congreso.cl/> (Senado de Chile)
- <http://data.worldbank.org/> (Banco Mundial)

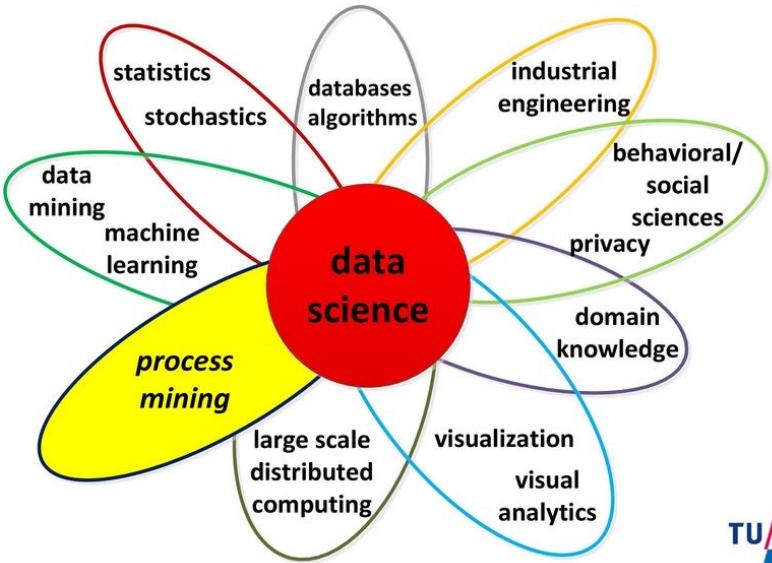


Data Scientist: The Sexiest Job of the 21st Century

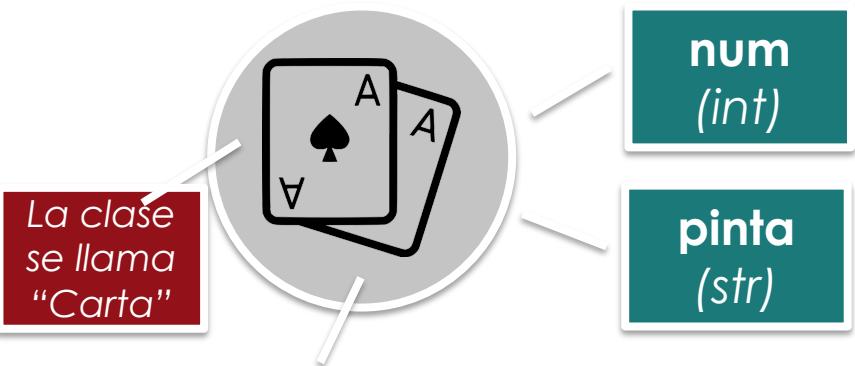
by Thomas H. Davenport and D.J. Patil

• Data Science

- Data Science: The Sexiest Job of the 21st century
<https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century/>
- Data Scientist: The Engineer of the Future
http://www.springer.com/cda/content/document/cda_downloaddocument/9783319049472-c2.pdf?SGWID=0-0-45-1444234-p176571711



CREAR TIPOS PROPIOS



(Y para crearse digo que necesita 2 parámetros: el color y el numero)

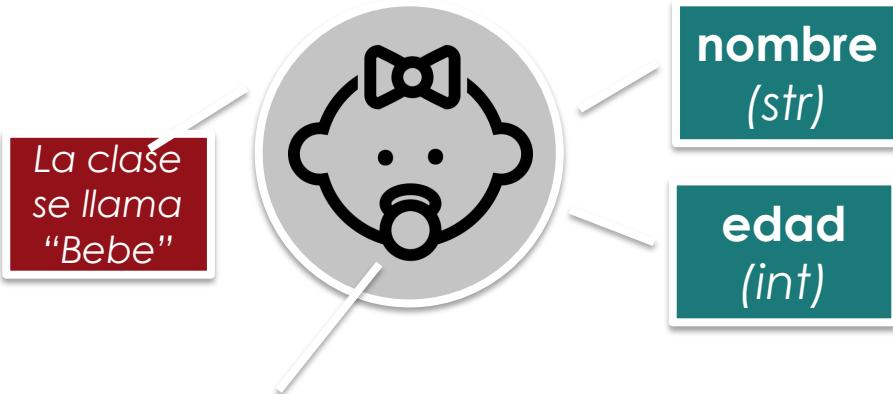
```
r5 = Carta("rojo",5)
print(r5.pinta)
print(r5.num)

v6 = Carta("verde",6)
print(v6.pinta)
print(v6.num)
```

>>>
rojo
5
verde
6
>>>

- **CREAR:**

- Para crear una **INSTANCIA** de un tipo, se pone el nombre de la **CLASE** y sus **PARÁMETROS** entre paréntesis
- Los parámetros son la **información que le pasamos** para que pueda crear una instancia del tipo que queremos



Y para crearse digo que necesita solo 1 parámetros: el nombre numero (es obvio que un bebe tendrá 0 años)

```
a = Bebe("Alicia")
print(a.nombre)
print(a.edad)
```

```
b = Bebe("Bob")
print(b.nombre)
print(b.edad)
```

```
>>>
Alicia
0
Bob
0
>>>
```

- No confundir **parámetros** (información que le paso para crear una instancia de ese tipo) con **atributos** (los componentes que tienen ese tipo)
- NO tienen por que haber el **mismo número** de parámetros y atributos

CURSO INTRO



La clase se
llama
“CursolIntro”

Y para crearse digo que necesita solo 3
parámetros: el promedio de la I1, I2, y Ex (el
nombre del curso es obvio)

```
#Pregunta (por input) los promedios
pil = float(input("Promedio I1: "))
pi2 = float(input("Promedio I2: "))
pex = float(input("Promedio Ex: "))
```

```
#Crea una variable de tipo Curso Intro
```

```
pf = c.i1*0.25+c.i2*0.25+c.ex*0.5
print("El promedio final de",c.nom,"es",pf)
```

>>>

```
Promedio I1: 4
Promedio I2: 4
Promedio Ex: 4
```

```
El promedio final de IntroProgra es 4.0
```

>>>

i1
(float)

i2
(float)

ex
(float)

nom
(str)





Situación
Real

La clase se llama "CursoIntro". Y para crearse digo que necesita solo 3 parámetros: el promedio de la I1, I2, y Ex (el nombre del curso es obvio)

```
#Pregunta (por input) los promedios
p1 = float(input("Promedio I1: "))
p2 = float(input("Promedio I2: "))
pex = float(input("Promedio Ex: "))
```

```
#Crea una variable de tipo Curso Intro
c = CursoIntro(p1,p2,pex)
```

```
pf = c.i1*0.25+c.i2*0.25+c.ex*0.5
print("El promedio final de",c.nom,"es",pf)
```

>>>

```
Promedio I1: 4
Promedio I2: 4
Promedio Ex: 4
```

```
El promedio final de IntroProgra es 4.0
```

>>>

i1
(float)

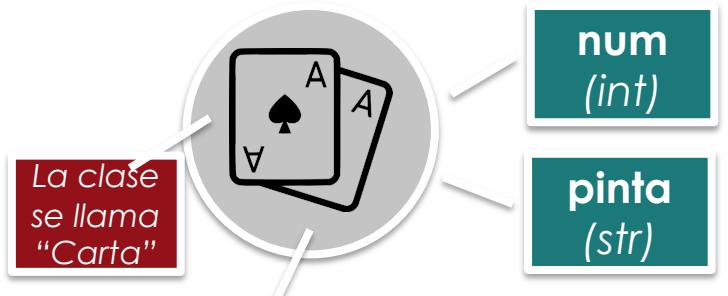
i2
(float)

ex
(float)

nom
(str)



DEFINIR TIPOS PROPIOS

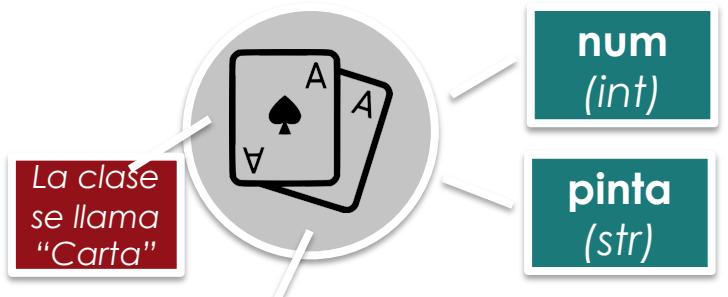


(Y para crearse digo que necesita
2 parámetros: el color y el
numero)

...

```
r5 = Carta("rojo",5)
print(r5.pinta)
print(r5.num)

v6 = Carta("verde",6)
print(v6.pinta)
print(v6.num)
```



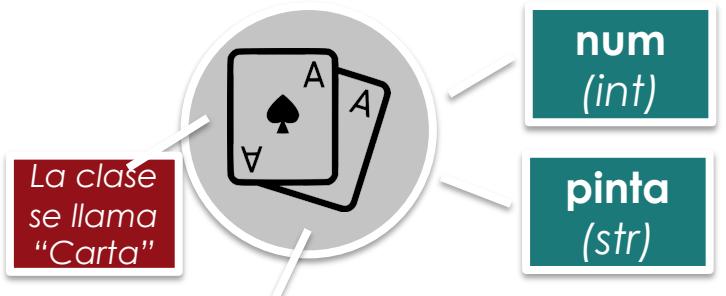
(Y para crearse digo que necesita 2 parámetros: el color y el numero)

class Carta:

...

```
r5 = Carta("rojo",5)
print(r5.pinta)
print(r5.num)

v6 = Carta("verde",6)
print(v6.pinta)
print(v6.num)
```

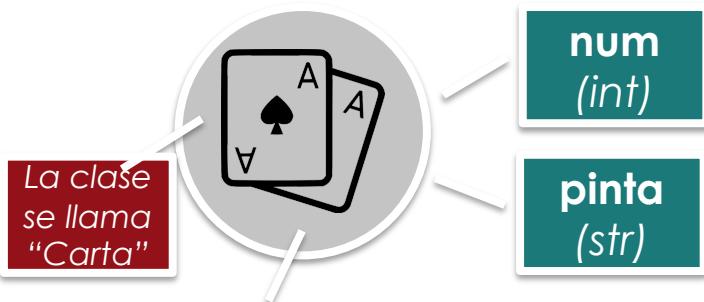


```
class Carta:  
    def __init__(self,p,n):
```

...

```
r5 = Carta("rojo",5)  
print(r5.pinta)  
print(r5.num)
```

```
v6 = Carta("verde",6)  
print(v6.pinta)  
print(v6.num)
```



(Y para crearse digo que necesita 2 parámetros: el color y el numero)

```
class Carta:  
    def __init__(self,p,n):  
        self.pinta =  
        self.num = i ...
```

```
r5 = Carta("rojo",5)  
print(r5.pinta)  
print(r5.num)
```

```
v6 = Carta("verde",6)  
print(v6.pinta)  
print(v6.num)
```

La clase
se llama
"Carta"



num
(int)

pinta
(str)

(Y para crearse digo que necesita
2 parámetros: el color y el
numero)

```
class Carta:  
    def __init__(self,p,n):  
        self.pinta = p  
        self.num = n  
  
r5 = Carta("rojo",5)  
print(r5.pinta)  
print(r5.num)  
  
v6 = Carta("verde",6)  
print(v6.pinta)  
print(v6.num)
```

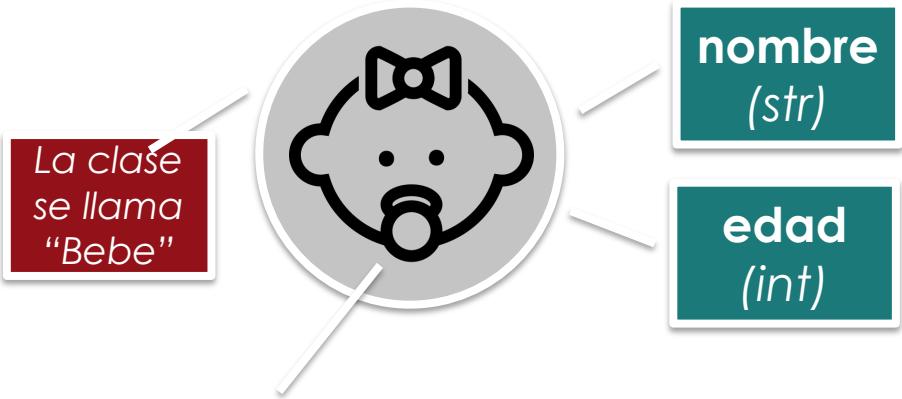
- **DEFINIR:**

- **class nombre**
- Constructor: **__init__**
- Constructor siempre lleva un parámetro **self** (que nunca esta en la llamada)
- Atributos se crean con **self.atributo** y se les da un **valor inicial**

...

```
a = Bebe("Alicia")
print(a.nombre)
print(a.edad)

b = Bebe("Bob")
print(b.nombre)
print(b.edad)
```



- NO confundir **atributos** (los componentes de un tipo) y **parámetros** (la información que le pasan para crearlo)
 - No tienen por que llamarse igual, ni ser el mismo número, ...

