Intro to Computing
ES112

Lecture 3

Some drawings and material from CS101 course pages and lectures of various universities, esp. IITB and OCW@MIT

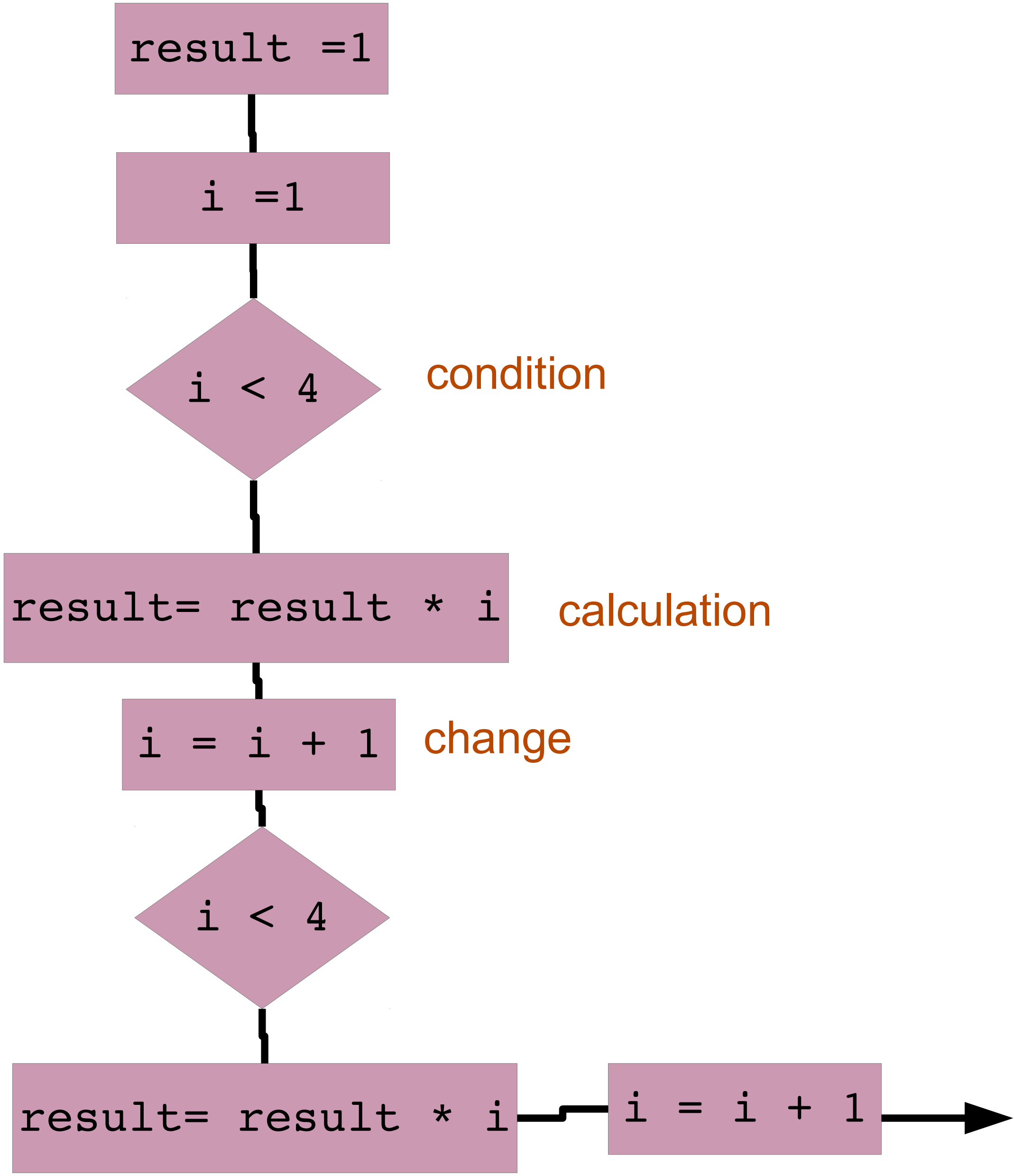
Iteration: while loop

```
result = 1
i = 1
while i < 4:
    result = result * i
    i = i + 1
print “result =”, result
```

- While loop has three parts:
  - the condition: which tells us when the loop should end
  - the actual calculation
  - somewhere where we are actual changing the variable

