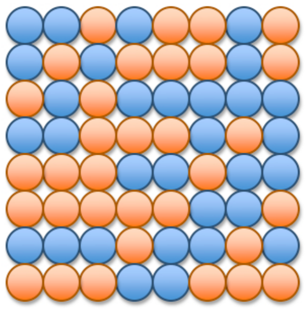

UNIT 9B

Randomness in Computation:
Games with Random Numbers

Simulating a Die

- We want to have a random number between 1 and 6.
- Algorithm:

Algorithm:	Range of Number:
– Generate a pseudo random number using a PRNG with a very large m.	[0, m-1]
– Take the result from the previous step and modulo by 6.	[0, 5]
– Add 1 to the result from the previous step.	[1, 6]

Rolling a die

```
from random import randint
def roll():
    return randint(0,15110) % 6 + 1

OR

def roll():
    return randint(1,6)
```

Simulating a Deck of Cards

- A deck of cards is made up of 52 cards, where each card has a suit and a rank:
  - Suits: Spades (♠), Hearts (♥), Diamonds (♦), Clubs (♣)
  - Ranks: 2, 3, 4, 5, 6, 7, 8, 9, 10, J (Jack), Q (Queen), K (King), A (Ace)

- A standard deck of cards has 1 of each combination of suit and rank.

Cards in PythonLabs

- PythonLabs has an object called a **`Card`** that represents a standard playing card.

```
>>> from PythonLabs.RandomLab import *
>>> c = Card()
>>> c
9♣
>>> c = Card()
>>> c
A♥
>>> c = Card()
>>> c
2♦
```

Use of cards requires us to import RandomLab also.

Cards in PythonLabs (cont'd)

- We can determine the rank or suit of a card:

```
>>> c = Card()
>>> c
7♦
>>> c.rank()
5
>>> c.suit()
1
```

card	suit
♣ (Clubs)	0
♦ (Diamonds)	1
♥ (Hearts)	2
♠ (Spades)	3

card	rank
2	0
3	1
4	2
5	3
6	4
7	5
8	6
9	7
10	8
J	9
Q	10
K	11
A	12

Cards in RubyLabs (cont' d)

- We can get a specific card using an index to the **`Card`** function:

```
>>> c = Card(9)
>>> c
=> J♣
```

Cards are indexed as follows:

2♣	3♣	…	A♣	2♦	3♦	…	A♦	2♥	3♥	…	A♥	2♠	3♠	…	A♠
0	1		12	13	14		25	26	27		38	39	40		51

A Deck of Cards

```
>>> d = []
>>> for i in range(0,52):
...     d.append(Card(i))
...
>>> d
[2♣, 3♣, 4♣, 5♣, 6♣, 7♣, 8♣, 9♣,
10♣, J♣, Q♣, K♣, A♣, 2♦, 3♦, 4♦, 5♦,
6♦, 7♦, 8♦, 9♦, 10♦, J♦, Q♦, K♦, A♦,
2♥, 3♥, 4♥, 5♥, 6♥, 7♥, 8♥, 9♥, 10♥,
J♥, Q♥, K♥, A♥, 2♠, 3♠, 4♠, 5♠, 6♠,
7♠, 8♠, 9♠, 10♠, J♠, Q♠, K♠, A♠]
```

Dealing Random Cards

- Suppose we have a card game like Poker where we want to be dealt a "hand" of 5 random cards from the deck.
- What is potentially wrong with the following code?

```
hand = []
for i in range(0,5):
    hand.append(Card())
```

Shuffling the Deck

- We should shuffle a deck and then create a hand from the first 5 cards in the deck.
- There are many ways to shuffle a deck of cards.
- One algorithm:
  – Exchange (swap) the first card with a random card.
  – Exchange the second card with a random card except the first card.
  – Exchange the third card with a random card except the first two cards.
  – ... Repeat until all cards have been swapped.

