15-112
Fundamentals of Programming

Week 5 - Lecture 4:
Wrap up

June 16, 2016

Exceptions

Exception

Exception: run-time error

"out of the ordinary" event

"exceptional" event

Handling Exceptions

try/except block

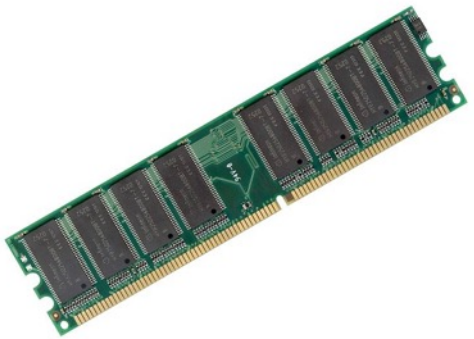
```
try:
    s = input("Enter a number:")
    s = int(s)
    print (1/s)
except:
    print ("Something is wrong…")
```

Reading from a file
Writing to a file

File I/O

- What happens when you run a program?

hard disk



RAM

- Should be able to interact with the files in hard disk
  > Read from a file. Write to a file.

File I/O

```
def readFile(path):
    with open(path, "rt") as f:
        return f.read()

def writeFile(path, contents):
    with open(path, "wt") as f:
        f.write(contents)

contentsToWrite = "This is a test!\nIt is only a test!"
writeFile("foo.txt", contentsToWrite)

contentsRead = readFile("foo.txt")
assert(contentsRead == contentsToWrite)
```

Reading from the web

Web Input

```
import urllib.request

url = "http://www.cs.cmu.edu/"
inurl = urllib.request.urlopen(url)
contents = inurl.read()
inurl.close()

print(contents)
```

List Comprehension

List comprehension

A concise way to create lists.

[<expr> <for clause> (additional/optional for and if clauses)]

```
a = [x  for x in range(10)]
```

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# Same as:
a = []
for x in range(10):
    a.append(x)

# Could of course just do this instead:
a = list(range(10))
```

List comprehension

A concise way to create lists.

[<expr> <for clause> (additional/optional for and if clauses)]

```
squares = []
for x in range(10):
    squares.append(x**2)
```

squares = [0, 1, 4, 9, 25, 36, 49, 64, 81]

```
squares = [x**2  for x in range(10)]
```

squares = [0, 1, 4, 9, 25, 36, 49, 64, 81]

```
primeSquares = [x**2  for x in range(10)  if isPrime(x)]
```

primeSquares = [4, 9, 25, 49]

Functions redux

Functions are first class objects

Functions are first-class citizens:

Can use them like you use any other object.
(in Python, pretty much everything is an object)

- Can pass functions as arguments to other functions

- Functions can be return values for other functions

- Functions can be assigned to other variables,
  or can be stored in data structures (e.g. lists)

Functions are first class objects

```
# Assume selectionSort, bubbleSort, mereSort are defined

def testSort(sortFn, n):
    a = [random.randint(0, 2**31) for i in range(n)]
    start = time.time()
    sortFn(a)
    end = time.time()
    return (end - start)


sortFunctions = [selectionSort, bubbleSort, mergeSort]
n = 2**12
for sortFn in sortFunctions:
    testSort(sortFn, n)
```

Keyword arguments

```
def f(x, y, z):
    print(x, y, z)
```

```
f(1, 2, 3)
```

```
f(1, z=3, y=2)
```

keyword arguments

```
canvas.create_rectangle(0, 0, 50, 50,
                        fill="green", outline="red", width=3)
```

keyword arguments

Variable-length argument list

```
def longestWord(*args):
    if (len(args) == 0): return None
    result = args[0]
    for word in args:
        if (len(word) > len(result)):
            result = word
    return result

print(longestWord("this", "is", "really", "nice"))
```

* "packs" arguments into one tuple

The * makes args = ("this","is","really","nice")

Nested functions

Can be used to avoid "polluting" the global space.

```
def f(a):
    def evens(a):
        return [value for value in a if (value % 2) == 0]
    return list(reversed(evens(a)))


print(f([1,2,3,4,5,6,7]))
print(evens([1,2,3,4,5,6,7]))    # Crashes
```

Nested functions

Can be used to change function signature.

