ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Εισαγωγή στους Υπολογιστές

Εργαστήριο 8

Καθηγητές: Αβούρης Νικόλαος, Παλιουράς Βασίλης, Κουκιάς Μιχαήλ, Σγάρμπας Κυριάκος

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Εργαστήριο 8: Πέμπτη Άσκηση Προγραμματισμού Python

8.1 Γενικά

Η άσκηση αυτή επιχειρεί μια εισαγωγή στον αντικειμενοστραφή προγραμματισμό. Αφορά **αντικείμενα, κλάσεις, μεθόδους** και **κληρονομικότητα.**

8.2 Αντικείμενα και κλάσεις

Θυμάστε σε προηγούμενη άσκηση που χρειάστηκε να κάνουμε υπολογισμούς με μιγαδικούς αριθμούς, πόσο βολικό ήταν που η Python διέθετε αυτή τη δυνατότητα; Βέβαια, ακόμα κι αν δεν την διέθετε, πάλι θα μπορούσαμε να κάνουμε τις ίδιες πράξεις αναπαριστώντας τους μιγαδικούς με κάποιον άλλο τρόπο (πχ. ως λίστες ή ως λεξικά δύο στοιχείων) απλώς το πρόγραμμα θα ήταν λίγο πιο περίπλοκο. Συχνά τα προγράμματά μας απλοποιούνται πολύ όταν ορίζουμε κατάλληλες σύνθετες δομές (συνήθως αρκετά πιο σύνθετες από έναν μιγαδικό αριθμό) οι οποίες λέγονται **αντικείμενα** (**objects**). Γύρω από τη χρήση των αντικειμένων υπάρχει μια ολόκληρη σχεδιαστική φιλοσοφία που λέγεται **αντικειμενοστραφής προγραμματισμός** (**object oriented programming**).

Ας δούμε ένα παράδειγμα. Έστω ότι θέλουμε να προγραμματίσουμε ένα παιχνίδι με τραπουλόχαρτα[1]. Για καθένα θα χρειαστεί να καταγράψουμε την αξία του (πχ. 5, 8, Q, A, κλπ), το σύμβολό του (μπαστούνι, κούπα, σπαθί, καρώ) και ίσως ακόμα κάποια επιπλέον στοιχεία που χρειάζονται για κάποια παιχνίδια, όπως το χρώμα (μαύρο ή κόκκινο) και το αν είναι φιγούρα ή όχι. Όλες αυτές τις πληροφορίες μπορούμε να τις διαχειριστούμε με πολλούς τρόπους, όμως φανταστείτε πόσο θα διευκολυνόμασταν αν ορίζαμε μεταβλητές που δεν θα ήταν ακέραιες, πραγματικές ή μιγαδικές, αλλά τραπουλόχαρτα (αντικείμενα), το καθένα με όλες τις πληροφορίες του.

Για να δημιουργήσουμε αντικείμενα πρέπει πρώτα να προγραμματίσουμε μια "μηχανή" που θα τα δημιουργεί. Αυτή λέγεται **κλάση** (**class**) και στην πιο απλή μορφή της είναι κάτι τέτοιο:

```
>>> class card():
pass
```

Η συντακτική της δομή είναι παρόμοια με τη δομή της def. Βάζουμε πρώτα τη λέξη class, μετά το όνομα της κλάσης με παρενθέσεις, άνω-κάτω τελεία και από την επόμενη γραμμή αρχίζουμε τις εντολές της κλάσης. Εδώ γράψαμε μόνο την εντολή pass. Στην Python, η pass είναι μια εντολή η οποία δεν κάνει τίποτα. Χρησιμοποιείται όπου είναι απαραίτητο να γράψουμε κάτι για συντακτικούς λόγους κι εμείς δε θέλουμε να βάλουμε τίποτε άλλο.

Η παραπάνω εντολή λοιπόν δημιούργησε απλώς μια κλάση με όνομα card και τίποτε άλλο. Αυτό και μόνο αρκεί για να δημιουργήσουμε μερικά αντικείμενα (τραπουλόχαρτα):

```
>>> c1=card()
>>> c2=card()
```

[1] Εξυπακούεται ότι οι διδάσκοντες σε καμία περίπτωση δεν ενθαρρύνουν την χαρτοπαιξία. Τα τραπουλόχαρτα και οι τράπουλες χρησιμοποιούνται εδώ επειδή αποτελούν ιδιαίτερα επιτυχημένα παραδείγματα για την κατανόηση των εξεταζόμενων εννοιών, όπως άλλωστε συμβαίνει και σε πλείστα επιστημονικά συγγράμματα θεωρίας πιθανοτήτων και στατιστικής.

Και πάλι η ομοιότητα με τις συναρτήσεις είναι εμφανής. Τα c1 και c2 είναι μεταβλητές που παίρνουν ως τιμή αυτό που επιστρέφει η card(). Η card() συμπεριφέρεται σα μια μηχανή που φτιάχνει τραπουλόχαρτα. Όμως αυτά τα τραπουλόχαρτα προς το παρόν είναι άδεια. Για να γράψουμε πάνω τους χρησιμοποιούμε dot notation και ορίζουμε χαρακτηριστικά (ιδιότητες) και τιμές:

```
>>> c1.value='5'
>>> c1.symbol='d'
>>> c1.color='R'
>>> c1.fig=False

>>> c2.value='K'
>>> c2.symbol='c'
>>> c2.color='B'
>>> c2.fig=True
```

Έτσι ορίσαμε ότι το τραπουλόχαρτο c1 είναι 5 καρώ (d=diamond), κόκκινο (R=red), και όχι φιγούρα (c1.fig=False), ενώ το c2 είναι ρήγας (K=King), σπαθί (c=club), μαύρο (B=black) και φιγούρα (c2.fig=True).

Και φυσικά μπορούμε να δούμε και να χρησιμοποιήσουμε πλέον αυτές τις τιμές:

```
>>> print c1.value, c1.symbol, c2.value, c2.symbol
5dKc
```

Αυτή είναι η βασική ιδέα. Και τώρα αρχίζει μια σειρά βελτιώσεων. Για παράδειγμα, παρατηρήστε ότι σύμφωνα με τα παραπάνω για κάθε νέο τραπουλόχαρτο που δημιουργούμε θα πρέπει να δίνουμε πέντε εντολές. Μία για να δημιουργηθεί το αντικείμενο και τέσσερις για να καταχωρηθούν οι τιμές. Δε θα ήταν καλύτερο οι τιμές να πέρναγαν ως παράμετροι στην πρώτη εντολή; Ή κάποιες από αυτές (color, fig) να υπολογίζονταν αυτόματα από άλλες (symbol, value); Να πώς μπορούμε να το πετύχουμε:

```
>>> class card():
    def __init__(self,val,sym):
        self.value=val
        self.symbol=sym
        if self.symbol in "sc": self.color='B'
        else: self.color='R'
        if self.value in "JQK": self.fig=True
        else: self.fig=False

>>> c1=card('5','d')
>>> print c1.value, c1.symbol, c1.color, c1.fig
5 d R False
```

Αλλάξαμε τον ορισμό της κλάσης αντικαθιστώντας την pass με τον ορισμό μιας συνάρτησης. Και δείτε το αποτέλεσμα: με μία μόνο εντολή δημιουργήσαμε το 5 καρώ δύο από τις τιμές υπολογίστηκαν αυτόματα.