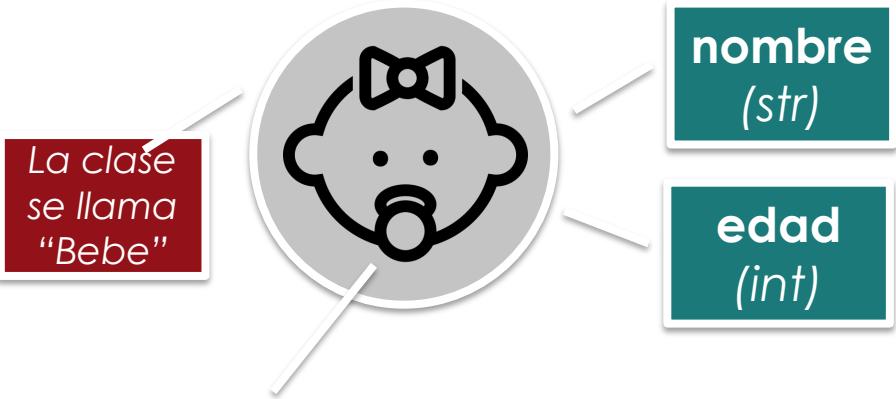


```
class Bebe:
```

```
    ...
```

```
a = Bebe("Alicia")
print(a.nombre)
print(a.edad)

b = Bebe("Bob")
print(b.nombre)
print(b.edad)
```



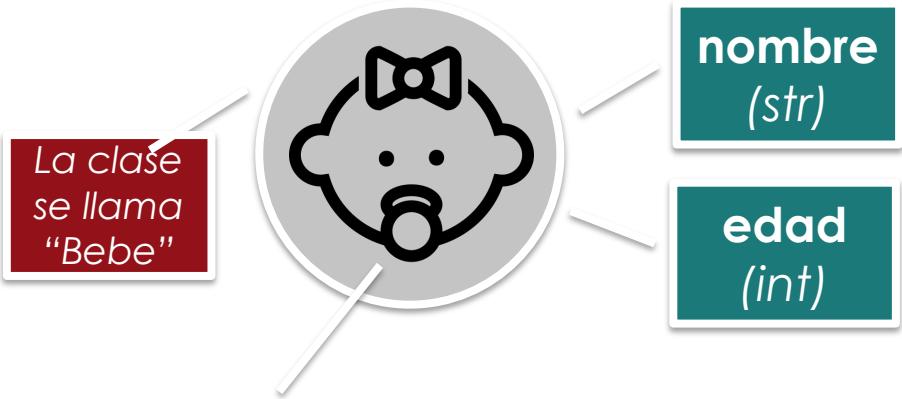
Y para crearse digo que necesita solo 1 parámetros: el nombre numero (es obvio que un bebe tendrá 0 años)

- NO confundir **atributos** (los componentes de un tipo) y **parámetros** (la información que le pasan para crearlo)
 - No tienen por que llamarse igual, ni ser el mismo número, ...



```
class Bebe:  
    def __init__(self, n):  
        ...
```

```
a = Bebe("Alicia")  
print(a.nombre)  
print(a.edad)  
  
b = Bebe("Bob")  
print(b.nombre)  
print(b.edad)
```



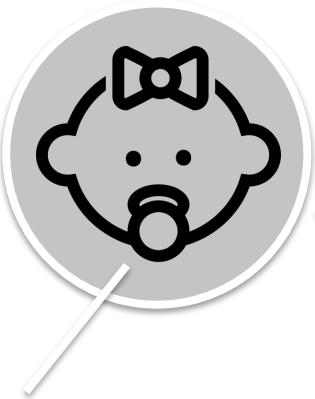
Y para crearse digo que necesita solo 1 parámetros: el nombre numero (es obvio que un bebe tendrá 0 años)

- NO confundir **atributos** (los componentes de un tipo) y **parámetros** (la información que le pasan para crearlo)
 - No tienen por que llamarse igual, ni ser el mismo número, ...



```
class Bebe:  
    def __init__(self,n):  
        self.nombre  
        self.edad = ...
```

La clase
se llama
“Bebe”



nombre
(str)

edad
(int)

Y para crearse digo que necesita solo 1
parámetros: el nombre numero (es obvio
que un bebe tendrá 0 años)

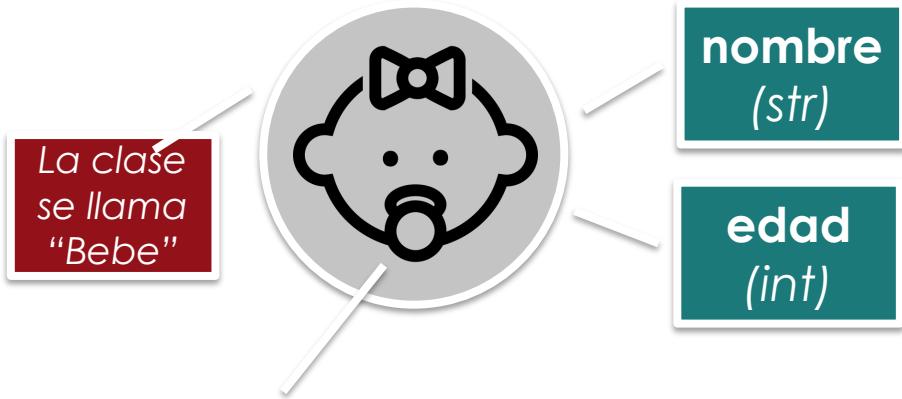
```
a = Bebe("Alicia")  
print(a.nombre)  
print(a.edad)
```

```
b = Bebe("Bob")  
print(b.nombre)  
print(b.edad)
```

- NO confundir **atributos** (los componentes de un tipo) y **parámetros** (la información que le pasan para crearlo)
 - No tienen por que llamarse igual, ni ser el mismo número, ...



```
class Bebe:  
    def __init__(self,n):  
        self.nombre = n  
        self.edad = 0  
  
a = Bebe("Alicia")  
print(a.nombre)  
print(a.edad)  
  
b = Bebe("Bob")  
print(b.nombre)  
print(b.edad)
```



Y para crearse digo que necesita solo 1 parámetros: el nombre numero (es obvio que un bebe tendrá 0 años)

- NO confundir **atributos** (los componentes de un tipo) y **parámetros** (la información que le pasan para crearlo)
 - No tienen por que llamarse igual, ni ser el mismo número, ...

SALGO DE



#Llego en



Comenzar Viaje

Hola! ¿Cómo están? Les cuento que estamos desarrollando un app móvil para aportar un granito de arena en todo este caos nacional. La app se llama **#salgode** (Salgo De) y busca que las personas puedan disponer sus medios de transporte a personas que no tienen como movilizarse de manera totalmente gratuita. Matías Andrade (@matiasAG) está liderando el proyecto. ¡Vamos que vamoos! El cambio parte desde cada uno. Necesitamos personas secas para React Native y que estén dispuestos a programar día y noche.

Telegram: t.me/salgode

Repo: <https://github.com/Varuscl/salgode>

Trello: <https://trello.com/b/GCTJ1iMU/salgode>



#Define la clase HaciaSJ

```
nom = input("Nombre? ")  
origen = input("Origen? ")
```

#Crea una variable de tipo HaciaSJ

```
print(v.nom,"necesita ir de",v.origen,"a",v.destino)
```

- Definir la clase “HaciaSJ”, con 3 atributos: nombre de la persona (nom), origen (ori), destino (des).
- Crear una variable “v” de tipo HaciaSJ
- Para crearla solo necesita 2 parámetros: el nombre y el origen (se entiende que el destino es “San Joaquin”)





I'LL WAIT
FOR YOU HERE



#Define la clase HaciaSJ

```
nom = input("Nombre? ")  
origen = input("Origen? ")
```

#Crea una variable de tipo HaciaSJ

```
print(v.nom,"necesita ir de",v.origen,"a",v.destino)
```

- Definir la clase “HaciaSJ”, con 3 atributos: nombre de la persona (nom), origen (ori), destino (des).
- Crear una variable “v” de tipo HaciaSJ
- Para crearla solo necesita 2 parámetros: el nombre y el origen (se entiende que el destino es “San Joaquin”)



```
#Define la clase HaciaSJ
class HaciaSJ:
    def __init__(self,n,o):
        self.nom = n
        self.ori = o
        self.des = "San Joaquin"

nom = input("Nombre? ")
origen = input("Origen? ")

#Crea una variable de tipo HaciaSJ
v = HaciaSJ(nom,origen)

print(v.nom,"necesita ir de",v.ori,"a",v.des)
```

- Definir la clase “HaciaSJ”, con 3 parámetros: nombre de la persona (nom), origen (ori), destino (des).
- Crear una variable “v” de tipo HaciaSJ
- Para crearla solo necesita 2 parámetros: el nombre y el origen (se entiende que el destino es “San Joaquin”)



- DCC Channel (Telegram – Estudiantes del DCC)
 - [https://t.me/DCC UC](https://t.me/DCC_UC)
- Alumnos DCC (Facebook)
 - <https://www.facebook.com/groups/197176750415821/>

GENIOMS



INTERESES

“Juegos”

“sobre videojuegos principalmente”

“RPG”



Situación
Real





```
>>>  
Geniom 1: Tesla  
Tesla tiene 10 de vida y 2 de ataque maximo  
Geniom 2: Darwin  
Darwin tiene 10 de vida y 5 de ataque maximo  
Tesla ataca con 1  
La vida de Darwin ahora es 9  
Darwin ataca con 3  
La vida de Tesla ahora es 7  
Tesla ataca con 1  
La vida de Darwin ahora es 8  
Darwin ataca con 1  
La vida de Tesla ahora es 6  
Tesla ataca con 0  
La vida de Darwin ahora es 8  
Darwin ataca con 2  
La vida de Tesla ahora es 4  
Tesla ataca con 2  
La vida de Darwin ahora es 6  
Darwin ataca con 4  
La vida de Tesla ahora es 0  
Combate acodado  
GANAN Darwin  
>>>
```



- Definir la clase “Geniom”, con 3 atributos: nombre de la del geniom (¿?), ataque máximo (¿?), y vida inicial (¿?). Los nombres de los atributos tienes que inferirlos del ejemplo.
- Crear una variable “¿?” y otra “¿?” de tipo Geniom. Los nombres de las variables tienes que inferirlos del ejemplo
- Para crearla solo necesita 1 parámetros: el nombre. La vida inicial siempre es 10, y el ataque máximo es un número aleatorio entre 1 y 5.



