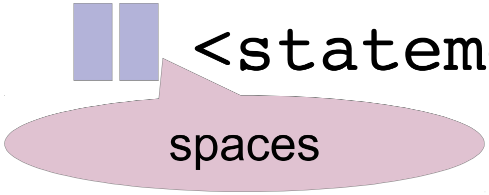
# Intro to Computing
# ES102

Lecture 4

# Creating new functions – syntax

```
def  <functionname>  ( <parameters> ):
    <statements>
```



spaces

- Note that we must indent

- Function block ends in the line when the indentation finishes

# Recursion

- As in almost every programming language, functions can call themselves

- Calculating  factorial :
  - `factorial(n) = n*factorial(n-1)`
  - `factorial(0) = 1`

Base conditions

```
def fact(n):
 if(n==0):
  return 1
 return n*fact(n-1)

n = input(“n=”)
print fact(n)
```

# Recursion

- Recursive functions

- Fibonacci numbers:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

- Write down a recursive function to calculate the n-th Fibonacci number

# Recursion

- Recursive functions

- Fibonacci numbers:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

Base conditions

```
def fib(n):
  if(n==0):
    return 0
  if(n==1):
    return 1
  return fib(n-1) + fib(n-2)

n = input(“n=”)
print fib(n)
```

# String operations

- Given that fruit is a string, what does `fruit[:]` mean?
  - What is `fruit[1:]`
  - What is `fruit[:4]`

- How can we convert the string “Hello world” into the string “Jello world” ?

# String operations

```
def find(word, letter):
    index = 0
    while index < len(word):
        if word[index] == letter:
            return index
        index = index + 1
    return -1
```

- What is the above function doing?

# Other string methods

Suppose `ex =“ This is “` is a string

| | |
|---|---|
| `upper(), lower()` | Converts to upper and lower cases |
| `capitalize()` | Capitalizes the first letters after spaces |
| `strip()` | Strips the spaces in the prefix and suffix. Can also be used to strip other characters |
| `replace('s','S')` | Replaces all occurrences of 's' by 'S' |
| `index('s')` | Returns the first position of 's' |
| `isalpha()` | Returns True if the string contains only alphabetical characters |
| `isnum()` | Returns True if the string contains only digits |
| isupper(), islower() | True or False depending on whether respective conditions are satisfied |

# String methods

- A method is a function, but associated with a particular type, and the syntax for it is a little different

```
>>> word = 'banana'
>>> new_word = word.upper()
>>> print new_word
BANANA
```

- Note that there is no input to the “upper” function
  - Instead note the dot notation
- The string that it is called upon is the implicit input
- This is a feature of object-oriented nature of python

# The “in” operator

- Suppose we want to see whether a substring is present in a string:
  - e.g. does “banana” contain the substring “nana” ?

```
>>> 'nana' in 'banana'
True
>>> 'seed' in 'banana'
False
```

# Looping over a string

- A string is a sequence, and so it is
  - Can obtain its length using len()
  - easy to write a for loop over the elements

```
prefixes = 'JKLMNOPQ'
suffix = 'ack'

for letter in prefixes:
    print letter + suffix
```

# Splitting a string

- `split()` is an useful function in splitting a string into a list of strings
- Without a parameter: takes whitespace as the default parameter

```
>>> a =  “This is a sentence. Another sentence here”
>>> print a.split()
['This', 'is', 'a', 'sentence.', 'Second', 'sentence']
```

- A specific string can be specified as a parameter

```
>>> a =  “set-of-words written-here”
>>> print a.split('-')
['set', 'of', 'words written', 'here']
```

# Exercise

- Write a program `dateConverter.py` that reads a date in the format “`22/09/2015`” and writes it out in the format “`22nd Sept Year 2015`”

- `Hint: Consider using an array of names of months`

# Finding Palindromes

- Palindromes are strings that are the same when read from front or back
  - E.g. “madam” , “civic” , “racecar”,...
- One way to find out a palindrome is to use a simple recursive definition:
  - String S is a palindrome is S[0] equals S[-1] and `S[1:-1]` is also a palindrome
  - Empty string, and strings of length one are palindromes
- Write a recursive function isPalindrome(s) that uses the above definition to find out whether s is a palindrome
  - Remember the base cases
- This is known as divide & conquer technique

# Finding Palindromes

```
def isPalindrome(s):
    if(len(s)<=1):
        return True
    if(s[0] != s[-1]):
        return False
    return isPalindrome(s[1:-1])
```

- Try to extend the above such that whitespaces, punctuations and upper/lower cases are ignored.
- E.g. : “Do geese see god”  is a palindrome. So is “"Madam, I'm Adam"

# Unrolling recursion

- Try using your previous program to find the 50-th Fibonacci number.