Ας εξετάσουμε τη συνάρτηση που τα έκανε όλα αυτά. Το όνομά της είναι __init__ (δύο underscores, init, δυο underscores) και ανήκει σε μια ομάδα ειδικών συναρτήσεων με παρόμοια ονόματα[2], οι οποίες καθορίζουν πώς θα συμπεριφέρονται τα αντικείμενα που θα δημιουργεί η κλάση. Η __init__ καθορίζει τί θα γίνεται μόλις δημιουργείται ένα αντικείμενο. Επειδή δεν μπορεί να γνωρίζει το όνομα που θα διαλέξουμε για το αντικείμενο (πχ. αν το ονομάσουμε c1 ή c2 ή κάπως αλλιώς), αναφέρεται σε αυτό με την προεπιλεγμένη ονομασία "self". Το "self" μπαίνει πάντα ως πρώτο όρισμα στην παρένθεση της __init__. Στην περίπτωσή μας η παρένθεση περιέχει και άλλα δύο ορίσματα, τα val και sym. Με αυτόν τον τρόπο, όταν εμείς δώσουμε την εντολή c1=card('5','d'), το self θα γίνει c1, το val θα γίνει '5' και το sym θα γίνει 'd'. Στη συνέχεια θα

[2] __str__, __del__, __new__, __cmp__, κλπ. Δείτε στο on-line help για τον πλήρη κατάλογο.

εκτελεστούν οι δύο πρώτες εντολές της __init__ και θα πάρουν τιμή τα c1.value και c1.symbol. Και ανάλογα με αυτές τις τιμές, οι εντολές if-else που ακολουθούν υπολογίζουν τις τιμές των c1,color και c1.fig.

Σημαντική λεπτομέρεια: Προσέξτε ότι ενώ κατά την κλήση c1=card('5','d') οι τιμές val και sym γράφονται στην παρένθεση της card, κατά τον ορισμό της κλάσης αυτές οι παράμετροι τοποθετούνται στην παρένθεση της __init__, ενώ η παρένθεση της κλάσης card μένει κενή! Αργότερα θα δούμε ότι η παρένθεση των κλάσεων χρησιμοποιείται για άλλο σκοπό (για να ορίσει κληρονομικότητα από άλλες κλάσεις).

Προσέξτε κάτι άλλο τώρα. Ενώ μπορούμε να τυπώνουμε τις τιμές όλων των ιδιοτήτων των αντικειμένων που ορίζουμε:

```
>>> print c1.value, c1.symbol, c1.color, c1.fig
5 d R False
```

... αν προσπαθήσουμε να τυπώσουμε τα ίδια τα αντικείμενα, παίρνουμε μόνο αναφορές στις διευθύνσεις μνήμης στις οποίες βρίσκονται:

```
>>> print c1
<__main__.card instance at 0x011CAAF8>
```

Κάθε αντικείμενο μπορούμε να το φανταστούμε σαν ένα κουτί που περιέχει τιμές για διάφορες ιδιότητες, ενδεχομένως συναρτήσεις/μεθόδους, ακόμα και άλλα αντικείμενα/κουτιά. Συνεπώς αν δεν ορίσουμε με κάποιον τρόπο τί ακριβώς σημαίνει η εκτύπωση ενός αντικειμένου (πχ. εκτύπωση όλων ή κάποιων από τα περιεχόμενά του, και με ποια σειρά) προφανώς δεν μπορούμε να έχουμε την απαίτηση από την Python να το τυπώσει. Αυτός ο ορισμός γίνεται με την ειδική συνάρτηση __str__. Δείτε το νέο ορισμό της κλάσης που πλέον έχουμε προσθέσει και την __str__ κάτω από την __init__:

```
>>> class card():
    def __init__(self,val,sym):
        self.value=val
        self.symbol=sym
        if self.symbol in "sc": self.color='B'
        else: self.color='R'
        if self.value in "JQK": self.fig=True
        else: self.fig=False
    def __str__(self):
        return self.value+self.symbol
```

Η __str__ δε χρειάζεται άλλη παράμετρο παρά μόνο το αντικείμενο (self). Πρέπει πάντα να επιστρέφει ένα αλφαριθμητικό (string). Εδώ τη βάλαμε να επιστρέφει το self.value+self.symbol. Συνεπώς μπορούμε πλέον να δώσουμε:

```
>>> c1=card('5','d')
>>> print c1
5d
```

Και φυσικά, εκτός από τις ειδικές συναρτήσεις μπορούμε να προσθέσουμε μέσα στην κλάση και δικές μας οι οποίες θα καλούνται ως μέθοδοι με dot notation. Για παράδειγμα:

```
>>> class card():
    def __init__(self,val,sym):
        self.value=val
        self.symbol=sym
        if self.symbol in "sc": self.color='B'
        else: self.color='R'
```

```
        if self.value in "JQK": self.fig=True
        else: self.fig=False
    def __str__(self):
        return self.value+self.symbol
    def detailed_info(self):
            print 'Αξία=',self.value, '     Σύμβολο=',self.symbol
            print 'Χρώμα=',self.color, '    Φιγούρα=',self.fig

>>> c1=card('5','d')
>>> c1.detailed_info()
Αξία= 5     Σύμβολο= d
Χρώμα= R    Φιγούρα= False
```

Και τώρα μπορούμε να δημιουργήσουμε 52 τραπουλόχαρτα για να τα χρησιμοποιήσουμε στο παιχνίδι μας, ή να κάνουμε κάτι πολύ καλύτερο. Τα τραπουλόχαρτα περιέχονται σε τράπουλες. Ας φτιάξουμε λοιπόν μία νέα κλάση που θα δημιουργεί πλήρεις τράπουλες με όλα τα τραπουλόχαρτα. Θα δούμε έτσι πως μια κλάση μπορεί να καλέσει αντικείμενα άλλης κλάσης. Δείτε εδώ τον πλήρη κώδικά της και στη συνέχεια θα εξηγήσουμε ένα-ένα τα μέρη του:

```
import random

class deck():
    values="A23456789TJQK"  # Όλες οι αξίες
    symbols="shcd"          # Όλα τα σύμβολα
    content=[] # Χαρτιά που βρίσκονται στην τράπουλα
    pile=[]                 # Χαρτιά που βγήκαν από την τράπουλα
    def __init__(self):
        self.content=[]
        self.pile=[]
        for s in self.symbols:
            for v in self.values:
                c=card(v,s)
                self.content.append(c)
    def __str__(self):
        s=""
        cntr=0
        for i in self.content:
            s=s+str(i)+" "
            cntr=cntr+1
            if cntr%13==0: s=s+'\n'
        if s[-1]<>'\n': s=s+'\n'
            s=s+str(len(self.content))+"-"+str(len(self.pile))
        return s
    def shuffle(self):
        random.shuffle(self.content)
    def draw(self):
        if len(self.content)<1: return "empty"
        c=self.content[0]
        self.content=self.content[1:]
        self.pile.append(c)
        return c
    def collect(self):
        self.content=self.content+self.pile
        self.pile=[]
```