Situación
Real

Vida Inicial: 10
Ataque Max: 1-5

Clase

```
import random

n1 = input("Geniom 1: ")
g1 = Geniom(n1)
print(g1.nom,"tiene",g1.vida,"de vida y", g1.maxat,"de ataque maximo")

n2 = input("Geniom 2: ")
g2 = Geniom(n2)
print(g2.nom,"tiene",g2.vida,"de vida y", g2.maxat,"de ataque maximo")

turno = 1
while (g1.vida > 0) and (g2.vida > 0):

    if turno == 1:
        at = random.randint(0,g1.maxat)
        print(g1.nom, "ataca con", at)
        g2.vida = g2.vida - at
        print("La vida de", g2.nom, "ahora es", g2.vida)
        turno = 2

    elif turno == 2:
        at = random.randint(0,g2.maxat)
        print(g2.nom, "ataca con", at)
        g1.vida = g1.vida - at
        print("La vida de", g1.nom, "ahora es", g1.vida)
        turno = 1

print("Combate acabado")
if g1.vida > 0:
    print("GANÓ", g1.nom)
elif g2.vida > 0:
    print("GANÓ", g2.nom)
```





I'LL WAIT
FOR YOU HERE



Situación
Real

Vida Inicial: 10
Ataque Max: 1-5

Clase

```
import random

n1 = input("Geniom 1: ")
g1 = Geniom(n1)
print(g1.nom,"tiene",g1.vida,"de vida y", g1.maxat,"de ataque maximo")

n2 = input("Geniom 2: ")
g2 = Geniom(n2)
print(g2.nom,"tiene",g2.vida,"de vida y", g2.maxat,"de ataque maximo")

turno = 1
while (g1.vida > 0) and (g2.vida > 0):

    if turno == 1:
        at = random.randint(0,g1.maxat)
        print(g1.nom, "ataca con", at)
        g2.vida = g2.vida - at
        print("La vida de", g2.nom, "ahora es", g2.vida)
        turno = 2

    elif turno == 2:
        at = random.randint(0,g2.maxat)
        print(g2.nom, "ataca con", at)
        g1.vida = g1.vida - at
        print("La vida de", g1.nom, "ahora es", g1.vida)
        turno = 1

print("Combate acabado")
if g1.vida > 0:
    print("GANÓ", g1.nom)
elif g2.vida > 0:
    print("GANÓ", g2.nom)
```



Vida Inicial: 10
Ataque Max: 1-5

Clase

```
import random

class Geniom:
    def __init__(self, nom):
        self.nom = nom
        self.vida = 10
        self.maxat = 5

n1 = input("Geniom 1: ")
g1 = Geniom(n1)
print(g1.nom, "tiene", g1.vida, "de vida y", g1.maxat, "de ataque maximo")

n2 = input("Geniom 2: ")
g2 = Geniom(n2)
print(g2.nom, "tiene", g2.vida, "de vida y", g2.maxat, "de ataque maximo")

turno = 1
while (g1.vida > 0) and (g2.vida > 0):

    if turno == 1:
        at = random.randint(0,g1.maxat)
        print(g1.nom, "ataca con", at)
        g2.vida = g2.vida - at
        print("La vida de", g2.nom, "ahora es", g2.vida)
        turno = 2

    elif turno == 2:
        at = random.randint(0,g2.maxat)
        print(g2.nom, "ataca con", at)
        g1.vida = g1.vida - at
        print("La vida de", g1.nom, "ahora es", g1.vida)
        turno = 1

print("Combate acabado")
if g1.vida > 0:
    print("GANÓ", g1.nom)
elif g2.vida > 0:
    print("GANÓ", g2.nom)
```

```
import random

class Geniom:
    def __init__(self,n):
        self.nom = n
        self.vida = 10
        self.maxat = 5
```

Clase

Vida Inicial: 10
Ataque Max: 1-5

```
n1 = input("Geniom 1: ")
g1 = Geniom(n1)
print(g1.nom,"tiene",g1.vida,"de vida y", g1.maxat,"de ataque maximo")

n2 = input("Geniom 2: ")
g2 = Geniom(n2)
print(g2.nom,"tiene",g2.vida,"de vida y", g2.maxat,"de ataque maximo")

turno = 1
while (g1.vida > 0) and (g2.vida > 0):

    if turno == 1:
        at = random.randint(0,g1.maxat)
        print(g1.nom, "ataca con", at)
        g2.vida = g2.vida - at
        print("La vida de", g2.nom, "ahora es", g2.vida)
        turno = 2

    elif turno == 2:
        at = random.randint(0,g2.maxat)
        print(g2.nom, "ataca con", at)
        g1.vida = g1.vida - at
        print("La vida de", g1.nom, "ahora es", g1.vida)
        turno = 1

print("Combate acabado")
if g1.vida > 0:
    print("GANÓ", g1.nom)
elif g2.vida > 0:
    print("GANÓ", g2.nom)
```



Vida Inicial: 10

Ataque Max: 1-5

Clase

```
import random

class Geniom:
    def __init__(self,n):
        self.nom =
        self.maxat =
        self.vida = 

n1 = input("Geniom 1: ")
g1 = Geniom(n1)
print(g1.nom,"tiene",g1.vida,"de vida y", g1.maxat,"de ataque maximo")

n2 = input("Geniom 2: ")
g2 = Geniom(n2)
print(g2.nom,"tiene",g2.vida,"de vida y", g2.maxat,"de ataque maximo")

turno = 1
while (g1.vida > 0) and (g2.vida > 0):

    if turno == 1:
        at = random.randint(0,g1.maxat)
        print(g1.nom, "ataca con", at)
        g2.vida = g2.vida - at
        print("La vida de", g2.nom, "ahora es", g2.vida)
        turno = 2

    elif turno == 2:
        at = random.randint(0,g2.maxat)
        print(g2.nom, "ataca con", at)
        g1.vida = g1.vida - at
        print("La vida de", g1.nom, "ahora es", g1.vida)
        turno = 1

print("Combate acabado")
if g1.vida > 0:
    print("GANÓ", g1.nom)
elif g2.vida > 0:
    print("GANÓ", g2.nom)
```



Vida Inicial: 10
Ataque Max: 1-5

```
import random

class Geniom:
    def __init__(self,n):
        self.nom = n
        self.maxat = random.randint(1,5)
        self.vida = 10

n1 = input("Geniom 1: ")
g1 = Geniom(n1)
print(g1.nom,"tiene",g1.vida,"de vida y", g1.maxat,"de ataque maximo")

n2 = input("Geniom 2: ")
g2 = Geniom(n2)
print(g2.nom,"tiene",g2.vida,"de vida y", g2.maxat,"de ataque maximo")

turno = 1
while (g1.vida > 0) and (g2.vida > 0):

    if turno == 1:
        at = random.randint(0,g1.maxat)
        print(g1.nom, "ataca con", at)
        g2.vida = g2.vida - at
        print("La vida de", g2.nom, "ahora es", g2.vida)
        turno = 2

    elif turno == 2:
        at = random.randint(0,g2.maxat)
        print(g2.nom, "ataca con", at)
        g1.vida = g1.vida - at
        print("La vida de", g1.nom, "ahora es", g1.vida)
        turno = 1

print("Combate acabado")
if g1.vida > 0:
    print("GANÓ", g1.nom)
elif g2.vida > 0:
    print("GANÓ", g2.nom)
```



Bibliografía &
Investigación

Harold casts Spark!



change.com/questions/190946/text-based-python-rpg-game

Stack Exchange NEW

Search on Code Review...

CODE REVIEW

Home

Questions

Tags

Users

Unanswered

Text-based Python RPG game

This is a little text-based Python adventure game I found in the Stack Overflow code review section. It has a simple battle system, a shop, a save option and more. You can find it here: [here](#).

2

1

```
import sys
import os
import random
import pickle

weapons = {"Great Sword":40}

class Player:
    def __init__(self, name):
        self.name = name
        self.maxhealth = 100
        self.health = self.maxhealth
        self.base_attack = 10
        self.gold = 40
        self.pots = 0
        self.weap = ["Rusty Sword"]
        self.curweap = ["Rusty Sword"]
```

```
@property
def attack(self):
    attack = self.base_attack
    if self.curweap == "Rusty Sword":
        attack += 5

    if self.curweap == "Great Sword":
        attack += 15

    return attack
```

```
class Goblin:
    def __init__(self, name):
        self.name = name
        self.maxhealth = 50
        self.health = self.maxhealth
        self.attack = 5
```

justinmeister / The-Stolen-Crown-RPG

Code

Issues 0

Pull requests 0

GitHub is home
and review

Branch: master The-Stolen-Crown-RPG / data

justinmeister Almost finished tmx map of town, a few

1 contributor

14 lines (11 sloc) | 369 Bytes

```
1 __author__ = 'justinarmstrong'
2 import pygame as pg
3 from .. import constants as c
4
5
6 class Portal(pg.sprite.Sprite):
7     """Used to change level state"""
8     def __init__(self, x, y, name):
9         super(Portal, self).__init__()
10        self.image = pg.Surface((32, 32))
11        self.image.fill(c.BLACK)
12        self.rect = pg.Rect(x, y, 32, 32)
13        self.name = name
```

PRINT

```
n = 5  
print(n)  
  
s = "hola"  
print(s)  
  
p = ["adios", 8, "mundo"]  
print(p)
```

```
5  
hola  
['adios', 8, 'mundo']
```

```
c = Carta("corazones", 7)  
print(c)
```

```
<__main__.Carta object at 0x11260df60>
```

```

class Carta:
    def __init__(self,p,n):
        self.pinta = p
        self.num = n

    def __str__(self):
        t = "El " + str(self.num) + " de " + self.pinta
        return t
  
```



num
(int)

pinta
(str)

```
c1 = Carta("corazones",7)
print(c1)
```

```
c2 = Carta("rombos", 2)
print(c2)
```

El 7 de corazones
El 2 de rombos

- Implementar `__str__`
- NO imprime, **RETORNA** un str
- Tienes que construirlo
- Acceder a atributos usar **self.atributo**

SALGO DE (PRINT)



#Llego en

[Comenzar Viaje](#)

Hola! ¿Cómo están? Les cuento que estamos desarrollando un app móvil para aportar un granito de arena en todo este caos nacional. La app se llama **#salgode** (Salgo De) y busca que las personas puedan disponer sus medios de transporte a personas que no tienen como movilizarse de manera totalmente gratuita. Matías Andrade (@matiasAG) está liderando el proyecto. ¡Vamos que vamoos! El cambio parte desde cada uno. Necesitamos personas secas para React Native y que estén dispuestos a programar día y noche.