Iteration: while loop

```
result = 1
i = 1
while i < 4:
    result = result * i
    i = i + 1
print "result =", result
```



result =1

i =1

i < 4

condition

result= result * i

calculation

i = i + 1

change

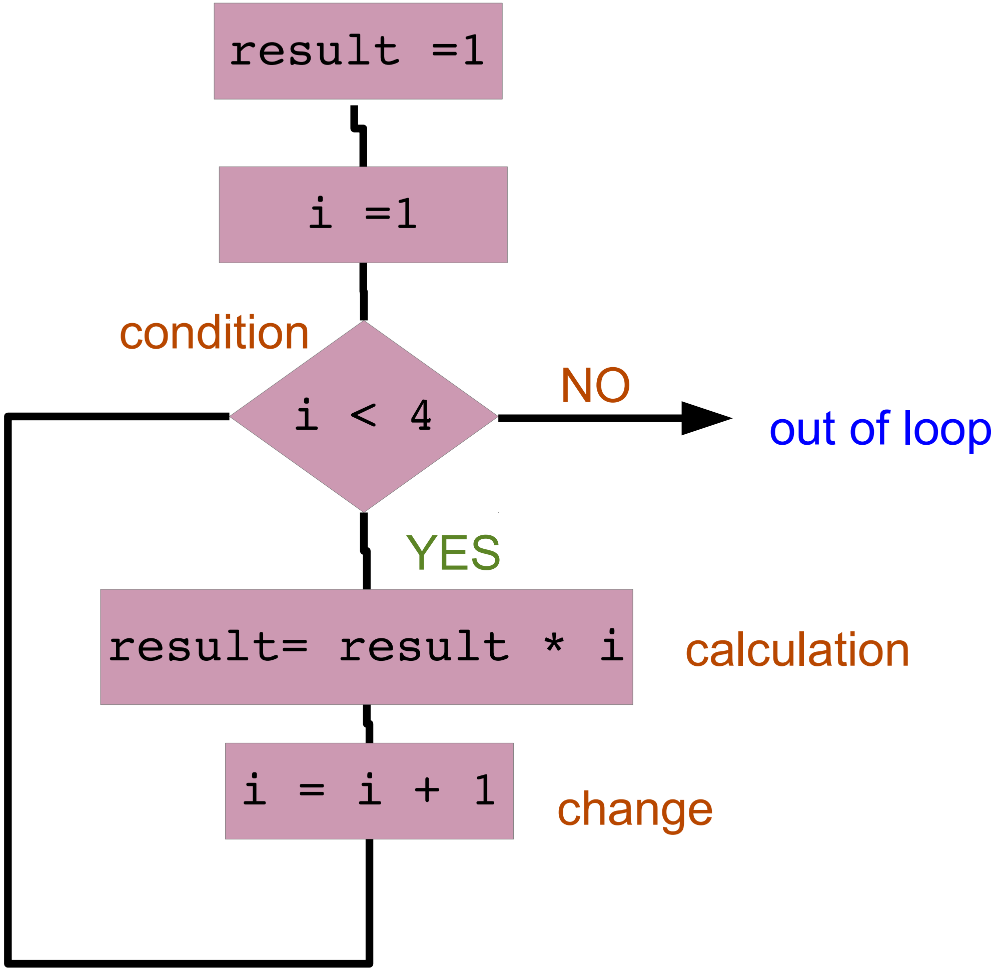
i < 4

result= result * i

i = i + 1

Iteration: while loop

```
result = 1
i = 1
while i < 4:
    result = result * i
    i = i + 1
print “result =”, result
```



Iteration: while loop

```
i = 1
while True:
    print “i = “, i
    i = i + 1
print result
```

- What will happen above?

Iteration: while loop

```
result = 1
i = 1
while i < 4:
    result = result * i
print result
```

- What will happen above?

Iteration: while loop

```
result = 1
i = 1
while i < 4:
    result = result * i
    i = i – 1
print result
```

- What will happen above?

break statement

```
result = 1
i = 10
while True:
    result = result * i
    i = i – 1
    if i < 0:
      break
print result
```

- break is a way to get out of a loop
- What happens above then?

Iteration: `while`

- Let's write a function to find out the cube root of a number:
  - Take `x` as input and pass `x` as a parameter to this function
  - If there exists x, such that `y**3 equals x`, then output `y`
  - Else output `“not a perfect cube”`
- `Hint: use a while loop to look for y`
- Also print out the count of how many `y` values did you try

for loop

- Another way of writing the program we did before

range(4) = [0, 1, 2, 3]