Στην αρχή του κώδικα, αμέσως μετά την import και το όνομα της κλάσης deck βλέπουμε τέσσερις μεταβλητές. Η values περιέχει σε ένα αλφαριθμητικό όλες (13) τις πιθανές αξίες ενός τραπουλόχαρτου (για να έχουμε ομοιομορφία αργότερα στην εκτύπωση, χρησιμοποιούμε για κάθε

αξία μόνο ένα χαρακτήρα, έτσι το 10 γράφτηκε ως T). Η symbols περιέχει με τον ίδιο τρόπο τα 4 σύμβολα (s=spade/ μπαστούνι, h=heart/ καρδιά, c=club/ σπαθί, d=diamond/ καρώ). Αργότερα (στην __init__) θα συνδυάσουμε τις 13 αξίες με τα 4 σύμβολα για να φτιάξουμε όλα τα 13x4=52 τραπουλόχαρτα. Και θα χρειαστεί να τα αποθηκεύσουμε σε μία λίστα. Αυτή είναι η content που προς το παρόν είναι κενή. Γεμίζει και αυτή στην __init__. Τέλος, έχουμε ορίσει μια άλλη λίστα, την pile. Αυτή χρησιμοποιείται ως μνήμη για να "θυμάται" η τράπουλα ποιά φύλλα έχει μοιράσει. Αυτή η μνήμη δεν είναι απαραίτητη, όμως διευκολύνει όταν τελειώσει ένα παιχνίδι και θα πρέπει να ξανασυγκεντρωθούν όλα τα χαρτιά στην τράπουλα για να αρχίσει το επόμενο. Αντί να τα μαζεύει από όλους τους παίκτες, αρκεί να τα μαζέψει από την pile. Αυτό το κάνει η μέθοδος collect(). Παρατηρήστε τώρα ότι αυτές οι τέσσερις μεταβλητές είναι μέσα στην κλάση deck αλλά έξω από τις μεθόδους. Αυτό σημαίνει ότι για να τις προσπελάσει μια μέθοδος θα πρέπει να τις καλέσει με dot notation από το αντικείμενο self, πχ. self.values.

Πάμε τώρα να δούμε την __init__. Έχει μόνο ένα όρισμα, το self. Αυτό σημαίνει ότι για να δημιουργήσουμε μία τράπουλα δε δίνουμε καμία παράμετρο στην deck. Γράφουμε απλώς κάτι τέτοιο:

```
>>> d=deck()
```

Το πρώτο for περνάει από όλα τα σύμβολα, το δεύτερο από όλες τις αξίες, και προσέξτε πώς η c=card(v,s) δημιουργεί ένα τραπουλόχαρτο το οποίο μπαίνει στο τέλος της content με την κλήση self.content.append(c). Όταν τελειώσουν οι δυο for όλα τα τραπουλόχαρτα θα είναι με τη σειρά στη λίστα content. Πράγματι, αν τυπώσουμε την d θα δούμε:

```
>>> print d
As 2s 3s 4s 5s 6s 7s 8s 9s Ts Js Qs Ks
Ah 2h 3h 4h 5h 6h 7h 8h 9h Th Jh Qh Kh
Ac 2c 3c 4c 5c 6c 7c 8c 9c Tc Jc Qc Kc
Ad 2d 3d 4d 5d 6d 7d 8d 9d Td Jd Qd Kd
52-0
```

Τον τρόπο εκτύπωσης καθορίζει η __str__. Επιστρέφει ένα αλφαριθμητικό s στο οποίο έχουμε βάλει όλα τα στοιχεία της self.content, 13 ανά γραμμή, και στο τέλος το μέγεθος της content και το μέγεθος της pile. Εδώ αξίζει προσοχή η s=s+str(i)+" ". Το i είναι τραπουλόχαρτο, επομένως το str(i) είναι το string που θα επέστρεφε η print i, δηλαδή αυτό που έχουμε ορίσει στην __str__ της κλάσης card.

Και βέβαια, δεν παίζουμε αν πρώτα δεν ανακατέψουμε την τράπουλα. Αυτό το κάνει η μέθοδος shuffle, η οποία καλεί μια μέθοδο με όμοιο τρόπο από το module random (για αυτό το λόγο κάναμε import την random στην αρχή του κώδικά μας). Προσέξτε ότι έχουμε δυο μεθόδους με το ίδιο όνομα. Η μία είναι η self.shuffle() και η άλλη είναι η random.shuffle(). Τις ξεχωρίζουμε με dot notation. Τώρα αρκεί να γράψουμε:

```
>>> d.shuffle()
```

... και πλέον έχουμε ανακατέψει την τράπουλα:

```
>>> print d
8d 3c 2c Ts Ks 9c Jc 5h 2h Kd 3h Td 5c
5d 7s 8h 9d Ad 5s Tc 7c 4c 6c 7d Qh Qs
3d Ah Ac Js Qc 4s 3s 6d Jd As Kh 4h 9h
Th 9s Jh 6h 4d 2s 7h 2d Qd Kc 8c 8s 6s
52-0
```

Τί άλλο θέλουμε να κάνουμε με μια τράπουλα; Να μοιραστούμε φύλλα. Αυτό κάνει η επόμενη μέθοδος draw() που έχει προγραμματιστεί στην κλάση. Κάθε φορά που καλείται, η draw() επιστρέφει

το φύλλο που βρίσκεται πρώτο (self.content[0]) στην τράπουλα, εκτός αν είναι άδεια, οπότε επιστρέφει "empty". Το φύλλο αφαιρείται από τη λίστα content και ένα αντίγραφό του αποθηκεύεται στην pile.

```
>>> h=d.draw()
>>> print h
8d
>>> print d
3c 2c Ts Ks 9c Jc 5h 2h Kd 3h Td 5c 5d
7s 8h 9d Ad 5s Tc 7c 4c 6c 7d Qh Qs 3d
Ah Ac Js Qc 4s 3s 6d Jd As Kh 4h 9h Th
9s Jh 6h 4d 2s 7h 2d Qd Kc 8c 8s 6s
51-1
```

Η τελευταία μέθοδος η collect είναι πολύ απλή. Προσθέτει την pile στο τέλος της content και μετά αδειάζει την pile. Με αυτόν τον τρόπο ξανασυγκεντρώνει όλα τα φύλλα της τράπουλας για να ξεκινήσει νέο παιχνίδι.

```
>>> for i in range(10): h=d.draw()
>>> print d
Td 5c 5d 7s 8h 9d Ad 5s Tc 7c 4c 6c 7d
Qh Qs 3d Ah Ac Js Qc 4s 3s 6d Jd As Kh
4h 9h Th 9s Jh 6h 4d 2s 7h 2d Qd Kc 8c
8s 6s
41-11
>>> d.collect()
>>> print d
Td 5c 5d 7s 8h 9d Ad 5s Tc 7c 4c 6c 7d
Qh Qs 3d Ah Ac Js Qc 4s 3s 6d Jd As Kh
4h 9h Th 9s Jh 6h 4d 2s 7h 2d Qd Kc 8c
8s 6s 8d 3c 2c Ts Ks 9c Jc 5h 2h Kd 3h
52-0
```

Όμως χρειάζεται λίγη προσοχή κατά τη χρήση. Επειδή η collect() από μόνη της δεν αφαιρεί τα φύλλα από τους παίκτες, θα πρέπει να θυμηθούμε να αδειάσουμε τα χέρια των παικτών με ξεχωριστό κώδικα.