Telegram: t.me/salgode

Repo: <https://github.com/Varuscl/salgode>

Trello: <https://trello.com/b/GCTJ1iMU/salgode>



```
class HaciaSJ:  
    def __init__(self,n,o):  
        self.nom = n  
        self.ori = o  
        self.des = "San Joaquin"
```

```
nom = input("Nombre? ")  
origen = input("Origen? ")  
v = HaciaSJ(nom,origen)  
print(v)
```

Nombre? Alice
Origen? Puente Alto
Trayecto Puente Alto a San Joaquin para Alice





I'LL WAIT
FOR YOU HERE



```
class HaciaSJ:  
    def __init__(self,n,o):  
        self.nom = n  
        self.ori = o  
        self.des = "San Joaquin"
```

```
nom = input("Nombre? ")  
origen = input("Origen? ")  
v = HaciaSJ(nom,origen)  
print(v)
```

Nombre? Alice
Origen? Puente Alto
Trayecto Puente Alto a San Joaquin para Alice



```
class HaciaSJ:  
    def __init__(self,n,o):  
        self.nom = n  
        self.ori = o  
        self.des = "San Joquin"  
  
    def __str__(self):  
        t = "Trayecto "+self.ori+" a "+ self.des+" para "+self.nom  
        return t  
  
nom = input("Nombre? ")  
origen = input("Origen? ")  
v = HaciaSJ(nom,origen)  
print(v)
```

Nombre? Alice
Origen? Puente Alto
Trayecto Puente Alto a San Joquin para Alice

MÉTODOS



list

En listas puedes hacer
“append” para agregar
un elemento al final



str

En strings puedes hacer
“lower” para devolver
un str todo en minúsculas



Que fácil sería nuestra vida (y códigos) si a un Estudiante le pudiéramos pedir directamente algo como “obtener_nota_final” o otros

```
s1 = "Hola Mundo"
mn = s1.lower()
my = s1.upper()
print(s1,"en mayusculas es",my, "y en minuscula", mn)
```

Hola Mundo en mayusculas es HOLA MUNDO y en minuscula hola mundo

La clase se llama "Estudiante"

Y para crearse digo que necesita solo 4 parámetros: el nombre, y las notas de la I1, I2, y el examen



nom
(str)

i1
(float)

i2
(float)

ex
(float)

```
ali = Estudiante("Alice",4.0,4.0,4.0)
bb = Estudiante("Bob",6.0,4.0,5.0)
```

```
nali = ali.calcular_nota()
nbb = bb.calcular_nota()
```

```
print("Alice tiene un",nali,"y Bob tiene un",nbb)
```

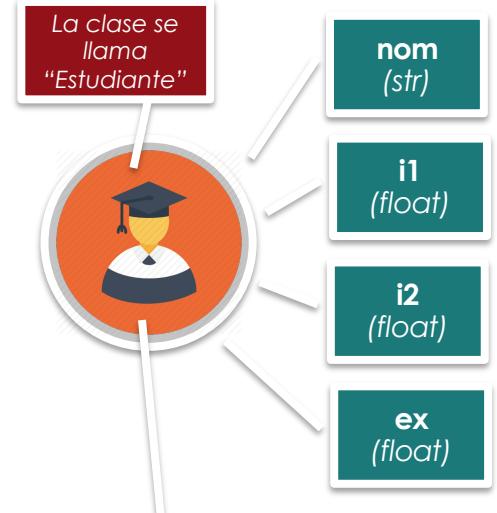
Alice tiene un 4.0 y Bob tiene un 5.0

```
class Estudiante:  
    def __init__(self,n,i1,i2,ex):  
        self.nom = n  
        self.i1 = i1  
        self.i2 = i2  
        self.ex = ex
```



```
ali = Estudiante("Alice",4.0,4.0,4.0)  
bb = Estudiante("Bob",6.0,4.0,5.0)  
  
nali = ali.calcular_nota()  
nbb = bb.calcular_nota()  
  
print("Alice tiene un",nali,"y Bob tiene un",nbb)
```

Alice tiene un 4.0 y Bob tiene un 5.0



Y para crearse digo que necesitas solo 4 parámetros: el nombre, y las notas de la I1, I2, y el examen

```
class Estudiante:  
    def __init__(self,n,i1,i2,ex):  
        self.nom = n  
        self.i1 = i1  
        self.i2 = i2  
        self.ex = ex  
  
    def calcular_nota(self):  
        n = (self.i1 * 0.25) + (self.i2 * 0.25) + (self.ex * 0.5)  
        return n  
  
ali = Estudiante("Alice",4.0,4.0,4.0)  
bb = Estudiante("Bob",6.0,4.0,5.0)  
  
nali = ali.calcular_nota()  
nbb = bb.calcular_nota()  
  
print("Alice tiene un",nali,"y Bob tiene un",nbb)
```

Alice tiene un 4.0 y Bob tiene un 5.0

El primer parámetro siempre es el “**self**”. Luego los otros parámetros (si necesita)

Es como una función

El “**self**” se usa para trabajar con el Estudiante al que le has preguntado la nota (Alice, Bob, o quien sea según el caso). Puedes usar **self.atributo** para acceder al valor de su atributo).

A los métodos se les llama “**x.metodo(...)**”

```
s1 = "Hola Mundo"  
mn = s1.lower()  
my = s1.upper()  
print(s1,"en mayusculas es",my, "y en minuscula", mn)
```

Hola Mundo en mayusculas es HOLA MUNDO y en minuscula hola mundo

¿Ahora entiendes por que s.lower() se llamaba así y no lower(s)?

String es un clase, y lower es un método suyo

¿Y puedo tener métodos que necesiten otros parámetros a parte del self?

sí

(Al final es como una función)

```
class Estudiante:  
    def __init__(self,nom, i1,i2,ex):  
        self.nom = nom  
        self.i1 = i1  
        self.i2 = i2  
        self.ex = ex
```



```
e1 = Estudiante("Alice",4.0,4.0,4.0)  
e2 = Estudiante("Bob",6.0,4.0,5.0)  
  
t1 = 3.5  
t2 = 4.5  
  
print(e1.nom,"supera", t1,"?", e1.supera(t1))  
print(e1.nom,"supera", t2,"?", e1.supera(t2))  
print(e2.nom,"supera", t1,"?", e2.supera(t1))  
print(e2.nom,"supera", t2,"?", e2.supera(t2))
```

```
Alice supera 3.5 ? True  
Alice supera 4.5 ? False  
Bob supera 3.5 ? True  
Bob supera 4.5 ? True
```

```
class Estudiante:  
    def __init__(self,nom, i1,i2,ex):  
        self.nom = nom  
        self.i1 = i1  
        self.i2 = i2  
        self.ex = ex  
  
    def supera(self,t):  
        n = (self.i1 * 0.25) + (self.i2 * 0.25) + (self.ex * 0.5)  
        if n >= t:  
            res = True  
        else:  
            res = False  
        return res  
  
e1 = Estudiante("Alice",4.0,4.0,4.0)  
e2 = Estudiante("Bob",6.0,4.0,5.0)  
  
t1 = 3.5  
t2 = 4.5  
  
print(e1.nom,"supera", t1,"?", e1.supera(t1))  
print(e1.nom,"supera", t2,"?", e1.supera(t2))  
print(e2.nom,"supera", t1,"?", e2.supera(t1))  
print(e2.nom,"supera", t2,"?", e2.supera(t2))
```

Alice supera 3.5 ? True
Alice supera 4.5 ? False
Bob supera 3.5 ? True
Bob supera 4.5 ? True

Me da pereza tener que hacer otra vez la formula para calcular la nota. ¿Puedo usar el método calcular_nota_des del método superá?

CLARO QUE SÍ
(Al final es como llamar a una función des de otra función)

```
class Estudiante:  
    def __init__(self,nom, i1,i2,ex):  
        self.nom = nom  
        self.i1 = i1  
        self.i2 = i2  
        self.ex = ex  
  
    def calcular_nota(self):  
        n = (self.i1 * 0.25) + (self.i2 * 0.25) + (self.ex * 0.5)  
        return n  
  
    def supera(self,t):  
        # Implementación de la función supera  
  
e1 = Estudiante("Alice",4.0,4.0,4.0)  
e2 = Estudiante("Bob",6.0,4.0,5.0)  
  
t1 = 3.5  
t2 = 4.5  
  
print(e1.nom,"supera", t1,"?", e1.supera(t1))  
print(e1.nom,"supera", t2,"?", e1.supera(t2))  
print(e2.nom,"supera", t1,"?", e2.supera(t1))  
print(e2.nom,"supera", t2,"?", e2.supera(t2))
```

>>>

```
Alice supera 3.5 ? True  
Alice supera 4.5 ? False  
Bob supera 3.5 ? True  
Bob supera 4.5 ? True
```

>>>

```
class Estudiante:  
    def __init__(self,nom, i1,i2,ex):  
        self.nom = nom  
        self.i1 = i1  
        self.i2 = i2  
        self.ex = ex  
  
    def calcular_nota(self):  
        n = (self.i1 * 0.25) + (self.i2 * 0.25) + (self.ex * 0.5)  
        return n  
  
    def supera(self,t):  
        n = self.calcular_nota()  
        if n >= t:  
            res = True  
        else:  
            res = False  
        return res  
  
e1 = Estudiante("Alice",4.0,4.0,4.0)  
e2 = Estudiante("Bob",6.0,4.0,5.0)  
  
t1 = 3.5  
t2 = 4.5  
  
print(e1.nom,"supera", t1,"?", e1.supera(t1))  
print(e1.nom,"supera", t2,"?", e1.supera(t2))  
print(e2.nom,"supera", t1,"?", e2.supera(t1))  
print(e2.nom,"supera", t2,"?", e2.supera(t2))
```

>>>

Alice supera 3.5 ? True

Alice supera 4.5 ? False

Bob supera 3.5 ? True

Bob supera 4.5 ? True

>>>

6 nimmt!



 SIMPLE GAME
Play as you wish

 REAL-TIME
Play in real-time

 AUTOMATIC
Automatic choice of opponents

 Play with friends

FAVORITE GAMES ❤

Display of

6 nimmt!

100





Play now!

Sushi Go!

268





Play now!

Saboteur

100





Play now!

Can't Stop

247





Play now!

Kingdomino

1





Carcassonne







Puerto Rico







Solo







General messages



Situación
Real

6 nimmt!



Play now!



<https://boardgamearena.com/>



```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)
```

Define una clase “ToroCarta” que solo tiene un atributo “num” (int) y solo necesita 1 parámetro para crearse (el numero de la carta)





I'LL WAIT
FOR YOU HERE



```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)
```

Define una clase “ToroCarta” que solo tiene un atributo “num” (int) y solo necesita 1 parámetro para crearse (el numero de la carta)



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)
```

Define una clase “ToroCarta” que solo tiene un atributo “num” (int) y solo necesita 1 parámetro para crearse (el numero de la carta)



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n
```



Carta 55 tiene 7 puntos
Carta 11 tiene 5 puntos
Carta 23 tiene 1 puntos

```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)
```

```
print("Carta",55,"tiene",c55.calcular_puntos(),"puntos")  
print("Carta",11,"tiene",c11.calcular_puntos(),"puntos")  
print("Carta",23,"tiene",c23.calcular_puntos(),"puntos")
```

Método calcular_puntos, sin parámetros, que te devuelve los puntos de la carta:

- 1 card with 7 cattle heads—number 55
- 8 cards with 5 cattle heads—multiples of 11 (except 55): 11, 22, 33, 44, 66, 77, 88, 99
- 10 cards with 3 cattle heads—multiples of ten: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
- 9 cards with 2 cattle heads—multiples of five that are not multiples of ten (except 55): 5, 15, 25, 35, 45, 65, 75, 85, 95
- 76 cards with 1 cattle head—the rest of the cards from 1 through 104





I'LL WAIT
FOR YOU HERE



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n
```



Carta 55 tiene 7 puntos
Carta 11 tiene 5 puntos
Carta 23 tiene 1 puntos

```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)
```

```
print("Carta",55,"tiene",c55.calcular_puntos(),"puntos")  
print("Carta",11,"tiene",c11.calcular_puntos(),"puntos")  
print("Carta",23,"tiene",c23.calcular_puntos(),"puntos")
```

Método calcular_puntos, sin parámetros, que te devuelve los puntos de la carta:

- 1 card with 7 cattle heads—number 55
- 8 cards with 5 cattle heads—multiples of 11 (except 55): 11, 22, 33, 44, 66, 77, 88, 99
- 10 cards with 3 cattle heads—multiples of ten: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
- 9 cards with 2 cattle heads—multiples of five that are not multiples of ten (except 55): 5, 15, 25, 35, 45, 65, 75, 85, 95
- 76 cards with 1 cattle head—the rest of the cards from 1 through 104



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p  
  
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)  
  
print("Carta",55,"tiene",c55.calcular_puntos(),"puntos")  
print("Carta",11,"tiene",c11.calcular_puntos(),"puntos")  
print("Carta",23,"tiene",c23.calcular_puntos(),"puntos")
```