```
def nQueens(n):
    def solve(n, m, constraints):
        ...

    return solve(n, n, [])
```

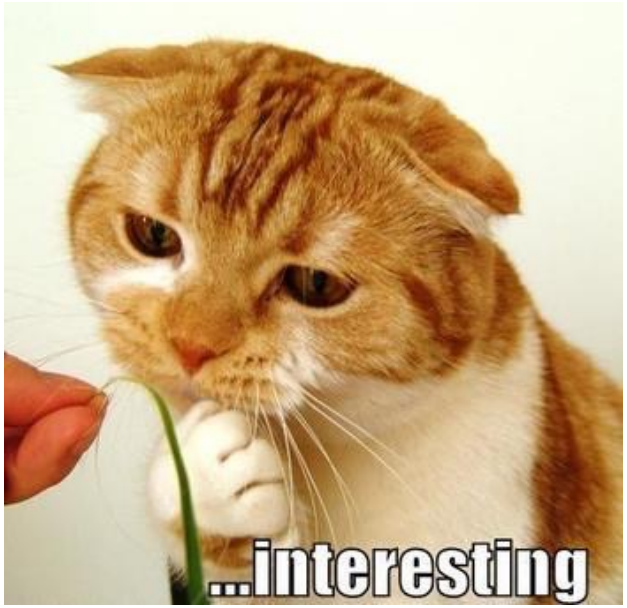
Term Project

What is the TP?

Design and implementation of a program of your choosing.

- graphical, text-based, file-based, …
- interactive, non-interactive
- fireworks, no fireworks

Our general expectations



Some general rules

- SOLO: must do your own independent project.

- COLLABORATIVE: can discuss ideas, designs, algorithms, help each other debug.

- Can use any external materials
e.g. code, designs, images, text, sounds, …

These must be very clearly cited!

This includes citing yourself!

You'll be graded on your original contributions.

Some general rules

- Must use Python

- You will be assigned a "Mentor CA":
  - Provides most of the support and guidance.
  - Will grade your TP.

The overall process

Sun	Mon	Tue	Wed	Thu	Fri	Sat
19	20 Meet	21	22 Meet	23	24 Meet	25 Meet
26 DEADLINE	27	28				

Meeting 1

- **Project proposal**
  - > Define the problem
  - > Description on how you intend to solve it
  - > List all modules/technologies you plan to use

- **Competitive analysis**
  - > Find existing products similar to what you propose
  - > List features you plan to include
  - > List features you plan to change

Meeting 1

- **Storyboard**

  > Hand-drawn pictures showing how app will run from the perspective of the user.

- **Technology demonstrations**

  > Demonstration of competency

- **Code artifacts**

  > If you have any

- **Timesheet**

  > timesheet.txt

  > Keep track of the time you spend on the project.

Meeting 2

- **Progress**

  > A good amount of code

  > Basic features implemented and functional

- **Timesheet**

Meeting 3

- **Working demo**

  > A working B-level final project

  > May miss some features, contain some bugs, etc…

- **Timesheet**

Submission

- **Project source files and support files**
  - > Python files + others (.jpg, midi, …)
  - > 3rd party libraries (if possible)

- **Readme file (readme.txt)**
  - > What is your project?
  - > How to install and run it
  - > How to download/install 3rd party libraries

Submission

- **Design documents**

  > Explain the problem, and how you solve it.

  > Why you chose the particular functions, data structures, algorithms that you used.

  > Discuss the user interface choices.

- **Project video**

  > 1-3 minutes long

  > Show the most important features, highlights

- **Timesheet**

Submission

Submission will be made to Autolab.

Single zip file.

Cannot exceed 10MB.

- Submit complete version to your mentor.
- You can run complete version in grading session.

Grading

Important Factors

- Complexity and sophistication
- Robust operational program
- User interface
- Effort
- Design
- Style
- Presentation

A+
A
A-
B+
B
B-
C+
C
C-
D+
D
D-
R

HAVE FUN!