Μέχρι τώρα είπαμε πολλά για χαρτιά και τράπουλες, όμως δεν είπαμε τίποτα για το παιχνίδι που θα προγραμματίσουμε. Αυτό δείχνει ότι οι κλάσεις που φτιάξαμε είναι ανεξάρτητες από το παιχνίδι και φυσικά θα μπορούσαν να χρησιμοποιηθούν σε οποιοδήποτε παιχνίδι με τράπουλα. Αυτό σημαίνει ότι αξίζει να σώσουμε τις κλάσεις μας σε ξεχωριστό αρχείο και να τις συμπεριλάβουμε με import ως module[3] από το αρχείο του κυρίως προγράμματος που θα φτιάξουμε. Βάλτε λοιπόν τις δυο κλάσεις σε ένα αρχείο , πρώτα την card(), μετά την deck() και μη ξεχάσετε την εντολή import random στην αρχή. Σώστε το αρχείο με το όνομα playing_cards.py στον κατάλογο που σώζετε όλα σας τα προγράμματα Python, και οι κλάσεις θα είναι πάντα διαθέσιμες μόλις γράψουμε "from playing_cards import * ".

8.3 Χρήση Αντικειμένων (Ο Κώδικας του Παιχνιδιού)

[3] Αυτό άλλωστε είναι όλα τα modules που χρησιμοποιήσαμε ως τώρα. Συλλογές κλάσεων που περιέχουν μεθόδους και δεδομένα. Δείτε για παράδειγμα με έναν text editor τα περιεχόμενα του αρχείου random.py στο φάκελο Lib της Python.

Το παιχνίδι που θα προγραμματίσουμε έχει τους εξής κανόνες:

1. Παίζεται με δυο παίκτες (ο άνθρωπος εναντίον του υπολογιστή) και μία τράπουλα.
2. Στην αρχή του παιχνιδιού μοιράζονται 7 φύλλα σε κάθε παίκτη και 1 φύλλο ανοικτό στο τραπέζι.
3. Με τυχαίο τρόπο καθορίζεται ποιός παίκτης θα παίξει πρώτος.
4. Ο κάθε παίκτης, όταν είναι η σειρά του να παίξει θα πρέπει να ρίξει στο τραπέζι ένα από τα φύλλα του, αρκεί να συμφωνεί σε αξία ή σύμβολο με το προηγούμενο φύλλο που έπεσε στο τραπέζι (πχ. αν στο τραπέζι είναι το 5 κούπα, ο παίκτης θα πρέπει να ρίξει ή 5 ή κούπα). Αν δεν έχει κατάλληλο φύλλο θα πρέπει να τραβήξει από την τράπουλα (όσες φορές χρειαστεί) μέχρι να βρει.
5. Νικητής του παιχνιδιού είναι όποιος πετάξει πρώτος όλα τα φύλλα του.
6. Αν κάποιος παίκτης προσπαθήσει να τραβήξει φύλλο αλλά η τράπουλα έχει τελειώσει, το παιχνίδι σταματά και νικητής ανακηρύσσεται όποιος κρατάει τα λιγότερα φύλλα.

Για τον προγραμματισμό του παιχνιδιού θα χρησιμοποιήσουμε το module playing_cards που φτιάξαμε προηγουμένως, 5 συναρτήσεις και λίγες γραμμές κώδικα για το κυρίως πρόγραμμα. Επίσης θα έχουμε την ευκαιρία να γνωρίσουμε την έννοια των καθολικών μεταβλητών (global variables) αφού θα χρησιμοποιήσουμε τέσσερις καθολικές μεταβλητές στο πρόγραμμά μας: την d που θα είναι μια τράπουλα και τρεις λίστες, τις table, computer_hand, human_hand, που θα περιέχουν τα τραπουλόχαρτα που βρίσκονται στο τραπέζι, στα "χέρια" του υπολογιστή, και στα χέρια του ανθρώπου-παίκτη, αντίστοιχα. Θα εξηγήσουμε τη λειτουργία όλων των συνιστωσών του προγράμματος βήμα-βήμα, καθώς θα συμπληρώνουμε τον κώδικα στο αρχείο.

Καταρχήν, ανοίγουμε ένα νέο αρχείο και γράφουμε στην αρχή του την εντολή:

```
from playing_cards import *
```

Αυτό θα μας δώσει τη δυνατότητα να κάνουμε χρήση[4] των κλάσεων card και deck.

Η πρώτη συνάρτηση που θα προσθέσουμε καθορίζει τον τρόπο που θα "σκέφτεται" και θα παίζει ο υπολογιστής:

```
def computer_plays():
    global d, table, computer_hand
    target_val=table[-1].value
    target_sym=table[-1].symbol
    for c in computer_hand:
        if c.value==target_val or c.symbol==target_sym:
            print "Ο Η/Τ ξίρλεη",c
            computer_hand.remove(c)
            table.append(c)
            if len(computer_hand)<1: return "computer wins"
            else: return "continue"
    new_card=d.draw()
    if new_card=="empty": return "count"
    else:
        print "Ο Η/Τ ηξαβάεη θύιιν."
        computer_hand.append(new_card)
return computer_plays()
```

[4] Αν τυχόν η Python δεν μπορέσει να διαβάσει το αρχείο προσθέστε στην πρώτη γραμμή του τη δήλωση κωδικοσελίδας (πχ. #-*- coding: cp1253 -*-).

Στην πρώτη γραμμή μετά το όνομα της συνάρτησης βλέπουμε την εντολή global και τα ονόματα των μεταβλητών d, table και computer_hand. Αυτό σημαίνει ότι οι μεταβλητές αυτές είναι καθολικές. Δηλαδή είναι ορισμένες στο κυρίως πρόγραμμα και η συνάρτηση τις διαβάζει από εκεί. Αυτό σημαίνει επίσης ότι αν η συνάρτηση αλλάξει την τιμή τους αυτή η αλλαγή θα μεταφερθεί και στο κυρίως πρόγραμμα και σε κάθε άλλη συνάρτηση που τις χρησιμοποιεί. Για να πετύχουμε παρόμοια συμπεριφορά με συνηθισμένες (τοπικές) μεταβλητές θα έπρεπε να τις περνάμε ως ορίσματα στη συνάρτηση (για να τις διαβάσει) και μετά να επιστρέφουμε τις τιμές τους με το όνομα της συνάρτησης (για να περάσει η νέα τιμή στο κυρίως πρόγραμμα). Έτσι, τώρα το τελευταίο φύλλο στη λίστα table (το table[-1]) στο οποίο αναφέρονται οι δυο επόμενες εντολές είναι το ίδιο και κοινό παντού κι όχι κάποιο αντίγραφο ή συνωνυμία.

Ως προς τη λογική της συνάρτησης τώρα, στις μεταβλητές target_val και target_sym αποθηκεύονται η αξία και το σύμβολο αντίστοιχα του τελευταίου φύλλου που έπεσε στο τραπέζι. Στη συνέχεια, η for ελέγχει με τη σειρά ένα-ένα τα χαρτιά στο computer_hand και αν κάποιο βρεθεί να έχει ίδια αξία με το target_val ή ίδιο σύμβολο με το target_sym τότε εκτυπώνεται το μήνυμα και το χαρτί φεύγει από το computer_hand με την remove() και μεταφέρεται στο table με την append(). Μετά γίνεται ένας έλεγχος μήπως ο Η/Υ έχει πειράξει όλα τα χαρτιά του και αν αυτό συμβαίνει η συνάρτηση επιστρέφει με τιμή το string "computer wins". Διαφορετικά επιστρέφει με την τιμή "continue".