Carta 55 tiene 7 puntos
Carta 11 tiene 5 puntos
Carta 23 tiene 1 puntos

Método calcular_puntos, sin parámetros, que te devuelve los puntos de la carta:

- 1 card with 7 cattle heads—number 55
- 8 cards with 5 cattle heads—multiples of 11 (except 55): 11, 22, 33, 44, 66, 77, 88, 99
- 10 cards with 3 cattle heads—multiples of ten: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
- 9 cards with 2 cattle heads—multiples of five that are not multiples of ten (except 55): 5, 15, 25, 35, 45, 65, 75, 85, 95
- 76 cards with 1 cattle head—the rest of the cards from 1 through 104



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p
```



```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)  
  
print(c55)  
print(c11)  
print(c23)
```

55(7)
11(5)
23(1)

Print “bonito” al imprimir una variable de tipo ToroCarta





I'LL WAIT
FOR YOU HERE



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p
```



```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)  
  
print(c55)  
print(c11)  
print(c23)
```

55(7)
11(5)
23(1)

Print “bonito” al imprimir una variable de tipo ToroCarta



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p  
  
    def __str__(self):  
        p = self.calcular_puntos()  
        t = str(self.num)+"("+str(p)+")"  
        return t
```

```
c55 = ToroCarta(55)  
c11 = ToroCarta(11)  
c23 = ToroCarta(23)  
  
print(c55)  
print(c11)  
print(c23)
```

55(7)
11(5)
23(1)

Print “bonito” al imprimir una variable de tipo ToroCarta



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p
```



```
def __str__(self):  
    p = self.calcular_puntos()  
    t = str(self.num)+"("+str(p)+")"  
    return t
```

```
c55 = ToroCarta(55)  
x = c55.al_lado(28)  
y = c55.al_lado(54)  
z = c55.al_lado(56)  
  
print("55 al lado de 28?",x)  
print("55 al lado de 54?",y)  
print("55 al lado de 56?",z)
```

```
55 al lado de 28? False  
55 al lado de 54? True  
55 al lado de 56? True
```

Método al_lado(num) que te dice si la carta está al lado del num (+1 o -1)





I'LL WAIT
FOR YOU HERE



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p
```



```
def __str__(self):  
    p = self.calcular_puntos()  
    t = str(self.num)+"("+str(p)+")"  
    return t
```

```
c55 = ToroCarta(55)  
x = c55.al_lado(28)  
y = c55.al_lado(54)  
z = c55.al_lado(56)  
  
print("55 al lado de 28?",x)  
print("55 al lado de 54?",y)  
print("55 al lado de 56?",z)
```

```
55 al lado de 28? False  
55 al lado de 54? True  
55 al lado de 56? True
```

Método al_lado(num) que te dice si la carta está al lado del num (+1 o -1)



```
class ToroCarta:  
    def __init__(self,n):  
        self.num = n  
  
    def calcular_puntos(self):  
        if self.num == 55:  
            p = 7  
        elif self.num % 11 == 0:  
            p = 5  
        elif self.num % 10 == 0:  
            p = 3  
        elif self.num % 5 == 0:  
            p = 2  
        else:  
            p = 1  
        return p  
  
    def al_lado(self,n):  
        if self.num == n+1 or self.num == n-1:  
            r = True  
        else:  
            r = False  
        return r  
  
    def __str__(self):  
        p = self.calcular_puntos()  
        t = str(self.num)+"("+str(p)+")"  
        return t  
  
c55 = ToroCarta(55)  
x = c55.al_lado(28)  
y = c55.al_lado(54)  
z = c55.al_lado(56)  
  
print("55 al lado de 28?",x)  
print("55 al lado de 54?",y)  
print("55 al lado de 56?",z)
```

```
55 al lado de 28? False  
55 al lado de 54? True  
55 al lado de 56? True
```

Método `al_lado(num)` que te dice si la carta está al lado del num (+1 o -1)



Si alguien pregunta decid que es “trabajo autónomo”

- <https://boardgamearena.com/lobby>
- https://en.wikipedia.org/wiki/6_Nimmt!

The screenshot shows the homepage of Board Game Arena. At the top, there are three main options: "SIMPLE GAME" (Play as you wish), "REAL-TIME" (Play in real-time), and "AUTOMATIC" (Automatic choice of opponents). Below these, a "Play with friends" button is visible. The main area is titled "FAVORITE GAMES" and displays eight board games with their respective boxes and "Play now!" buttons:

- 6 nimmt! (with 100 reviews)
- Sushi Go! (with 268 reviews)
- Saboteur (with 100 reviews)
- Can't Stop (with 247 reviews)
- Kingdomino (with 1 review)
- Carcassonne (with 1 review)
- Puerto Rico (with 1 review)
- SOLO (with 1 review)

At the bottom right, there is a link for "General messages".

PRIMEROS AUXILIOS UC



- Colaboras con los estudiantes de medicina UC en un puesto de primeros auxilios para todas las personas.
- Decides desarrollar una app para llevar registro de los casos atendidos.



Situación
Real

```
>>>
APP Ini
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 1
Nombre? alice
Descripcion de la Atención? corte
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 2
Nombre? alice
El numero de atenciones de alice es 1
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 2
Nombre? bob
El numero de atenciones de bob es 0
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 3
Nombre? alice
alice ha sido atendido: True
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 3
Nombre? bob
bob ha sido atendido: False
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 1
Nombre? alice
Descripcion de la Atención? quemadura
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 2
Nombre? charlie
El numero de atenciones de charlie es 0
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 1
Nombre? charlie
Descripcion de la Atención? corte
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 1
Nombre? charlie
Descripcion de la Atención? quemadura
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 4
Personas con multiples atenciones ['alice', 'charlie']
1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: 0
APP End
>>>
```



```
class Atenciones:
```

```
...
```

```
print("APP Ini")
ats = Atenciones()

continuar = True
while continuar:
    op = int(input("1-Nueva Atencion 2-Num Atenciones, 3-Atendido 4-Multiples 0-Salir: "))
    if op == 0:
        continuar = False
    elif op == 1:
        n = input("Nombre? ")
        a = input("Descripción de la Atención? ")
        ats.nueva_atencion(n,a)
    elif op == 2:
        n = input("Nombre? ")
        num = ats.num_atenciones(n)
        print("El número de atenciones de",n,"es",num)
    elif op == 3:
        n = input("Nombre? ")
        res = ats.atendido(n)
        print(n," ha sido atendido:", res)
    elif op == 4:
        mul = ats.multiples()
        print("Personas con multiples atenciones",mul)

print("APP End")
```



- Crea una clase Atenciones con 2 atributos: dos listas “noms” y “atens”, para guardar el nombre de la persona y la atención respectivamente (se entiende que un nombre en la posición “i” de noms se le atendió de lo que diga la posición i de atens. ¿qué valores iniciales tienes estas listas al empezar, antes de atender a nadie?)
- Crea un método “nueva_atención” que recibe un nombre y una atención y la agrega correctamente.
- Crea un método “num_atenciones” que recibe un nombre y te dice cuántas veces ha sido atendido.
- Crea un método “atendido”, que recibe un nombre y retorna un boolean indicando si esa persona ha sido atendida. Usa OBLIGATORIAMENTE el método “num_atenciones”.
- Crea un método que no recibe ningún parámetro y retorna una lista con los nombres de las personas que han sido atendidos múltiples veces.



```
class Atenciones:  
    def __init__(self):  
        self.noms = []  
        self.atens = []  
  
    def nueva_atencion(self,nom,aten):  
        self.noms.append(nom)  
        self.atens.append(aten)  
  
    def num_atenciones(self,nom):  
        res = 0  
        for n in self.noms:  
            if n == nom:  
                res += 1  
        return res  
  
    def atendido(self,nom):  
        num = self.num_atenciones(nom)  
        if num == 0:  
            res = False  
        else:  
            res = True  
        return res  
  
    def multiples(self):  
        #Nombres sin repetidos  
        norep = []  
        for n in self.noms:  
            if n not in norep:  
                norep.append(n)  
  
        multiples = []  
        for n in norep:  
            if self.num_atenciones(n) >= 2:  
                multiples.append(n)  
        return multiples
```



- Build your first app (Android Studio)
 - <https://developer.android.com/training/basics/firstapp>
- Flutter
 - <https://flutter.dev/>

CASINO



```
>>>
Que Fondo? pollo
Que Agregado? arroz
Quiere doblar agregado? 1-Si 2-No: 1
Quiere postre? 1-Si 2-No: 1
Que postre? yogurt
Su colacion es pollo con doble de arroz y yogurt
Precio: 2500
>>> ====== RESTART =====
>>>
Que Fondo? pollo
Que Agregado? arroz
Quiere doblar agregado? 1-Si 2-No: 2
Quiere postre? 1-Si 2-No: 2
Su colacion es pollo con arroz
Precio: 1600
>>> ====== RESTART =====
>>>
Que Fondo? cerdo
Que Agregado? papas
Quiere doblar agregado? 1-Si 2-No: 1
Quiere postre? 1-Si 2-No: 1
Que postre? yogurt
Su colacion es cerdo con doble de papas y yogurt
Precio: 2500
>>> ====== RESTART =====
>>>
Que Fondo? quinoa
Que Agregado? tomate
Quiere doblar agregado? 1-Si 2-No: 1
Quiere postre? 1-Si 2-No: 2
Su colacion es quinoa con doble de tomate
Precio: 1900
>>>
```

```
#NUEVA COLACION
c = Colacion()

c.fondo = input("Que Fondo? ")
c.agre = input("Que Agregado? ")

d = int(input("Quiere doblar agregado? 1-Si 2-No: "))
if d == 1:
    c.doble = True
else:
    c.doble = False

q = int(input("Quiere postre? 1-Si 2-No: "))
if q == 1:
    c.postre = input("Que postre? ")

print("Su colacion es", c)
print("Precio: ", c.precio())
```



Situación
Real



DHQato.com anano



\$ 35.990

Raspberry Pi 3 Model B+

MCI05221



```
>>>
Que Fondo? pollo
Que Agregado? arroz
Quiere doblar agregado? 1-Si 2-No: 1
Quiere postre? 1-Si 2-No: 1
Que postre? yogurt
Su colacion es pollo con doble de arroz y yogurt
Precio: 2500
>>> ====== RESTART =====
>>>
Que Fondo? pollo
Que Agregado? arroz
Quiere doblar agregado? 1-Si 2-No: 2
Quiere postre? 1-Si 2-No: 2
Su colacion es pollo con arroz
Precio: 1600
>>> ====== RESTART =====
>>>
Que Fondo? cerdo
Que Agregado? papas
Quiere doblar agregado? 1-Si 2-No: 1
Quiere postre? 1-Si 2-No: 1
Que postre? yogurt
Su colacion es cerdo con doble de papas y yogurt
Precio: 2500
>>> ====== RESTART =====
>>>
Que Fondo? quinoa
Que Agregado? tomate
Quiere doblar agregado? 1-Si 2-No: 1
Quiere postre? 1-Si 2-No: 2
Su colacion es quinoa con doble de tomate
Precio: 1900
>>>
```

