Doing the show

Tame the Python.

- Jump into the cage
- Get hands dirty
- Start with examples
- Do some programming

A Sample Program - Factorial

Write iterative and recursive functions to compute the factorial of a given integer. (Assume that the integer is positive).
How will you?

```
1  def recFact(n):
2      if n == 0:
3          return 1
4      return n * recFact(n-1)
5
6  def itrFact(n):
7      ret = 1
8      while n > 0:
9          ret = ret * n
10         n   = n - 1
11     return ret
```

A Sample Program - Factorial

- Simple enough
- Short enough [1]
- Can it be shortened?

[1] more details about them to follow

```
1 def recFactSmall(n):
2     return 1 if n == 0 else n*recFactSmall(n-1)
```

Peep at the hole called big-Oh

- How familiar are you all with algorithm/complexity analysis?
- Do you think size matters?
- What is efficiency? (let's go with layman terms?)
- What is easy? what is hard? in computational terms.

If you hate this stuff, leave it for now... and fall in love with it some time soon. It's always been given the wrong image.

Another Sample Program

Write a program to find the largest number in a list of numbers.
How will you?

```
1  def largest_number(numlist):
2      if len(numlist) < 1:
3          print ("Empty Array")
4          return -999;
5      max = numlist(0)
6      for x in numlist(1:):
7          if max < x:
8              max = x
9      return max
10
11
12 def read_numbers():
13     cardi = int(input("How many numbers: "))
14     listnum = ()
15     if cardi > 0:
16         while cardi != 0:
17             next_num = (int(input("Next number: ")))
```

```
18          listnum.append(next_num)
19          cardi -= 1
20      return listnum
21
22
23  list_of_numbers = read_numbers()
24  if len(list_of_numbers) > 0:
25      print ("\nThe largest number is:", largest_numb
```

Line by line Analysis

- It is not complicated.
- But it is detail oriented.
- Syntax is not important to learn byheart
- Vital thing: Have the concept in mind.
- All the rest comes automatically.

```
1 def largest_number(numlist):
2     if len(numlist) <= 1:
3         print ("Empty Array")
4         return -999;
```

- Function definition with `def` keyword
- Not necessary to have the type of the parameter
- There is a colon `:' at the end of function definition; Also at the end of the `if` statement.
- `len` is the keyword for getting the length of arrays
- Just notice the print statement.

- Semicolon ';' at the end of `return` statement.

```
1  max = numlist(0)
2  for x in numlist(1:):
3      if max < x:
4          max = x
5  return max
```

- Python arrays (lists - (Not the only kind of arrays. Will see more later)) start with index `0'
- Notice the `numlist[1:0]` - this is called slicing. It gives a list with all the elements of the original list starting from index `1' - (Slicing is a very important tool. Will see more later)
- `for` can take each item from the list. (We'll learn about iterators later)

```
1 def read_numbers():
2     cardi = int(input("How many numbers: "))
3     listnum = []
```

- Function without parameters
- Reading input `input` - reads the input as a string.
- Initialising a list with empty list

```
1  if cardi > 0:
2      while cardi != 0:
3          next_num = (int(input("Next number: ")))
4          listnum.append(next_num)
5          cardi -= 1
6  return listnum
```

- How a `while` loop works.
- One of the list operation - `append` - adds the item provided, to the end of the list.
- Remove redundancy?
- `cardi = cardi - 1`
- Instead of `if cardi > 0`, all we need is while `while cardi`

How do we call the functions?

```
1 list_of_numbers = read_numbers()
2 if len(list_of_numbers) > 0:
3     print ("\nThe largest number is:",
4     largest_number( list_of_numbers ))
```

- Just call them from the command line / from outside the function

The Output

```
(sadanand@lxmayr10 % pffp)python largestnum.py
How many numbers: 3
Next number: 1223
Next number: 123333333123
Next number: 12

The largest number is: 123333333123
(sadanand@lxmayr10 % pffp)exit
```

Variables, Values and Types

- Variables are just the positions of what you store in them.
- In the main memory
- Usual naming conventions. `_` or alphabets as beginning, then could be followed by any alphanumeric characters or `_` – `this_is_a_variable`, `____this_too_`, `_t_h_e_3rd_one`

Values and Datatypes

- Values can be different datatypes
- Numbers
  `int, long, float, complex`
- Characters
  A single character, string, unicode, ..
- Collections
  List, Dict, Set, ...
- Other objects we could make
  Tree, Graph, ...,

A break from 'data flooding'

How to write and run a program?

1. Open an editor - your favorite one
   Some editors support syntax highlighting for python. (e.g.: Vim, Emacs, IDLE, etc.)
   Some don't: Notepad
2. Type in the program
3. Save it with extension `py` giving `program.py`

Four ways to Run it

1. Run it with `$python program.py` at the prompt
2. Use IDLE to run it (for Windows)
3. Have `#!/usr/bin/python` as the first line of the file; make the file executable and run it from terminal. $./program.py
4. Configure your editor to have a shortcut key to run it straight from the editor.