Αν τελειώσει η for και δεν βρεθεί κατάλληλο φύλλο, τότε η d.draw() επιχειρεί να τραβήξει ένα νέο φύλλο από την τράπουλα. Μια if ελέγχει μήπως η τράπουλα ήταν άδεια και δεν επέστρεψε φύλλο και σε αυτήν την περίπτωση η συνάρτηση επιστρέφει ως τιμή τη λέξη "count". Σε αντίθετη περίπτωση η else τυπώνει το μήνυμα ότι ο Η/Υ τραβάει φύλλο, προσθέτει το φύλλο στο computer_hand και τέλος εκτελείται η return computer_plays(). Αυτό το τελευταίο θέλει λίγη προσοχή. Σημαίνει "επέστρεψε την τιμή που θα σου δώσει μια νέα κλήση της computer_plays()". Βλέπουμε λοιπόν ότι η συνάρτηση καλεί τον εαυτό της, δηλαδή είναι όπως λέμε **αναδρομική** (**recursive**). Αυτός είναι ένας βολικός[5] τρόπος να ξεκινήσει η διαδικασία από την αρχή ώστε να ελεγχθεί το νέο computer_hand.

Τελικά η computer_plays() επιστρέφει ένα αλφαριθμητικό με την κατάσταση της παρτίδας μέχρι εκείνη τη στιγμή. Το αλφαριθμητικό θα είναι ή "continue", ή "computer wins", ή "count", σηματοδοτώντας αν το παιχνίδι συνεχίζεται ή τελείωσε και με ποιον τρόπο. Η συνάρτηση πάντα θα επιστρέφει κάποια από αυτές τις τιμές, κι αν αυτό δεν συμβεί με την πρώτη κλήση, μετά καλεί τον εαυτό της για κάθε νέο φύλλο που τραβάει ώστε να επαναληφθεί η διαδικασία.

Ακολουθεί η συνάρτηση που εκτελείται όταν είναι η σειρά του ανθρώπου να παίξει. Αν και θα περιμέναμε να είναι πιο απλή από την προηγούμενη (πχ. να διαβάζει την επιλογή του παίκτη και να την εκτελεί) ο κώδικας είναι λίγο μεγαλύτερος για δύο λόγους: (α) τυπώνει όλες τις πληροφορίες που χρειάζονται για να αποφασίσει ο παίκτης το φύλλο που θα πετάξει, και (β) προφυλάσσει το πρόγραμμα από τυχόν ανθρώπινα σφάλματα κατά την είσοδο:

```
def human_plays():
    global d, table, human_hand
    print
    print "Η τράπουλα έχει", len(d.content), "φύλλα"
    print "Ο Η/Υ έχει", len(computer_hand), "φύλλα"
    print "Στο τραπέζι έχουν πέσει", len(table), "φύλλα"
```

[5] Αλλά όχι ιδιαίτερα αποτελεσματικός αφού ξαναελέγχει όλα τα προηγούμενα φύλλα από την αρχή. Θα μπορούσαμε να βελτιώσουμε αυτό το σημείο με λίγο περισσότερο κώδικα, όμως θα περιπλέαμε τη συνάρτηση αρκετά περισσότερο χωρίς λόγο, αφού προς το παρόν δε μας ενδιαφέρει η βελτίωση της ταχύτητας του κώδικα.

```
    t=table[-1]
    print "Το πάνω φύλλο είναι",t
    print "Τα φύλλα σου είναι",len(human_hand)
    HHS=[str(x) for x in human_hand]
    print HHS
    print
    print "Διάλεξε ποιο θα πετάξεις"
    print "ή πάτα σκέτο ENTER για να τραβήξεις"
    sel=raw_input()
    if sel=="":
        new_card=d.draw()
        if new_card=="empty": return "count"
        else:
            human_hand.append(new_card)
            return human_plays()
    else:
        if not(sel in HHS):
            print sel, ";;;   Γελ Δεν έχεις τέτοιο φύλλο."
            return human_plays()
        target_val=t.value
        target_sym=t.symbol
        if sel[0]<>target_val and sel[1]<>target_sym:
            print "Δεν επιτρέπεται να ρίξεις το φύλλο ",sel
            return human_plays()
        print "Ρίχνεις το",sel
        ind=HHS.index(sel)
        selc=human_hand[ind]
        human_hand.remove(selc)
        table.append(selc)
        if len(human_hand)<1: return "human wins"
        else: return "continue"
```

Η αρχή του κώδικα της human_plays() δεν έχει εκπλήξεις. Πάλι η global δηλώνει τις τρεις καθολικές μεταβλητές που χρησιμοποιεί η συνάρτηση και μετά μια σειρά από εντολές print τυπώνουν διάφορες βοηθητικές πληροφορίες. Εξήγηση χρειάζεται μόνο η χρήση της βοηθητικής μεταβλητής που ορίζεται με την εντολή HHS=[str(x) for x in human_hand]: επειδή δεν έχουμε ορίσει πώς τυπώνεται μια λίστα από τραπουλόχαρτα, αν τυπώναμε απευθείας την human_hand θα παίρναμε μια σειρά από διευθύνσεις μνήμης. Αντιθέτως, η νέα λίστα που δημιουργούμε από τις εκτυπώσεις του κάθε στοιχείου της αρχικής είναι επίσης εκτυπώσιμη.

Για την καταγραφή της επιλογής του παίκτη δε χρησιμοποιήθηκε η raw_input(). Λειτουργεί όπως η input() αλλά θεωρεί αυτομάτως ως αλφαριθμητικό την είσοδο του χρήστη, πράγμα που στη συγκεκριμένη περίπτωση μας εξυπηρετεί. Η if που ακολουθεί ελέγχει μήπως ο παίκτης πατήσει απλώς ENTER χωρίς να γράψει κάποιο φύλλο. Αυτό σημαίνει ότι ο παίκτης θέλει να τραβήξει από την τράπουλα. Μετά από έναν έλεγχο μήπως η τράπουλα έχει αδειάσει, το νέο φύλλο προστίθεται στην human_hand και η συνάρτηση καλεί αναδρομικά τον εαυτό της για να επαναλάβει τη διαδικασία. Σε αυτά τα σημεία η human_plays() μοιάζει πολύ με την computer_plays() που είδαμε νωρίτερα.

Ακολουθούν δύο έλεγχοι, πρώτα αν το φύλλο που έγραψε ο παίκτης πράγματι το έχει στα χέρια του και ύστερα αν το φύλλο ταιριάζει με αυτό που είναι πάνω στο τραπέζι. Και στις δύο περιπτώσεις η συνάρτηση ξανακαλεί τον εαυτό της, διαφορετικά μεταφέρει το φύλλο από την human_hand στην table. Και πάλι παρομοίως με την computer_plays(), η human_plays() επιστρέφει ένα κατάλληλο αλφαριθμητικό ("human wins", "continue", ή "count") για να δηλώσει το αποτέλεσμα.