```
#NUEVA COLACION
c = Colacion()

c.fondo = input("Que Fondo? ")
c.agre = input("Que Agregado? ")

d = int(input("Quiere doblar agregado? 1-Si 2-No: "))
if d == 1:
    c.doble = True
else:
    c.doble = False

q = int(input("Quiere postre? 1-Si 2-No: "))
if q == 1:
    c.postre = input("Que postre? ")

print("Su colacion es", c)
print("Precio: ", c.precio())
```

- Fondo y Agregado (1600)
- Doble agregado (+300)
- Postre (+600)



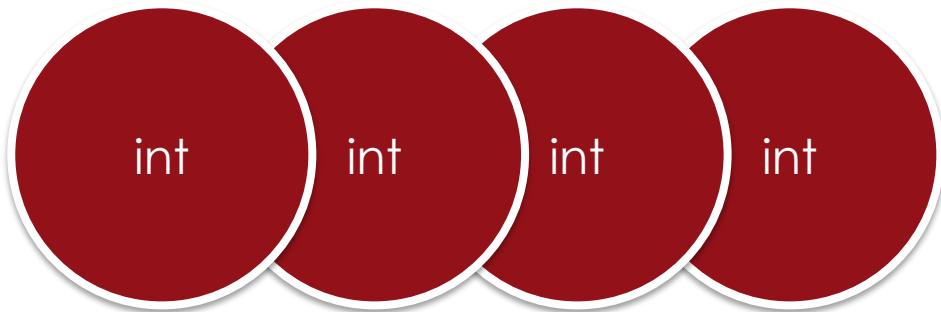
```
class Colacion:  
    def __init__(self):  
        self.fondo = ""  
        self.agre = ""  
        self.doble = ""  
        self.postre = ""  
  
    def precio(self):  
        p = 1600 #Fondo + agregado  
        if self.doble:  
            p += 300  
        if self.postre != "":  
            p += 600  
        return p  
  
    def __str__(self):  
        t = self.fondo + " con "  
        if self.doble:  
            t += "doble de "  
        t += self.agre  
        if self.postre != "":  
            t += " y " + self.postre  
        return t
```



- Raspberry Pi
 - <https://www.raspberrypi.org/>
 - https://www.mcielectronics.cl/en_US/shop/category/raspberry-pi-tarjetas-pi-416?src=raspberrypi
- Raspberry Pi Touch Display
 - <https://www.raspberrypi.org/products/raspberry-pi-touch-display/>
 - https://www.mcielectronics.cl/en_US/shop/product/raspberry-pi-7-touch-screen-display-11377?src=raspberrypi

LISTAS DE TIPOS PROPIOS

(RECORDATORIO)



Lista de int



Lista de Estudiante

```
#lista de ints
p = [10,20,10,40]

#Calcular promedio
suma = 0
for x in p:
    suma += x
promedio = suma / len(p)
print("Promedio ints: ", promedio)
```

```
#Clase Persona
class Persona:
    def __init__(self, n, e):
        self.nombre = n
        self.edad = e

#Lista de Persona
p1 = Persona("Alice",10)
p2 = Persona("Bob",20)
p3 = Persona("Charlie",10)
p4 = Persona("David",40)
q = [p1,p2,p3,p4]

#Calcular promedio edad
suma = 0
for x in q:
    suma += x.edad
promedio = suma / len(q)
print("Promedio edades: ", promedio)
```

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
#PARA TEST  
n1 = Nota("IIC1103", 6.5)  
n2 = Nota("IIC3757", 6.8)  
n3 = Nota("IIC2293", 3.5)  
ns = [n1,n2,n3]  
  
#Como cualquier lista  
for n in ns:  
    print(n.curso,"-",n.nota,"=> Pase:",n.pase())
```

```
>>>  
IIC1103 - 6.5 => Pase: True  
IIC3757 - 6.8 => Pase: True  
IIC2293 - 3.5 => Pase: False  
>>>
```

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
  
#Se puede append como cualquier lista de basicos  
ns = []  
n = int(input("Numero de ramos: "))  
for i in range(0,n):  
    c = input("Curso? ")  
    x = float(input("Nota? "))  
    n = Nota(c,x)  
    ns.append(n)  
  
print("Hiciste", len(ns), "cursos")  
  
r = 0  
for e in ns:  
    if e.pase():  
        r += 1  
print("Pasaste", r, "cursos")
```

```
>>>  
Numero de ramos: 3  
Curso? IIC1103  
Nota? 6.5  
Curso? IIC3757  
Nota? 6.8  
Curso? IIC2293  
Nota? 3.5  
Hiciste 3 cursos  
Pasaste 2 cursos  
>>>
```

INTERACCIONES ENTRE TIPOS PROPIOS

*¿Si en un tipo mio tengo un atributo de tipo String,
puedo usar los métodos de String?*

¿por qué no podrías?



```
class Persona:  
    def __init__(self,n,a):  
        self.nom = n #Un string  
        self.ape = a #Un string  
  
    def obtener_nombre_may(self):  
        #Como nom es str, puedo usar  
        #los métodos de str como siempre  
        res = self.nom.upper()  
        return res  
  
    def obtener_apellido_may(self):  
        res = self.ape.upper()  
        return res
```

```
alice = Persona("Alice", "Anderson")  
  
x = alice.obtener_nombre_may()  
print(x)  
  
y = alice.obtener_apellido_may()  
print(y)
```

ALICE
ANDERSON

```
class Persona:  
    def __init__(self,n,a):  
        self.nom = n #Un string  
        self.apel = a #Un string  
  
    def obtener_nombre_may(self):  
        #Como nom es str, puedo usar  
        #los métodos de str como siempre  
        res = self.nom.upper()  
        return res  
  
    def obtener_apellido_may(self):  
        res = self.apel.upper()  
        return res  
  
    def obtener_nom_ape_may(self):  
        #La clase persona tiene un metodo  
        #obtener_nombre_may. Lo uso  
        #como cualquier metodo de una clase  
        n = self.obtener_nombre_may()  
        a = self.obtener_apellido_may()  
        res = n + " " + a  
        return res  
  
alice = Persona("Alice", "Anderson")  
  
x = alice.obtener_nombre_may()  
print(x)  
  
y = alice.obtener_apellido_may()  
print(y)  
  
z = alice.obtener_nom_ape_may()  
print(z)
```

ALICE
ANDERSON
ALICE ANDERSON

¿Un tipo propio puede tener otro tipo propio como atributo?

¿y por qué no?

(¿Es lo mismo que tener un atributo de tipo *String* o de tipo *Lista*, no?)

```
class Jugador:  
    def __init__(self,n,e):  
        self.nom = n  
        self.equipo = e  
  
    def obtener_nom_equi(self):  
        res = self.nom+" - "+self.equipo  
        return res
```

```
alice = Jugador("Alice", "The Artful Dodgers")  
bob = Jugador("Bob", "The Dodgefathers")
```



```
class Jugador:  
    def __init__(self,n,e):  
        self.nom = n  
        self.equipo = e  
  
    def obtener_nom_equi(self):  
        res = self.nom+" - "+self.equipo  
        return res  
  
class Partida:  
    def __init__(self,j1,j2):  
        self.jugador1 = j1  
        self.jugador2 = j2  
  
alice = Jugador("Alice", "The Artful Dodgers")  
bob = Jugador("Bob", "The Dodgefathers")  
p = Partida(alice,bob)
```

```
class Jugador:  
    def __init__(self,n,e):  
        self.nom = n  
        self.equipo = e  
  
    def obtener_nom_equi(self):  
        res = self.nom+" - "+self.equipo  
        return res  
  
class Partida:  
    def __init__(self,j1,j2):  
        self.jugador1 = j1  
        self.jugador2 = j2  
  
    def obtener_equipo_j1(self):  
        #self.jugador1 es de tipo Jugador  
        #asi que podemos acceder a un atributo suyo  
        #como siempre  
        res = self.jugador1.equipo  
        return res  
  
alice = Jugador("Alice", "The Artful Dodgers")  
bob = Jugador("Bob", "The Dodgefathers")  
p = Partida(alice,bob)  
  
x = p.obtener_equipo_j1()  
print(x)
```

The Artful Dodgers

```
class Jugador:  
    def __init__(self,n,e):  
        self.nom = n  
        self.equipo = e  
  
    def obtener_nom_equi(self):  
        res = self.nom+" - "+self.equipo  
        return res  
  
class Partida:  
    def __init__(self,j1,j2):  
        self.jugador1 = j1  
        self.jugador2 = j2  
  
    def obtener_equipo_j1(self):  
        #self.jugador1 es de tipo Jugador  
        #asi que podemos acceder a un atributo suyo  
        #como siempre  
        res = self.jugador1.equipo  
        return res  
  
    def obtener_nom_equi_j1(self):  
        #self.jugador1 es de tipo Jugador  
        #por lo tanto tiene un metodo obtener_nom_equi  
        #y lo usamos como siempre  
        res = self.jugador1.obtener_nom_equi()  
        return res  
  
alice = Jugador("Alice", "The Artful Dodgers")  
bob = Jugador("Bob", "The Dodgefathers")  
p = Partida(alice,bob)  
  
x = p.obtener_equipo_j1()  
print(x)  
y = p.obtener_nom_equi_j1()  
print(y)
```

The Artful Dodgers
Alice-The Artful Dodgers

¿Puedes poner un ejemplo más?

Si, claro

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)
```

```
n1 = Nota("IIC1103", 6.5)  
n2 = Nota("IIC3757", 6.8)  
n3 = Nota("IIC2293", 3.5)
```

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
  
class Semestre:  
    def __init__(self):  
        self.ns = []  
  
        n1 = Nota("IIC1103", 6.5)  
        n2 = Nota("IIC3757", 6.8)  
        n3 = Nota("IIC2293", 3.5)  
  
        s = Semestre()
```

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
  
class Semestre:  
    def __init__(self):  
        self.ns = []  
  
    def agregar_nota(self, n):  
        self.ns.append(n)  
  
n1 = Nota("IIC1103", 6.5)  
n2 = Nota("IIC3757", 6.8)  
n3 = Nota("IIC2293", 3.5)  
  
s = Semestre()  
s.agregar_nota(n1)  
s.agregar_nota(n2)  
s.agregar_nota(n3)
```

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
  
class Semestre:  
    def __init__(self):  
        self.ns = []  
  
    def agregar_nota(self, n):  
        self.ns.append(n)  
  
    def cuantas_notas(self):  
        return len(self.ns)  
  
n1 = Nota("IIC1103", 6.5)  
n2 = Nota("IIC3757", 6.8)  
n3 = Nota("IIC2293", 3.5)  
  
s = Semestre()  
s.agregar_nota(n1)  
s.agregar_nota(n2)  
s.agregar_nota(n3)  
  
num = s.cuantas_notas()  
print(num, "Notas")
```

```
class Nota:  
    def __init__(self,curso,nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
  
class Semestre:  
    def __init__(self):  
        self.ns = []  
  
    def agregar_nota(self, n):  
        self.ns.append(n)  
  
    def cuantas_notas(self):  
        return len(self.ns)  
  
    def algun_siete(self):  
        r = False  
        for e in self.ns:  
            if e.nota == 7.0:  
                r = True  
        return r  
  
  
n1 = Nota("IIC1103", 6.5)  
n2 = Nota("IIC3757", 6.8)  
n3 = Nota("IIC2293", 3.5)  
  
s = Semestre()  
s.agregar_nota(n1)  
s.agregar_nota(n2)  
s.agregar_nota(n3)  
  
num = s.cuantas_notas()  
print(num, "Notas")  
  
siete = s.algun_siete()  
print("Algun siete: ",siete)
```

```

class Nota:
    def __init__(self,curso,nota):
        self.curso = curso
        self.nota = nota

    def pase(self):
        return (self.nota >= 4.0)

class Semestre:
    def __init__(self):
        self.ns = []

    def agregar_nota(self, n):
        self.ns.append(n)

    def cuantas_notas(self):
        return len(self.ns)

    def algun_siete(self):
        r = False
        for e in self.ns:
            if e.nota == 7.0:
                r = True
        return r

    def cuantas_pase(self):
        r = 0
        for e in self.ns:
            if e.pase():
                r += 1
        return r
  
```

```

n1 = Nota("IIC1103", 6.5)
n2 = Nota("IIC3757", 6.8)
n3 = Nota("IIC2293", 3.5)

s = Semestre()
s.agregar_nota(n1)
s.agregar_nota(n2)
s.agregar_nota(n3)

num = s.cuantas_notas()
print(num, "Notas")

siete = s.algun_siete()
print("Algun siete: ",siete)

npase = s.cuantas_pase()
print("Pase", npase)
  
```

FERIA DE INVESTIGACIÓN DCC



Pregunta 3

¡Bienvenido a la Feria de Investigación del DCC! El siguiente código usa programación orientada a objetos, incluyendo las clases de objetos **Stand** y **Feria** para gestionar la asignación de stands durante la feria. En concreto, primero se crea la feria con `ns` stands vacíos. A continuación, se escoge entre tres opciones:

1. **Inscribir un stand.** Si quedan stands vacíos, se solicita el título del stand y el número de participantes, de lo contrario, se imprime que no quedan stands vacíos.
2. **Información.** Dado un número de stand, en caso de estar vacío, se indica que está vacío. En caso contrario, se indica el título del stand y el número de participantes.
3. **Cerrar.** Cierra la feria.



