

## Comments on the homework

---

- Python won't accept `range(0.0,0.5,0.1)`
- Arguments to `range` must be integers
- Why? Cumulative rounding may lead to odd results
- You can use a while loop
- Alternatively:

```
for i in range(0,10) :  
    index = i * 0.1
```

## Comments on the homework

---

- Many people let their function return an error message
- This leads to special-case code in the main program:
  - Detect that an error message was returned
  - Remove it from the table of regular results
- It is easy to break this code, for example by changing the error message
- Today's lecture will show a better way to handle this

# Exceptions

---

- What is an exception?
- How to throw an exception
- How to catch an exception

# Exception

---

- An exception signals an error or unusual condition
- It immediately stops execution of the function that throws it
- The try statement allows the calling code to “catch” the exception
- An uncaught exception stops the program with an error message

## Exception example

---

```
import math
answer = math.log(0)
print answer
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
OverflowError: math range error
```

The log function threw an exception which was not caught. Note that the line "print answer" did not execute.

## Exception example

---

```
import math
try:
    answer = math.log(0)
    print answer
except:
    print "Log function caused an error"
```

This code prints "Log function caused an error" and does not display the OverflowError we saw earlier.

# How to throw an exception

---

```
BadLog = "oops"
def mylog(number) :
    import math
    if (number > 0) :
        return math.log(number)
    else :
        raise BadLog

try:
    answer = mylog(0)
    print answer
except BadLog:
    print "Log function caused an error"
```

# The raise statement

---

- “raise” throws an exception
- You can raise a variable that contains a string or a class object
- When an exception is thrown, the function that threw it ends immediately

# The `except` statement

---

- `except` catches an exception
- plain `except` can catch anything
- `except variablename` catches only that particular exception
- If an exception is not caught, it terminates the program
- If it is caught, the program continues from the place where it was caught
- There is no way to go back to the function which threw the exception

## **Hazards of plain except**

---

- plain except can catch anything...
- including all sorts of system errors you might not have meant to catch
- It is often better to catch a specific exception and let the others through

## Why use exceptions?

---

- We recently wrote a function (LOD score) which could fail on certain input
- What should it do when it fails?
  - Return a special value
  - Return an error message
  - Print an error message and return nothing
  - Throw an exception

# Why use exceptions?

---

- Return a special value
  - If all return values are possible, what to use for special value?
  - If calling code fails to check for special value, might treat it as a normal answer with bad results
- Return an error message
  - If calling code fails to check for error message, might treat it as a normal answer with bad results
- Print an error message and return nothing
  - Error message might show up in a bad place, like the middle of a table
  - Calling code might treat `None` as a normal answer with bad results

# Why use exceptions?

---

- Throw an exception
  - Unless it is caught, program ends immediately—hard to ignore
  - Does not return any answer at all, so answer can't be misinterpreted
- Throwing an exception when a function can't perform is a standard Python idiom

# Practice problem 1

---

Here is a LOD function:

```
import math
def lodscore(NR, R, theta) :
    term1 = ((1.0-theta)**NR * theta**R) / 0.5**(NR+R)
    if term1 > 0 :
        return math.log10(term1)
    else :
        print "Illegal input"
        return None
```

# Practice problem 1

---

- Change this function to throw an exception rather than printing an error message
- Call the exception `BadLodScore`
- Write a try block which:
  - Calls this function
  - Catches the `BadLodScore` exception and prints an error message
  - Otherwise, prints the lod score value
- Test the program with  $\theta=0$ ,  $NR=5$ ,  $R=1$
- Test the program with  $\theta=0.2$ ,  $NR=5$ ,  $R=1$

## Practice problem 2

---

- A dictionary throws a `KeyError` when asked for a key it doesn't contain
- Write a program which:
  - Imports the `rna_to_aa` dictionary from file `code.py` (on the web site)
  - Reads a codon from the command line
  - Tries to look it up in the `rna_to_aa` dictionary
  - If it succeeds, prints the amino acid
  - Otherwise, catches the `KeyError` and prints a message informing the user what the illegal codon was

# Practice problem 1 – solution

---

```
import math
def lodscore(NR, R, theta) :
    term1 = ((1.0-theta)**NR * theta**R) / 0.5**(NR+R)
    if term1 > 0 :
        return math.log10(term1)
    else :
        raise BadLodScore, "Illegal input"
```

```
import sys
theta = float(sys.argv[1])
NR = float(sys.argv[2])
R = float(sys.argv[3])
```

```
BadLodScore = ""
```

```
try :
    lod = lodscore(NR,R,theta)
```

```
    print "Lod = ",lod
except BadLodScore :
    print "No Lod Score due to illegal value(s)"
```

## Practice problem 2 – solution

---

```
import sys
codon = sys.argv[1]

import code

try :
    aa = code.rna_to_aa[codon]
except KeyError :
    print "You entered the codon",codon,"which doesn't map to any",
    print "amino acid"
else :
    print "codon",codon,"translates to amino acid",aa
```