Κατά τη διάρκεια του παιχνιδιού η εκτέλεση των συναρτήσεων computer_plays() και human_plays() θ εναλλάσσεται συνεχώς. Αυτό μπορεί να επιτευχθεί με την επαναληπτική κλήση της συνάρτησης next_turn() που ακολουθεί. Η συνάρτηση δέχεται ως όρισμα έναν αριθμό που δηλώνει τον παίκτη που έχει σειρά να παίξει (+1 ο άνθρωπος, -1 ο Η/Υ) και επιστρέφει (για την ακρίβεια μεταφέρει) την έξοδο της αντίστοιχης συνάρτησης human_plays() ή computer_plays().

```
def next_turn(who_plays):
    if who_plays==1:
        print
        print "------------------- ΣΕΙΡΑ ΠΑΙΚΤΗ -------------------"
        return human_plays()
    else:
        print
        print "-------------------- ΣΕΙΡΑ Η/Υ ---------------------"
        print
        return computer_plays()
```

Αν η παρτίδα τελειώσει, η ακόλουθη συνάρτηση evaluate() δέχεται ως όρισμα ένα από τα αλφαριθμητικά "human_wins", "computer_wins", ή "count" και τυπώνει τα κατάλληλα μηνύματα. Ειδικά στην περίπτωση του "count" μετράει και τυπώνει το πλήθος των φύλλων που έχει ο κάθε παίκτης.

```
def evaluate(result):
    global computer_hand, human_hand
    if result=="count":
        ch=len(computer_hand)
        hh=len(human_hand)
        print "Η ηξάπνπια ηειείσζε, ν Η/Τ έρεη",ch, "θύπια θαη ν παίθηεο", hh
        if ch>hh: result="human wins"
        if ch<hh: result="computer wins"
        if ch==hh: print "Ι□ΟΠΑΛΙΑ!!!"
    if result=="human wins": print "□ΤΓΥΑΡΗΣΗΡΙΑ. ΚΔΡΓΙ□Δ□!!!"
    if result=="computer wins": print "Ο Η/Τ ΚΔΡΓΙ□Δ."
    print
```

Η συνάρτηση initial() που ακολουθεί αναλαμβάνει να εκτελέσει όλες τις λειτουργίες που γίνονται στην αρχή μίας παρτίδας. Η d.collect() μαζεύει τα χαρτιά στην τράπουλα, καθαρίζονται το τραπέζι και τα "χέρια" των παικτών, μετά η d.shuffle() ανακατεύει την τράπουλα, και η d.draw() καλείται για να μοιράσει επτά φύλλα στους παίκτες και ένα στο τραπέζι. Τέλος η random.random() καλείται και χρησιμοποιείται για να καθορίσει ποιος παίκτης θα παίξει πρώτος. Το αποτέλεσμα (+1 ή -1) επιστρέφεται ως τιμή της initial() για να χρησιμοποιηθεί στο κυρίως πρόγραμμα.

```
def initial():
    global d, table, computer_hand, human_hand
    print "Μαδεύσ ηα ραξηηά..."
    d.collect()
    table=[]
    computer_hand=[]
    human_hand=[]

    print "Αλαθαηεύσ ηελ ηξάπνπια..."
    d.shuffle()

    print "Μνηξάδσ ηα θύπια..."
    table.append(d.draw())
    print "□ην ηξαπέδη έπεδε",table[-1]
    for i in range(7):
```

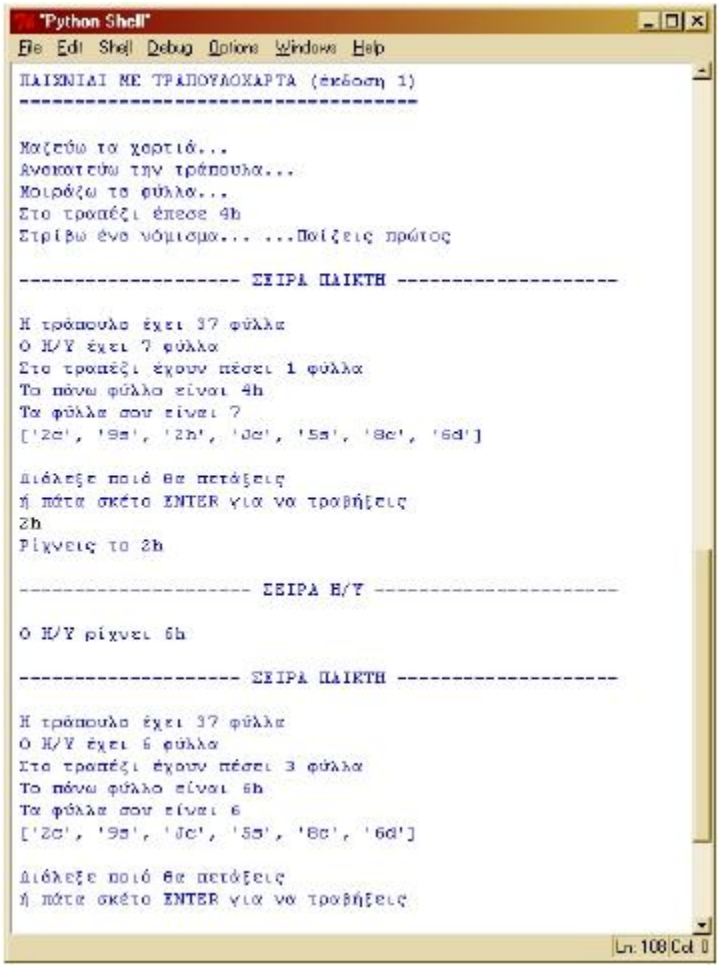
```
        human_hand.append(d.draw())
        computer_hand.append(d.draw())

    print "□ηξίβσ έλα λόκηζκα...",
    if random.random()<0.5:
        who=-1
        print "...Παίδεηο πξώηνο"
    else:
        who=1
        print "...Ο Η/Τ παηδεη πξώηνο"
    return who
```

Και στο τέλος του αρχείου προσθέτουμε τον ακόλουθο κώδικα ως κυρίως πρόγραμμα:

```
# ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ
print "ΠΑΙΧΝΙΔΙ ΜΕ ΤΡΑΠΟΥΛΟΧΑΡΤΑ (έκδοση 1)"
print "======================================"
print

# ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ
d=deck()
table=[]
computer_hand=[]
human_hand=[]

again="y"
while again=="y":
    who=initial()

    result="continue"
    while result=="continue":
        who=-who
        result=next_turn(who)
    evaluate(result)

    print "Θέλεις να ξαναπαίξουμε;"
    again=raw_input("γράψε y ή n: ")
print "Τέλος Προγράμματος"
```

Μετά την εκτύπωση των αρχικών μηνυμάτων και την αρχικοποίηση των καθολικών μεταβλητών, ακολουθούν δυο while loops. Το εξωτερικό χρησιμοποιείται για να παίζουμε πολλές παρτίδες. Κάθε φορά που τελειώνει, ρώτα το χρήστη αν θέλει να ξαναπαίξει. Κάθε φορά που ο χρήστης απαντά "y" καλείται η initial() και ξεκινά μια καινούρια παρτίδα. Το εσωτερικό while είναι υπεύθυνο για την εναλλαγή της σειράς των παικτών. Όσο το result της next_turn() είναι "continue" αλλάζει το πρόσημο που καθορίζει τη σειρά του παίκτη (who = - who) και ξαναεκτελείται η next_turn(). Μόλις το result πάψει να είναι "continue" το παιχνίδι έχει τελειώσει και καλείται η evaluate (result) για να γράψει τα κατάλληλα μηνύματα.