```
ns = int(input("Número de stands en la Feria: "))
f = Feria(ns)

op = int(input("1-Inscribir 2-Info 3-Cerrar: "))
while op != 3:

    if op == 1:
        if f.quedan_stands_vacios():
            t = input("Título del stand: ")
            p = int(input("Número de participantes del stand: "))
            f.inscribir(t,p)
        else:
            print("No quedan stands vacíos")

    elif op == 2:
        n = int(input("Número de stand: "))
        s = f.obtener_stand(n) #s de tipo Stand
        if s.vacio():
            print("Stand vacío")
        else:
            print("Título =>", s.titulo())
            print("Participantes =>", s.participantes())

    op = int(input("1-Inscribir 2-Info 3-Cerrar: "))

print("Feria Cerrada")
print("Quedaron", f.numero_stands_vacios(), "stands por llenar")
```



Se te pide que implementes las clases **Stand** y **Feria** con los siguientes métodos con el fin de **que sea compatible con el código principal de ejemplo**:

1) Stand:

- **(6 puntos)** `__init__`: No recibe parámetros. Inicializa un stand vacío con sus atributos que corresponda.
- **(3 puntos)** `titulo`: No recibe parámetros. Retorna un *string* con el título del stand.
- **(3 puntos)** `participantes`: No recibe parámetros. Retorna un *int* con la cantidad de participantes del stand.
- **(3 puntos)** `vacio`: No recibe parámetros. Retorna `True` si el stand está vacío; y `False`, en caso contrario.





I'LL WAIT
FOR YOU HERE



Se te pide que implementes las clases **Stand** y **Feria** con los siguientes métodos con el fin de **que sea compatible con el código principal de ejemplo**:

1) Stand:

- **(6 puntos)** `__init__`: No recibe parámetros. Inicializa un stand vacío con sus atributos que corresponda.
- **(3 puntos)** `titulo`: No recibe parámetros. Retorna un *string* con el título del stand.
- **(3 puntos)** `participantes`: No recibe parámetros. Retorna un *int* con la cantidad de participantes del stand.
- **(3 puntos)** `vacio`: No recibe parámetros. Retorna `True` si el stand está vacío; y `False`, en caso contrario.



```
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0
```

Se te pide que implementes las clases `Stand` y `Feria` con los siguientes métodos con el fin de **que sea compatible con el código principal de ejemplo**:

1) `Stand`:

- **(6 puntos)** `__init__`: No recibe parámetros. Inicializa un stand vacío con sus atributos que corresponda.
- **(3 puntos)** `titulo`: No recibe parámetros. Retorna un *string* con el título del stand.
- **(3 puntos)** `participantes`: No recibe parámetros. Retorna un *int* con la cantidad de participantes del stand.
- **(3 puntos)** `vacio`: No recibe parámetros. Retorna `True` si el stand está vacío; y `False`, en caso contrario.



2) Feria:

- **(10 puntos) `__init__`:** Recibe como parámetro un *int* con el número de stands vacíos (de la clase `Stand`) que tiene la feria al crearse. Se inicializa la feria con los stands vacíos creados. Los stands están ordenados, y se pueden identificar con un número que va de 0 al número de stands -1.
- **(15 puntos) `inscribir`:** Recibe un *string* con el título del stand y un *int* con la cantidad de participantes de ese stand (debe ser mayor a 0 y puedes asumir que el input para este atributo siempre cumplirá dicha restricción). Se inscribe el primer stand vacío con dichos parámetros. Puedes suponer que al inscribirse, ya se verificó antes si hay stands vacíos. No retorna nada.
- **(5 puntos) `obtener_stand`:** Recibe un *int* con el índice del stand (va de 0 al número de stands -1) Retorna un objeto de tipo `Stand` con el stand solicitado.
- **(10 puntos) `numero_stands_vacios`:** No recibe parámetros. Retorna un *int* con el número de stands vacíos de la feria.
- **(5 puntos) `quedan_stands_vacios`:** No recibe parámetros. Retorna `True` si quedan stands vacíos o `False` en caso contrario. **Este método debe usar el método `numero_stands_vacios` anterior.**



```
class Nota:  
    def __init__(self, curso, nota):  
        self.curso = curso  
        self.nota = nota  
  
    def pase(self):  
        return (self.nota >= 4.0)  
  
class Semestre:  
    def __init__(self):  
        self.ns = []  
  
    def agregar_nota(self, n):  
        self.ns.append(n)
```

```
n1 = Nota("IIC1103", 6.5)  
n2 = Nota("IIC3757", 6.8)  
n3 = Nota("IIC2293", 3.5)  
  
s = Semestre()  
s.agregar_nota(n1)  
s.agregar_nota(n2)  
s.agregar_nota(n3)
```





I'LL WAIT
FOR YOU HERE



```
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0
```

2) Feria:

- **(10 puntos)** `__init__`: Recibe como parámetro un *int* con el número de stands vacíos (de la clase `Stand`) que tiene la feria al crearse. Se inicializa la feria con los stands vacíos creados. Los stands están ordenados, y se pueden identificar con un número que va de 0 al número de stands -1.
- **(15 puntos)** `inscribir`: Recibe un *string* con el título del stand y un *int* con la cantidad de participantes de ese stand (debe ser mayor a 0 y puedes asumir que el input para este atributo siempre cumplirá dicha restricción). Se inscribe el primer stand vacío con dichos parámetros. Puedes suponer que al inscribirse, ya se verificó antes si hay stands vacíos. No retorna nada.
- **(5 puntos)** `obtener_stand`: Recibe un *int* con el índice del stand (va de 0 al número de stands -1) Retorna un objeto de tipo `Stand` con el stand solicitado.
- **(10 puntos)** `numero_stands_vacios`: No recibe parámetros. Retorna un *int* con el número de stands vacíos de la feria.
- **(5 puntos)** `quedan_stands_vacíos`: No recibe parámetros. Retorna `True` si quedan stands vacíos o `False` en caso contrario. **Este método debe usar el método numero_stands_vacios anterior.**



```
class Feria:  
    def __init__(self, ns):  
        self.stands = []  
        for i in range(0,ns):  
            s = Stand()  
            self.stands.append(s)  
  
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0
```

2) Feria:

- **(10 puntos)** `__init__`: Recibe como parámetro un *int* con el número de stands vacíos (de la clase `Stand`) que tiene la feria al crearse. Se inicializa la feria con los stands vacíos creados. Los stands están ordenados, y se pueden identificar con un número que va de 0 al número de stands -1.



```
class Feria:  
    def __init__(self, ns):  
        self.stands = []  
        for i in range(0,ns):  
            s = Stand()  
            self.stands.append(s)  
  
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0  
  
def inscribir(self,t,p):  
    i = 0  
    inscrito = False  
    while not inscrito:  
        s = self.stands[i]  
        if s.vacio():  
            s.titulo = t  
            s.par = p  
            inscrito = True  
        i += 1
```

2) Feria:

- **(10 puntos)** `__init__`: Recibe como parámetro un *int* con el número de stands vacíos (de la clase `Stand`) que tiene la feria al crearse. Se inicializa la feria con los stands vacíos creados. Los stands están ordenados, y se pueden identificar con un número que va de 0 al número de stands -1.
- **(15 puntos)** `inscribir`: Recibe un *string* con el título del stand y un *int* con la cantidad de participantes de ese stand (debe ser mayor a 0 y puedes asumir que el input para este atributo siempre cumplirá dicha restricción). Se inscribe el primer stand vacío con dichos parámetros. Puedes suponer que al inscribirse, ya se verificó antes si hay stands vacíos. No retorna nada.



```
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0
```

```
class Feria:  
    def __init__(self, ns):  
        self.stands = []  
        for i in range(0,ns):  
            s = Stand()  
            self.stands.append(s)  
  
    def inscribir(self,t,p):  
        i = 0  
        inscrito = False  
        while not inscrito:  
            s = self.stands[i]  
            if s.vacio():  
                s.titulo = t  
                s.par = p  
                inscrito = True  
            i += 1  
  
    def obtener_stand(self, n):  
        return self.stands[n]
```

2) Feria:

- **(10 puntos)** `__init__`: Recibe como parámetro un *int* con el número de stands vacíos (de la clase `Stand`) que tiene la feria al crearse. Se inicializa la feria con los stands vacíos creados. Los stands están ordenados, y se pueden identificar con un número que va de 0 al número de stands -1.
- **(15 puntos)** `inscribir`: Recibe un *string* con el título del stand y un *int* con la cantidad de participantes de ese stand (debe ser mayor a 0 y puedes asumir que el input para este atributo siempre cumplirá dicha restricción). Se inscribe el primer stand vacío con dichos parámetros. Puedes suponer que al inscribirse, ya se verificó antes si hay stands vacíos. No retorna nada.
- **(5 puntos)** `obtener_stand`: Recibe un *int* con el índice del stand (va de 0 al número de stands -1) Retorna un objeto de tipo `Stand` con el stand solicitado.



```
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0
```

```
class Feria:  
    def __init__(self, ns):  
        self.stands = []  
        for i in range(0,ns):  
            s = Stand()  
            self.stands.append(s)  
  
    def inscribir(self,t,p):  
        i = 0  
        inscrito = False  
        while not inscrito:  
            s = self.stands[i]  
            if s.vacio():  
                s.titulo = t  
                s.par = p  
                inscrito = True  
            i += 1  
  
    def obtener_stand(self, n):  
        return self.stands[n]  
  
    def numero_stands_vacios(self):  
        vacios = 0  
        for s in self.stands:  
            if s.vacio():  
                vacios += 1  
        return vacios
```

2) Feria:

- (10 puntos) `numero_stands_vacios`: No recibe parámetros. Retorna un `int` con el número de stands vacíos de la feria.



```
class Stand:  
    def __init__(self):  
        self.titulo = ""  
        self.par = 0  
  
    def titulo(self):  
        return self.titulo  
  
    def participantes(self):  
        return self.par  
  
    def vacio(self):  
        return self.par == 0
```

```
class Feria:  
    def __init__(self, ns):  
        self.stands = []  
        for i in range(0,ns):  
            s = Stand()  
            self.stands.append(s)  
  
    def inscribir(self,t,p):  
        i = 0  
        inscrito = False  
        while not inscrito:  
            s = self.stands[i]  
            if s.vacio():  
                s.titulo = t  
                s.par = p  
                inscrito = True  
            i += 1  
  
    def obtener_stand(self, n):  
        return self.stands[n]  
  
    def numero_stands_vacios(self):  
        vacios = 0  
        for s in self.stands:  
            if s.vacio():  
                vacios += 1  
        return vacios  
  
    def quedan_stands_vacios(self):  
        return self.numero_stands_vacios() > 0
```

2) Feria:

- **(10 puntos)** `numero_stands_vacios`: No recibe parámetros. Retorna un *int* con el número de stands vacíos de la feria.
- **(5 puntos)** `quedan_stands_vacios`: No recibe parámetros. Retorna `True` si quedan stands vacíos o `False` en caso contrario. Este método debe usar el método `numero_stands_vacios` anterior.

DCCHURRASCO



Pregunta 3

Bienvenido al restaurante *DCChurrasco!* El siguiente código usa la programación orientada a objetos y las clases de objetos **Mesa** y **Restaurante** para gestionar el restaurante. En concreto 1) Cuando llega un grupo de clientes, el sistema los sienta en una mesa libre, o les dice que no hay mesas disponibles, 2) Se puede pedir comida para una mesa, indicando el plato y su precio, 3) Al pedir la cuenta el sistema indica el número de platos consumidos por la mesa, el precio total y el precio por persona, en partes iguales, quedando la mesa libre en ese momento, 4) Da la opción de salir del programa. Considera que todas las mesas tienen 4 sillas y que siempre llegarán grupos de al menos 1 persona y a lo más 4 personas.