Some basic Datatypes

- Numbers
- String
- Lists

1 Numbers

```
>>> 2+2
4
>>> (50-5*6)/4
5.0

>>> 7/3
2.3333333333333335
>>> 7/-3
-2.3333333333333335

>>> width = 20
>>> height = 5*9
>>> width * height
900

>>> x = y = z = 0
```

```
>>> x
0
>>> y
0
>>> z
0

>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5

>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)

```

```
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)

>>> (1+2j)/(1+1j)
(1.5+0.5j)

>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5
>>>
```

2 Strings

```
>>> 'spam eggs'
```

```
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"

>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'

>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
```

```
'<HelpAHelpAHelpAHelpAHelpA>'

>>> word(4)
'A'
>>> word(0:2)
'He'
>>> word(2:4)
'lp'
>>> word(:2)
'He'
>>> word(2:)
'lpA'

>>> word(0) = 'x'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
```

```
36 >>>
```

3 Lists

```
1 >>> a = ('spam', 'eggs', 100, 1234)
2 >>> a
3 ('spam', 'eggs', 100, 1234)
4 >>> a[0]
5 'spam'
6 >>> a[3]
7 1234
8
9 >>> a[-2]
10 100
11 >>> a[1:-1]
12 ('eggs', 100)
13 >>> a[:2] + ('bacon', 2*2)
14 ('spam', 'eggs', 'bacon', 4)
```

```
15
16 >>> 2*a[:3] + ('Boo!')
17 ('spam', 'eggs', 100, 'spam', 'eggs', 100, 'B
18 >>> a
19 ('spam', 'eggs', 100, 1234)
20 >>> a[2] = a[2] + 23
21 >>> a
22 ('spam', 'eggs', 123, 1234)
23
24
25 Replace some items:
26 ... a[0:2] = [1, 12]
27 >>> a
28 [1, 12, 123, 1234]
29
30
31 Remove some:
```

```
... a(0:2) = ()
>>> a
(123, 1234)


Insert some:
... a(1:1) = ('bletch', 'xyzzy')
>>> a
(123, 'bletch', 'xyzzy', 1234)


Clear the list: replace all items with an em
... a(:) = ()
>>> a
()
>>> q = (2, 3)
>>> p = (1, q, 4)
```

```
>>> len(p)
3
>>> p[1]
(2, 3)
>>> p[1][0]
2

>>> p[1].append('xtra')
>>> p
(1, (2, 3, 'xtra'), 4)
>>> q
(2, 3, 'xtra')
>>>
```

Basic Operators

- Numerical Operators
  `+, -, *, /, %`
- Logical Operators
  `True, False, and, not, or`
- Bitwise Operators
  `&, |, ^`
- Shift Operators
  `<<, >>`

Some Little Points

- `chr(i)`
  Return a string of one character whose ASCII code is the integer *i*. For example, chr(97) returns the string `a'
- `ord(c)`
  Given a string of length one, return an integer representing character. For example, ord('a') returns the integer 97 [2]

[2] Unicode

Three Different Fibonaccis

Write three different python functions, each of which gives the fibonacci number corresponding to the input number.
Bonus: Write a 4^{th} and better function.

Find out the square root

Write a program to find out the square root of a given number. (Without the help of python math library)
Bonus: Extend this to n^{th} root.

Power of Two

Write two functions (iterative and recursive) to computer 2^n. Compare the complexities. (only in your mind)

atoi and itoa

Write a program, without using the `int` functionality of python, to convert a string (representing an integer) to the integer.
Also, do the reverse: Integer to String
Try to do both of them recursively and iteratively.

Combinations of Characters

Write a program to generate all the combinations of all the characters in a given string, or a list of characters.
Bonus: Beauty of the program.

Control Structures

- Branching on different conditions
  `if`
- Looping
  `while, for`

indexing Selecting a single item from a sequence based on its relative posi
sequence.

infinite loop A loop that does not terminate. See Loop, infinite.

inheritance Defining a new class as a specialization of another class.

loop A control construct for executing

loop index A variable that is used to control a loop. In the statemen
`range(n)`, `i` is being used as a loop index.

loop, infinite See Infinite Loop.

machine code A program in machine language.

machine language The low-level (binary) instructions that a given CPU

main memory The place where all data and program instructions that t
Also known as random access memory

IF

```
if expr:
___ statement1
else:
___ statement2
```

Statement1 is executed if `expr` is true.

What is not TRUE?

False, 0, empty string, empty set, empty list - all are non-true.

All the rest are TRUE.

```
>>> x = int(input("Please enter an int: "))
Please enter an int: 42
>>> if x < 0:
...     x = 0
...     print ('Negative changed to zero')
... else:
...     print ('Non-negative')
...
```

Switch Case

- No switch case in Python
- Implemented with `if... elif... elif... else`
  (see example)

```
>>> x = int(raw_input("Please enter an int:"))
Please enter an int: 42
>>> if x < 0:
...     x = 0
...     print ('Negative changed to zero')
... elif x == 0:
...     print ('Zero')
... elif x == 1:
...     print ('Single')
... else:
...     print ('More')
...
```

Where is the else?

```
if expr:
    if expr2:
        statement
    else:
        statement2
```

```
if expr:
    if expr2:
        statement
else:
    statement2
```

FOR

- Not necessarily Arithmetic progression
- Could iterate on lists, strings etc.

Do NOT modify the list while looping/iterating on a list.
Then how to do?
Make a Slice `for item in mylist[:]`

```
>>>
... a = ('cat', 'window', 'defenestrate')
>>> for x in a:
...     print (x, len(x))
...
cat 3
window 6
defenestrate 12
>>>
>>>
>>>
>>> for x in a[:]:  make a slice copy of list
...     if len(x) > 6: a.insert(0, x)
...
>>> a
('defenestrate', 'cat', 'window', 'defenestrate')
```

While Loops

- The expression is evaluated
- The loop executes as long as it is true.
- Gets out when it becomes false.

```
while expr:
    statement
```

WHILE ELSE - Unique to Python?

Break and Continue

Just like in any other programming language

The `break` statement, like in C, breaks out of the smallest enclosing `for` or `while` loop.

The `continue` statement, also borrowed from C, continues with the next iteration of the loop.

```
1  n = 2
2  while n < 100:
3          m = 2
4          while m < n:
5                  if n % m == 0:
6                          break
7                  m += 1
8          else:
9                  print (n, 'is a Prime Number')
10         n += 1
```