Σώστε το αρχείο με ένα κατάλληλο όνομα (πχ. card_game.py) και τρέξτε το. Μόλις το τρέξετε θα δείτε μια οθόνη σαν την ακόλουθη. Εδώ ο παίκτης έτυχε να παίζει πρώτος και έριξε το 2h πάνω στο 4h που ήταν στο τραπέζι. Ο Η/Υ συνέχισε με 6h και τώρα είναι πάλι η σειρά του παίκτη.



8.4 Κληρονομικότητα κλάσεων

Ας υποθέσουμε ότι θέλουμε να φτιάξουμε μια νέα έκδοση του παιχνιδιού που να παίζεται με 2, 3 ή περισσότερες τράπουλες. Στην αρχή του παιχνιδιού ας ρωτάει ο Η/Υ τον παίκτη με πόσες τράπουλες θέλει να παίξει, κι ανάλογα με την απάντηση του παίκτη να προχωράει το παιχνίδι. Και πάλι, αυτή η αλλαγή μπορεί να γίνει με πολλούς τρόπους. Εδώ θα δείξουμε πώς γίνεται, προσθέτοντας πολύ λίγες γραμμές κώδικα, εκμεταλλευόμενοι μια ιδιότητα των κλάσεων που λέγεται **κληρονομικότητα**. Ανοίξτε το αρχείο playing_cards.py και προσθέστε στο τέλος του τον παρακάτω νέο ορισμό κλάσης:

```
class pack(deck):
    def __init__(self,number_of_decks=2):
        d=deck()
        self.content=d.content*number_of_decks
```

Αυτή η νέα κλάση μας δίνει τη δυνατότητα να δημιουργούμε "πακέτα" από τράπουλες. Πριν εξηγήσουμε πώς δουλεύει ας δούμε μερικά παραδείγματα. Σώστε και τρέξτε το αρχείο ώστε να μάθει η Python τη νέα κλάση και γράψτε:

```
>>> p=pack(3)
>>> print p
As 2s 3s 4s 5s 6s 7s 8s 9s Ts Js Qs Ks
Ah 2h 3h 4h 5h 6h 7h 8h 9h Th Jh Qh Kh
```

```
Ac 2c 3c 4c 5c 6c 7c 8c 9c Tc Jc Qc Kc
Ad 2d 3d 4d 5d 6d 7d 8d 9d Td Jd Qd Kd
As 2s 3s 4s 5s 6s 7s 8s 9s Ts Js Qs Ks
Ah 2h 3h 4h 5h 6h 7h 8h 9h Th Jh Qh Kh
Ac 2c 3c 4c 5c 6c 7c 8c 9c Tc Jc Qc Kc
Ad 2d 3d 4d 5d 6d 7d 8d 9d Td Jd Qd Kd
As 2s 3s 4s 5s 6s 7s 8s 9s Ts Js Qs Ks
Ah 2h 3h 4h 5h 6h 7h 8h 9h Th Jh Qh Kh
Ac 2c 3c 4c 5c 6c 7c 8c 9c Tc Jc Qc Kc
Ad 2d 3d 4d 5d 6d 7d 8d 9d Td Jd Qd Kd
156-0
```

Το p είναι ένα αντικείμενο τύπου pack το οποίο αποτελείται από 3 τράπουλες, επειδή γράψαμε pack(3). Πράγματι, βλέπουμε στην εκτύπωση ότι κάθε φύλλο εμφανίζεται 3 φορές και έχουμε σύνολο 156 (=3x52) φύλλα.

```
>>> p.shuffle()
>>> print p
2c 5s 3c Qd 8h Jd Js 8d Ac 7s 9c 5s 9d
Ad 5h 4c 6d 9h Jc 5h 4d 8c Qs 6d 4h Qh
4h Ks As Td 5c Jh 7s 6h 2s 3s Kh Kh 3d
Qc 8d 9c 9d Kh 9s 3d 9h 6h 7s Kc 5h Ah
Tc 6c 8c 9h Tc Kd Kc 2d 7c Qc 4c 2d 9d
2h Jh 5c 3h 4h Ac Ah 2s Ks Th Th 7d 7d
Kd 6s Kc As Jc Qs 6s 6d 2d 4s 7c 5d 2h
5d 6h 3c Tc 2s 8d 9c 5d 2c Js 6c Qd 5c
4s Kd 3h Jh Td 2c 3s 8h 3d 7h 3h Qs Ts
4s 4c Ad 6c Ts 5s Th 7h 8s 7d 4d Ah Js
Qh Qc 9s Ad Jd 3c 7h Td Ks 8c 8s Qd 3s
9s 7c 8h 2h 4d As Jc 6s Ac Ts Qh 8s Jd
156-0
```

Όμως κάτι παράξενο γίνεται εδώ. Εμείς όταν ορίσαμε την κλάση pack γράψαμε μόνο την __init__. Δεν προσθέσαμε μέθοδο shuffle(), ούτε καν μέθοδο __str__. Πώς μπορεί και ανακατεύει το πακέτο; Πώς γίνεται και το τυπώνει;

Η απάντηση βρίσκεται στο "class pack(deck):" με το οποίο αρχίσαμε τον ορισμό. Προσέξτε πως όλες οι προηγούμενες κλάσεις που ορίσαμε δεν έγραφαν κάτι μέσα στην παρένθεση. Εδώ όμως γράψαμε το όνομα της κλάσης deck. Με αυτόν τον τρόπο δηλώνουμε στην Python ότι η κλάση pack είναι ειδική περίπτωση της κλάσης deck. Και για αυτό το λόγο **κληρονομεί** από την deck όλες τις μεθόδους και όλα τα δεδομένα που έχουμε παραλείψει να γράψουμε (ακριβώς επειδή είναι ίδια). Αρκεί να γράψουμε μόνο εκείνα στα οποία διαφέρει (εδώ μόνο την __init__) κι έτσι απλοποιούμε πολύ τον κώδικά μας. Με άλλα λόγια κάθε αντικείμενο τύπου pack θα έχει τα string values και symbols, τις λίστες content και pile και τις μεθόδους __str__, shuffle(), draw() και collect() ακριβώς όπως κάθε αντικείμενο τύπου deck. Θα διαφέρει μόνο στην __init__, η οποία θα είναι διαφορετική.

Ας δούμε πόσο: μέσα στην παρένθεση της __init__ μετά από το self, υπάρχει το όρισμα number_of_decks=2 που καθορίζει από πόσες τράπουλες θα αποτελείται το πακέτο. Αυτό το "=2" θα μπορούσαμε να το παραλείψουμε. Έχει την έννοια της προεπιλεγμένης τιμής (**default value**). Δηλαδή τώρα που το βάλαμε είτε γράψουμε e=pack(2) είτε σκέτο e=pack() θα μας φτιάξει ένα πακέτο με 2 τράπουλες.