```
r = Restaurante('DCChurrasco', 2) # restaurante llamado 'DCChurrasco' con 2 mesas vacías

continuar = True
while continuar:
    opcion = int(input('1-Llegan clientes 2-Pedir comida 3-La cuenta 9-Salir: '))

    if opcion == 1:
        cantidad_clientes = int(input('Número de clientes: '))
        numero_mesa = r.llegan_clientes(cantidad_clientes)
        if numero_mesa == -1:
            print('No hay mesas disponibles :(')
        else:
            print('Siéntense en la mesa', numero_mesa)

    elif opcion == 2:
        numero_mesa = int(input('Número de mesa: '))
        plato = input('Plato: ')
        precio = int(input('Precio: '))
        r.mesas[numero_mesa].agregar_plato(plato, precio)

    elif opcion == 3:
        numero_mesa = int(input('Número de mesa: '))
        mesa = r.mesas[numero_mesa]
        print('Pidieron', len(mesa.obtener_platos()), 'platos')
        print('La cuenta son', mesa.obtener_cuenta())
        print('Eso hace', mesa.obtener_cuenta_por_persona(), 'por persona')
        mesa.vaciar_mesa()

    elif opcion == 9:
        continuar = False
        print('Saliendo')
```



Se te pide que, considerando el código anterior, implementes las clases **Mesa** y **Restaurante** con los siguientes métodos:

1) **Mesa:**

- **(7 puntos) __init__:** Inicializa una mesa vacía. No recibe ningún parámetro.
- **(7 puntos) ocupar_mesa:** Establece la mesa como ocupada. Recibe un parámetro: un **int** correspondiente al número de personas que se sientan en la mesa. No retorna nada.
- **(4 puntos) esta_vacia:** Retorna **False** si la mesa se encuentra ocupada, **True** en el caso contrario. No recibe ningún parámetro.
- **(4 puntos) agregar_plato:** Registra un nuevo plato a los platos pedidos por la mesa. Recibe como parámetros el nombre del plato como un **string** y el precio del plato como un **int**.
- **(4 puntos) obtener_cuenta:** Retorna un **int** con la suma de precios de todos los platos pedidos en una mesa. No recibe ningún parámetro.
- **(4 puntos) obtener_cuenta_por_persona:** Retorna un **float** con lo que debe pagar cada cliente. Se asume que dividen la cuenta en partes iguales. **Es necesario que se use el método obtener_cuenta para hacer este cálculo.** No recibe ningún parámetro.
- **(4 puntos) obtener_platos:** Retorna una **lista** de **string** con los platos pedidos por la mesa. No recibe ningún parámetro.
- **(7 puntos) vaciar_mesa:** Establece la mesa como vacía, cambiando los atributos necesarios para que pueda ser usada nuevamente por otro grupo de clientes.





I'LL WAIT
FOR YOU HERE



Se te pide que, considerando el código anterior, implementes las clases **Mesa** y **Restaurante** con los siguientes métodos:

1) **Mesa:**

- **(7 puntos)** `__init__`: Inicializa una mesa vacía. No recibe ningún parámetro.
- **(7 puntos)** `ocupar_mesa`: Establece la mesa como ocupada. Recibe un parámetro: un `int` correspondiente al número de personas que se sientan en la mesa. No retorna nada.
- **(4 puntos)** `esta_vacia`: Retorna `False` si la mesa se encuentra ocupada, `True` en el caso contrario. No recibe ningún parámetro.
- **(4 puntos)** `agregar_plato`: Registra un nuevo plato a los platos pedidos por la mesa. Recibe como parámetros el nombre del plato como un `string` y el precio del plato como un `int`.
- **(4 puntos)** `obtener_cuenta`: Retorna un `int` con la suma de precios de todos los platos pedidos en una mesa. No recibe ningún parámetro.
- **(4 puntos)** `obtener_cuenta_por_persona`: Retorna un `float` con lo que debe pagar cada cliente. Se asume que dividen la cuenta en partes iguales. **Es necesario que se use el método `obtener_cuenta` para hacer este cálculo.** No recibe ningún parámetro.
- **(4 puntos)** `obtener_platos`: Retorna una `lista` de `string` con los platos pedidos por la mesa. No recibe ningún parámetro.
- **(7 puntos)** `vaciar_mesa`: Establece la mesa como vacía, cambiando los atributos necesarios para que pueda ser usada nuevamente por otro grupo de clientes.



```
class Mesa:  
    def __init__(self):  
        self.ocupantes = 0  
        self.cuenta = 0  
        self.platos = []  
  
    def ocupar_mesa(self, num_ocupantes):  
        self.ocupantes = num_ocupantes  
  
    def esta_vacia(self):  
        return self.ocupantes == 0
```

Se te pide que, considerando el código anterior, implementes las clases **Mesa** y **Restaurante** con los siguientes métodos:

1) **Mesa:**

- **(7 puntos)** `__init__`: Inicializa una mesa vacía. No recibe ningún parámetro.
- **(7 puntos)** `ocupar_mesa`: Establece la mesa como ocupada. Recibe un parámetro: un `int` correspondiente al número de personas que se sientan en la mesa. No retorna nada.
- **(4 puntos)** `esta_vacia`: Retorna `False` si la mesa se encuentra ocupada, `True` en el caso contrario. No recibe ningún parámetro.



```
class Mesa:  
    def __init__(self):  
        self.ocupantes = 0  
        self.cuenta = 0  
        self.platos = []  
  
    def ocupar_mesa(self, num_ocupantes):  
        self.ocupantes = num_ocupantes  
  
    def esta_vacia(self):  
        return self.ocupantes == 0  
  
    def agregar_plato(self, plato, precio):  
        self.cuenta += precio  
        self.platos.append(plato)
```

1) Mesa:

- **(7 puntos) `__init__`:** Inicializa una mesa vacía. No recibe ningún parámetro.
- **(7 puntos) `ocupar_mesa`:** Establece la mesa como ocupada. Recibe un parámetro: un `int` correspondiente al número de personas que se sientan en la mesa. No retorna nada.
- **(4 puntos) `esta_vacia`:** Retorna `False` si la mesa se encuentra ocupada, `True` en el caso contrario. No recibe ningún parámetro.
- **(4 puntos) `agregar_plato`:** Registra un nuevo plato a los platos pedidos por la mesa. Recibe como parámetros el nombre del plato como un `string` y el precio del plato como un `int`.



```
class Mesa:  
    def __init__(self):  
        self.ocupantes = 0  
        self.cuenta = 0  
        self.platos = []  
  
    def ocupar_mesa(self, num_ocupantes):  
        self.ocupantes = num_ocupantes  
  
    def esta_vacia(self):  
        return self.ocupantes == 0  
  
    def agregar_plato(self, plato, precio):  
        self.cuenta += precio  
        self.platos.append(plato)  
  
    def obtener_cuenta(self):  
        return self.cuenta  
  
    '''Obligado usar obtener_cuenta()'''  
    def obtener_cuenta_por_persona(self):  
        return self.obtener_cuenta() / self.ocupantes
```

- **(4 puntos) obtener_cuenta:** Retorna un `int` con la suma de precios de todos los platos pedidos en una mesa. No recibe ningún parámetro.
- **(4 puntos) obtener_cuenta_por_persona:** Retorna un `float` con lo que debe pagar cada cliente. Se asume que dividen la cuenta en partes iguales. **Es necesario que se use el método `obtener_cuenta` para hacer este cálculo.** No recibe ningún parámetro.



```
class Mesa:  
    def __init__(self):  
        self.ocupantes = 0  
        self.cuenta = 0  
        self.platos = []  
  
    def ocupar_mesa(self, num_ocupantes):  
        self.ocupantes = num_ocupantes  
  
    def esta_vacia(self):  
        return self.ocupantes == 0  
  
    def agregar_plato(self, plato, precio):  
        self.cuenta += precio  
        self.platos.append(plato)  
  
    def obtener_cuenta(self):  
        return self.cuenta  
  
    '''Obligado usar obtener_cuenta()'''  
    def obtener_cuenta_por_persona(self):  
        return self.obtener_cuenta() / self.ocupantes  
  
    def obtener_platos(self):  
        return self.platos  
  
    def vaciar_mesa(self):  
        self.ocupantes = 0  
        self.cuenta = 0  
        self.platos = []
```

- **(4 puntos) obtener_platos:** Retorna una lista de string con los platos pedidos por la mesa. No recibe ningún parámetro.
- **(7 puntos) vaciar_mesa:** Establece la mesa como vacía, cambiando los atributos necesarios para que pueda ser usada nuevamente por otro grupo de clientes.



2) Restaurante:

- **(12 puntos)** `__init__`: Inicializa un restaurante. Recibe como parámetro el nombre del restaurante como un `string` y un `int` con el número de mesas vacías (de la clase `Mesa`) que tiene el restaurante al crearse.
- **(7 puntos)** `llegan_clientes`: Recibe un `int` con la cantidad de clientes que llegaron. En caso de que exista una mesa vacía, la mesa queda ocupada con los clientes, y el método retorna un `int` con el número que identifica la mesa. Considera que las mesas de un restaurante se identifican con un número del 0 al número de mesas - 1. En caso de que haya más de una mesa vacía, el sistema puede elegir cualquiera. En caso de que no haya ninguna mesa vacía, el método simplemente retorna -1.





I'LL WAIT
FOR YOU HERE



2) Restaurante:

- **(12 puntos)** `__init__`: Inicializa un restaurante. Recibe como parámetro el nombre del restaurante como un `string` y un `int` con el número de mesas vacías (de la clase `Mesa`) que tiene el restaurante al crearse.
- **(7 puntos)** `llegan_clientes`: Recibe un `int` con la cantidad de clientes que llegaron. En caso de que exista una mesa vacía, la mesa queda ocupada con los clientes, y el método retorna un `int` con el número que identifica la mesa. Considera que las mesas de un restaurante se identifican con un número del 0 al número de mesas - 1. En caso de que haya más de una mesa vacía, el sistema puede elegir cualquiera. En caso de que no haya ninguna mesa vacía, el método simplemente retorna -1.



```
class Restaurante:

    def __init__(self, nombre, num_mesas):
        self.nombre = nombre
        self.mesas = []
        for i in range(0,num_mesas):
            self.mesas.append(Mesa())

    def llegan_clientes(self, num_comensales):
        ix = -1
        for i in range(0,len(self.mesas)):
            mesa = self.mesas[i]
            if ix == -1 and mesa.esta_vacia():
                ix = i
                mesa.ocupar_mesa(num_comensales)
        return ix
```

2) Restaurante:

- **(12 puntos)** `__init__`: Inicializa un restaurante. Recibe como parámetro el nombre del restaurante como un `string` y un `int` con el número de mesas vacías (de la clase `Mesa`) que tiene el restaurante al crearse.
- **(7 puntos)** `llegan_clientes`: Recibe un `int` con la cantidad de clientes que llegaron. En caso de que exista una mesa vacía, la mesa queda ocupada con los clientes, y el método retorna un `int` con el número que identifica la mesa. Considera que las mesas de un restaurante se identifican con un número del 0 al número de mesas - 1. En caso de que haya más de una mesa vacía, el sistema puede elegir cualquiera. En caso de que no haya ninguna mesa vacía, el método simplemente retorna -1.