ECS15

for and sequence
Comments

- Another way to repeat.
Flows

Sequential

Block of instructions 1

Block of instructions 2

Block of instructions 3

Conditional

Condition

True

instructions 1

False

instructions 2

Repeating

For each item

Block of instructions
While loop example

```python
count = 1
while count < 11:
    print count
    count = count + 1
```
for loop

for count in range(1,11):
    print count

- Shorter and sweeter, but exactly the same effect
- range(start,stop) is a built-in function
- Produces the sequence [1,2,...,9,10]
Counting forward

for count in range(0, 50, 5):
    print i
Counting backward

```python
for count in range(10, 1, -1):
    print(i)
```
Using a string

word = "my string"

for char in word:
    print char,
Sequences

- A sequence is an ordered collection of objects.
- Printed enclosed in brackets (the brackets are not part of the sequence)
- Examples:
  - \([5,7,1,3]\) – no reason they have to be in numerical order
  - \["r", "p", "s"] – a sequence of one-letter strings
Sequences

- There are different kinds of sequences in Python:
  
  **Strings** – we use these a lot.
  “albatross”

  **Lists** – we saw those last time, they are important. [1,1,2,3,5,8,13]

  **Tuples** – a specialized kind of list: only does some of the things a list can, but does them more efficiently. We will ignore tuples in this course. (1,3,5,7,9)

  **Dictionary**
Strings are sequences

- A string is a sequence of letters
- Well, really, a string is a sequence of **characters**
- Letters, digits, punctuation marks, spaces, are all characters.
- ‘ecs15-f14’ is a sequence of characters.
- ['c', 'o', 'w'] is a sequence of strings, each one character long.
Using sequences

- We can **concatenate** sequences.
  
  “drive” + “way” has value “driveway”
  
  
  \[ [1,2,3] + [3,2,1] = [1,2,3,3,2,1] \]
Looking for an item

- Boolean expressions to test for the presence of an item:

  'i' in 'team'

  7 in range(1,7)
Check if something is in the sequence:

- “r” in [“r”, “p”, “s”] is a Boolean expression, and its value is **True**.
- “x” in [“r”, “p”, “s”] is a Boolean expression, and its value is **False**.
Using sequences

- Use sequences in for loops

  ```python
  for count in range(1,11):
    print count
  
  for animal in ["dog", "cat", "pig", "cow"]: 
    print "I love my",animal
  ```
Checking numerical input

```python
x = input("Enter interest rate: ")
# x is a string
```

- Sometimes we want to convert `x` to a number using `int(x)` or `float(x)`
- If `x` is **not** a string which is a valid integer or floating point number, this causes an error!
Check each digit

- Look at them one by one
- Is each one a digit?

if y in '0123456789':
Nested blocks

goodInput = False # So far, no valid input
while not goodInput:
    x = input("Enter an integer: ")
goodInput = True
for char in x:
    if char not in '0123456789':
        goodInput = False
if not goodInput:
    print "That was not an integer!"
Putting everything together

- Play Black Jack with the computer
- Create a deck (for and list)
- Shuffle a deck
- Deal cards (index)
- Check index (if)
- Learn the value (string and list)
Shuffling!

- Another random function in the random module

  ```python
  random.shuffle(deck)
  ```

- Takes the list deck and shuffles it.
Now to deal a hand…

- `deck[0]` is the first item in the deck
- `deck[i]` is the `i`th item in the deck
- `deck[51]` is the last item (there are 52 cards in a deck)
- `deck[52]` is an **error**!
- `deck[i]`, where `i` is the **index**.
How to avoid the error?

len(deck) = 52

if i >= len(deck):
    print ("Not a good index.")
else:
    print ("The",str(i)+"th card is",deck[i])
Dealing!

- Dealer gets two cards
- Player gets two cards
- How do we decide who wins?
Need to know values

- Make lists of values paralleling list of ranks
- Look up value of rank of each card
Finally, add up score

- My hand and your hand