Στη συνέχεια, η __init__ δημιουργεί μία τράπουλα d, ώστε να φτιαχτεί σωστά η λίστα d.content και στη συνέχεια την πολλαπλασιάζει number_of_decks φορές ονομάζοντας το αποτέλεσμα self.content.

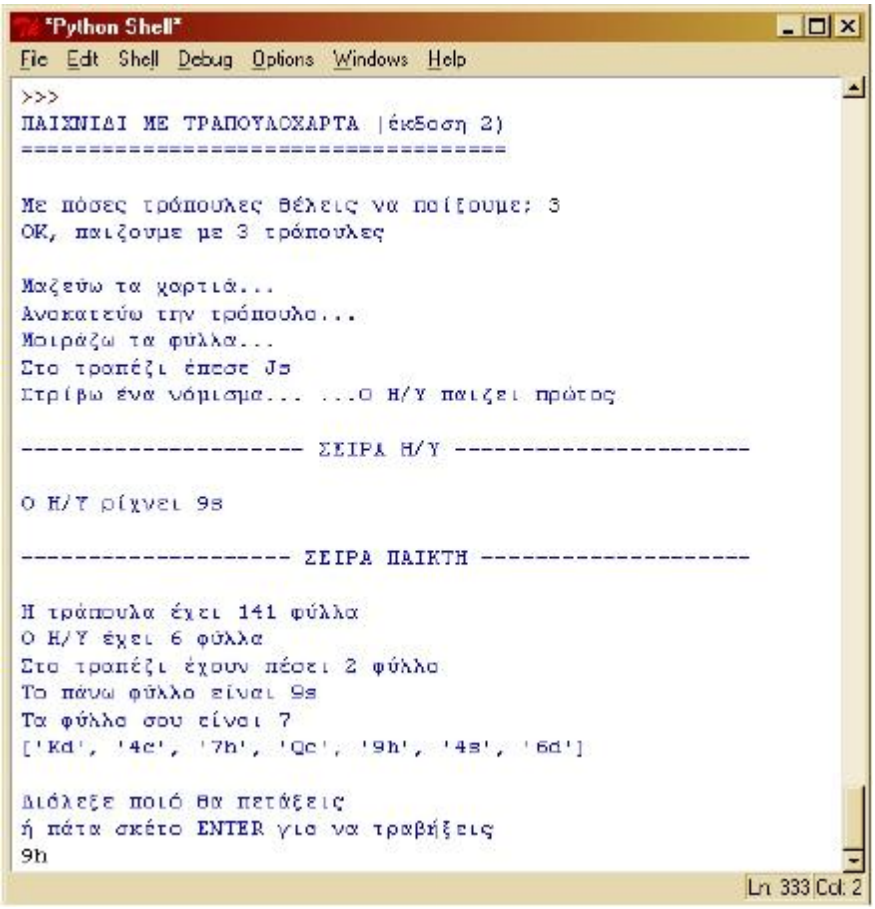
Δηλαδή η λίστα content του pack θα αποτελείται από πολλά αντίγραφα της λίστας content μιας απλής τράπουλας.

Κι αυτό είναι όλο. Δείτε τώρα τί θα αλλάζαμε στο αρχείο card_game.py:

```
...
# ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ
print "ΠΑΙΧΝΙΔΙ ΜΕ ΤΡΑΠΟΥΛΟΧΑΡΤΑ (έκδοση 2)"
print "====================================="
print
n=input("Με πόσες τράπουλες θέλεις να παίξουμε; ")
print "ΟΚ, παίζουμε με",n,"τράπουλες"
print

# ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ
d=pack(n)
...
```

Ουσιαστικά προσθέσαμε μόνο την input() και αλλάξαμε την καθολική μεταβλητή d από deck() σε pack(n). Όλος ο προηγούμενος κώδικας κι όλος ο επόμενος στο αρχείο μένουν ίδιοι. Βέβαια, αν θέλαμε να ήμασταν πιο συνεπείς, θα έπρεπε να κάνουμε κι έναν έλεγχο μήπως ο παίκτης δώσει ακατάλληλη (πχ. αρνητική) τιμή για το n πριν το δεχτούμε. Όμως αυτό δεν είναι επί του παρόντος. Το σημαντικό ήταν να δείξουμε πόσο χρήσιμη είναι η ιδιότητα της κληρονομικότητας μεταξύ των κλάσεων και πόσο μπορεί να απλοποιήσει τον κώδικά μας.

Δείτε παρακάτω μια οθόνη από το βελτιωμένο πρόγραμμα. Προσέξτε στο σημείο που λέει ότι η τράπουλα έχει 141 φύλλα. 141+1+7+7=156=3x52, σωστά.



8.6 Ασκήσεις

Αφού ετοιμάσετε τα δυο αρχεία playing_cards.py και card_game.py σύμφωνα με τις οδηγίες που προηγήθηκαν, εκπονήστε τις παρακάτω ασκήσεις. Αναρτήστε τα αρχεία σας ως ένα ενιαίο συμπιεσμένο αρχείο.

Άσκηση #1

Κάντε πιο ενδιαφέρον το παιχνίδι προσθέτοντας τους εξής επιπλέον κανόνες:

1. Αν ένας παίκτης έχει Άσσο (A), μπορεί να τον ρίξει ό,τι φύλλο κι αν υπάρχει στο τραπέζι. Φυσικά ο αντίπαλος θα πρέπει να συνεχίσει με το σύμβολο του Άσσου (ή με νέο Άσσο).
2. Όταν ένας παίκτης ρίξει Ρήγα (K) ξαναπαίζει.
3. Όταν ένας παίκτης ρίξει Ντάμα (Q) αναγκάζει τον αντίπαλο να τραβήξει 2 φύλλα από την τράπουλα.

Άσκηση #2

Βελτιώστε τον κώδικα του παιχνιδιού έτσι ώστε ο παίκτης να μπορεί να παίζει με πολλούς αντιπάλους (που θα τους ελέγχει ο υπολογιστής). Στην αρχή ο παίκτης θα επιλέγει το πλήθος τους. (Υπόδειξη: Δημιουργήστε μια κλάση computer_player και προσθέστε της ως μέθοδο την συνάρτηση computer_plays(). Τροποποιήστε την όσο χρειαστεί. Με την ευκαιρία, δοκιμάστε αν μπορείτε να κάνετε το Η/Υ να παίζει με πιο έξυπνη στρατηγική.)

Σημειώματα

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

- Έκδοση 1.0 διαθέσιμη εδώ.

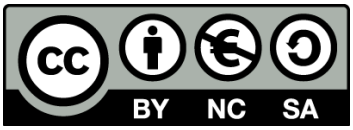
Σημείωμα Αναφοράς

Copyright Πανεπιστήμιο Πατρών, Αβούρης Νικόλαος, Παλιουράς Βασίλειος, Κουκιάς Μιχαήλ, Σγάρμπας Κυριάκος. «Εισαγωγή στους Υπολογιστές Ι, Κοινωνική Διάσταση». Έκδοση: 1.0. Πάτρα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση:

https://eclass.upatras.gr/modules/course_metadata/opencourses.php?fc=15

Σημείωμα Αδειοδότησης

Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

- Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Εικόνες/Σχήματα/Διαγράμματα/Φωτογραφίες

Εικόνες: Προέρχονται από Python IDLE.

Πίνακες

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.