- We will see some tricks to avoid recursion:
  - Using loops
  - Using memoization (after mid-sem)

# Using loops

- This is often simple
  - Writing a factorial function that uses loops only: need to keep a variable that accumulates the result

```
def fact(n):
    result = 1
    for i in range(n):
        result = result * i
    return result
```

- What about Fibonacci?

# Using loops for Fibonacci

- Can be done using 2 accumulator variables

Base case

```
def fibo(n):
    if(n<=1):
        return n
    f1, f2 = 0, 1
    for i in range(2, n):
        temp = f1 + f2
        f1 = f2
        f2 = temp
    return f2
```

Be careful about the loop range. Why is this correct?

# List operations

| | |
|---|---|
| Joining two lists | `list1  + list2` |
| Getting a sub-list (called slice) | `list1[a:b]` |
| Adding one element to the end of existing list | `list1.append(elem)` |
| Adding a number of elements to the end of list | `list1.extend(list2)` |
| Sorting a list | `list1.sort()` |
| | |

# List operations: reduce

- Say you want to find the sum of all elements in a list

```
def add_all(mylist):
  total = 0
  for t in mylist:
    total = total + t
  return total
```

The variable total accumulates the sum of elements in the list

- Python also has a inbuilt function called sum

```
def add_all(mylist):
  return sum(mylist)
```

- An operation like this that combines all the values in the list to one value is called a “reduce” type

# Map functions for lists

- Say you have lists of strings, and want to convert all of them into upper case

```
def capitalize_all(t):
 res = []
 for s in t:
  res.append(s.upper())
 return res
```

- Here we are using the list `res` to accumulate the results

# Filter functions for lists

- Suppose instead, we wanted to make a new list out strings which are only upper cased, from the given list

```
def only_upper(t):
  res = []
  for s in t:
    if (s.isupper()):
      res.append(s.upper())
  return res
```

An operation like `only_upper` is called a filter because it selects some of the elements and filters out the others.

# Python's map function

- Since some of these operations are so common, python offers special way to do these easily

```
def myupper(x):
  return x.upper()

t = ['this', 'is', 'list']
newt = map(myupper, t)
```

The input list

The function myupper() gets input one element of the list

- Note that we have to define a function toupper(), since we need to provide one that takes an input, so the upper() method in string cannot be used directly

# Python's reduce function

- There is a similar reduce keyword

```
def mysum(x, y):
  return x + y

T = [1, 2, 3, 4, 5]
newt = reduce(mysum, t)
```

The input list

The function mysum() gets two inputs:
The sum until now, and the next element of the list

- What do you think should the type and value of newt be?

# Python's reduce function

- There is a similar reduce keyword

```
def mysum(x, y):
  return x + y

T = [1, 2, 3, 4, 5]
newt = reduce(mysum, t)
```

The input list

The function mysum() gets two inputs:
The sum until now, and the next element of the list

- newt is a number, not a list, and contains the sum of all entries of t
- Also note that here we need a function with two inputs

# Python's reduce function

- The reduce function is a little subtle

```
def mysum(x, y):
  return x + y

T = [1, 2, 3, 4, 5]
newt = reduce(mysum, t)
```

The variable x is the accumulator – stores sum till now

The next list element is put in the variable y.

- The order in which the above sum is computed is

(((x[0] + x[1]) + x[2]) + x[3])

# Python's filter function

- There is a similar inbuilt filter function

```
def myupper(s):
  return s.isupper()

res = [“THIS”, “is”, “a”, “String”]
res2 = filter(myupper, res)
```

The function that is being provided to `filter()` should return `True` or `False` depending on the condition

# Summarizing

- All the three inbuilt functions map, reduce and filter have similar structure
  - map(function, iterable)
  - reduce(function, iterable)
  - filter(function, iterable)

- But the functions should have different properties in each case:
  - For map, the function needs to take as input exactly one parameter
  - For reduce, the function needs to take as input exactly two parameters
  - For filter, the function needs to take as input exactly one parameter and should return True or False

# Exercise

- Write down a function that
  - Reads in a list of numbers
  - Calculate the mean and standard deviation
  - Use map and reduce as much as possible

# Exercise solution

```
def sq(x):
    return x*x

def mean_sd(numlist):
    n = len(numlist)
    s = sum(numlist)
    mean = float(s)/n
    sqlist = map(sq, numlist)
    sumsq = float(sum(sqlist))/n
    sd = (sumsq - mean*mean)**0.5
    return mean, sd
```

# Exercise

- Use reduce to calculate factorial

# Exercise solution

- Use reduce to calculate factorial

```
def mult(x, y):
    return x*y

def fact(n):
    result = 1
    mylist = range(1, n+1)
    result = reduce(mult,mylist)
    return result
```