Building the Function

- For the first card (at index 0) in deck **`d`**, how do we generate a random index for a card to swap?
```
r = randint(0,len(d)-1)
```
- How do we swap the first card with the randomly-selected card?
```
temp = d[0]
d[0] = d[r]
d[r] = temp
```
or we can use ***parallel assignment*** in Python...
```
d[0], d[r] = d[r], d[0]
```

Building the Function (cont'd)

- For the second card (at index 1) in deck **`d`**, how do we generate a random index for any card except the first card?
```
r = randint(1,len(d)-1)
```
- How do we swap the first card with the randomly-selected card?
```
temp = d[1]
d[1] = d[r]
d[r] = temp
```
or we can use ***parallel assignment*** in Python...
```
d[1], d[r] = d[r], d[1]
```

Building the Function (cont'd)

- For the third card (at index 2) in deck **`d`**, how do we generate a random index for any card except the first two cards?

```
r = randint(2,len(d)-1)
```

- How do we swap the first card with the randomly-selected card?

```
temp = d[2]
d[2] = d[r]
d[r] = temp
```

or we can use ***parallel assignment*** in Python...

```
d[2], d[r] = d[r], d[2]
```

In general...

- For the card at index **`i`** in deck **`d`**, how do we generate a random index for a card to swap?

```
r = randint(i,len(d)-1)
```

- How do we swap the first card with the randomly-selected card?

```
temp = d[i]
d[i] = d[r]
d[r] = temp
```

or we can use ***parallel assignment*** in Python...

```
d[i], d[r] = d[r], d[i]
```

Shuffling the entire deck and dealing five cards...

```
def permute(d):
    for i in range(0,len(d)-1):
        r = randint(i,len(d)-1)
        d[i],d[r] = d[r],d[i]
    return d

>> hand = permute(d)[0:5]
[10♥, 10♠, J♠, 4♣, Q♠]
```

Poker: Detecting a Flush

- In poker, a flush is a hand where all of the cards have the same suit.
- One possible algorithm:
  If all of the cards have a suit of spades, return true.
  If all of the cards have a suit of hearts, return true.
  If all of the cards have a suit of diamonds, return true.
  If all of the cards have a suit of clubs, return true.
  If none of the above tests returns true, return false.

Poker: Detecting a Flush (cont'd)

```
def all_spades(hand)
    for i in range(0,len(hand)):
        if (hand[i].suit() != 3):
            return False
    return True
```

`all_hearts`, **`all_diamonds`** and **`all_clubs`** are written similarly.

Poker: Detecting a Flush (cont'd)

```
def flush(hand)
    if all_spades(hand):
        return True
    if all_hearts(hand):
        return True
    if all_diamonds(hand):
        return True
    if all_clubs(hand):
        return True
    return False
```

Poker: Detecting a Flush (Another way)

```
def flush2(hand)
    for j in range(0,4):   # j is suit index
        count = 0         # reset for suit j
        for i in range(0,len(hand)):
            if (hand[i].suit() == j):
                count = count + 1
        if count == len(hand):
            return True   # all suits were j
    return False
```

Simple dice game

- A player has two die. On each roll, if the player does not roll “doubles” (same value on each die), then the player wins the sum of the die values. Otherwise, the player earns a “strike”. The game ends once the player has three strikes.
- Write a function that returns the amount the player wins in a simulated simple dice game.

Rolling a die

```
def roll():
    return randint(1,6)
```

One round of the game

```
die1 = roll()
die2 = roll()
if die1 == die2:
    strikes = strikes + 1
else:
    sum = sum + die1 + die2
```

Putting it together

```
def simple_game():
    strikes = 0
    sum = 0
    while (strikes < 3):
        die1 = roll()
        die2 = roll()
        if die1 == die2:
            strikes = strikes + 1
        else:
            sum = sum + die1 + die2
    return sum
```

What is the average winnings for 1000 players of this game?

```
>>> games = []
>>> for i in range(0,1000):
...     games.append(simple_game())
...
>>> games
[16, 60, 252, 131, 70,  ... , 209, 70, 107]
>>> total = 0
>>> for score in games:
...     total = total + score
...
>> total/1000
=> 102.674
```