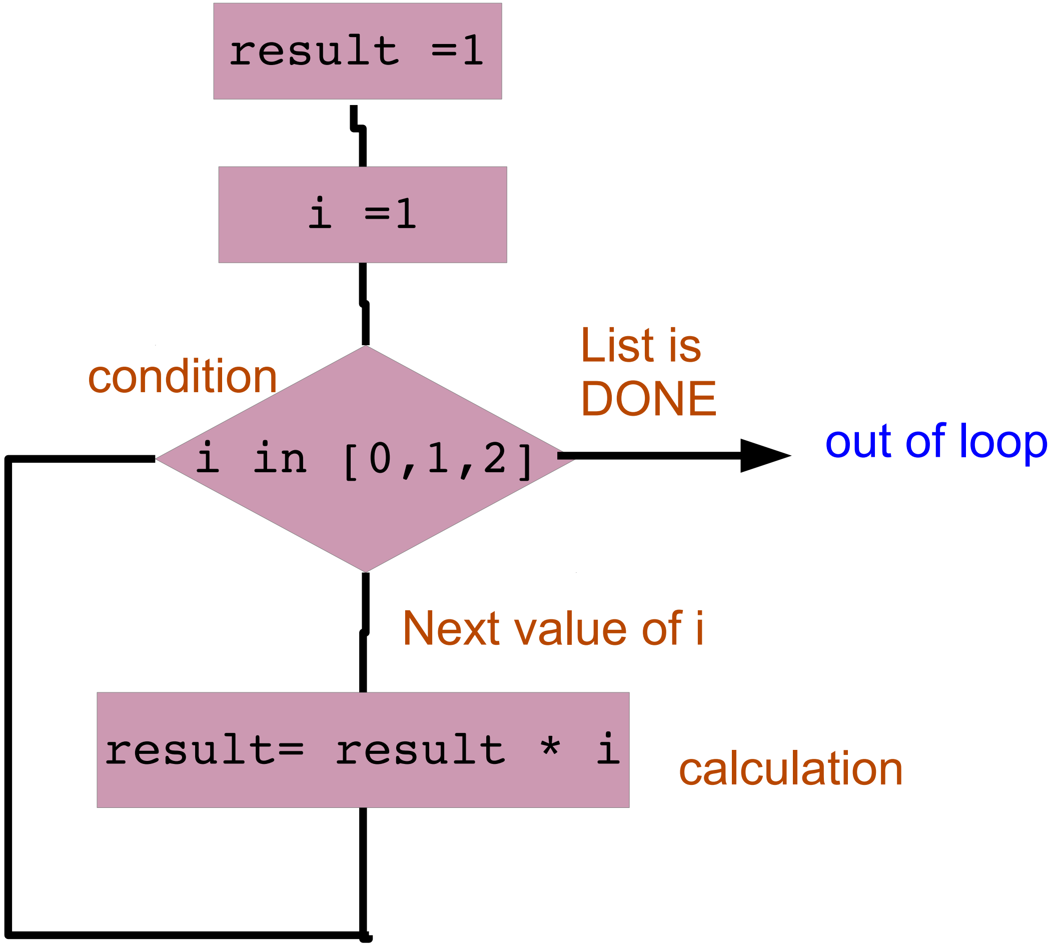
```
result = 1
i = 1
for i in range(4):
    result = result * i
print “result = “, result
```

The loop code is executed for every value of `i` in the list.

So what is `result`?

for loop

```
result = 1
i = 1
for i in range(4):
    result = result * i
print “result = “, result
```



result =1

i =1

condition

i in [0,1,2]

List is DONE

out of loop

Next value of i

result= result * i

calculation

Difference between `for` and `while` loops

```
result = 1
i = 1
for i in range(4):
    result = result * i
print “result = “, result
```

```
result = 1
i = 1
while i < 4:
    result = result * i
    i = i + 1
print “result =”, result
```

- Notice that a `for` loop a programmer does not change the loop variable separately, it automatically takes the next value
- For a while loop, we have to change it

- Which one to use is a matter of taste, either is fine

Rewrite using `for` loop

- Let's write a program to find out the cube root of a number:
  - Take `x` as input
  - If there exists x, such that `y**3 = x`, then output `y`
  - Else output `“not a perfect cube”`

Rewrite using `for` loop

- Let's write a program to find out the cube root of a number:
  - Take `x` as input
  - If there exists x, such that `y**3 = x`, then output `y`
  - Else output `“not a perfect cube”`

- Change your function to find out approximate cube root
  - i.e. find out the integer y such that math.abs(`y**3 – x)` is the smallest among all possible y

Factorial

- Write down a function `factorial(n)` to compute the factorial of n

Approximating sin by a series

- Let's write a program to find out the sin(x) for any x
  - Take as input the number n and the value x
  - Use your function that calculate the factorial of a number n

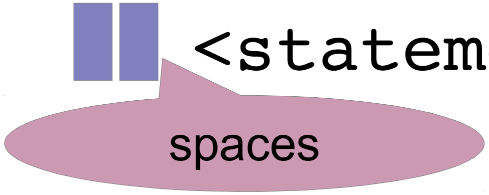
  - Use the above function to calculate

$$\mathrm{mysin}(n, x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + ... + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

  - Print out both the value of math.sin(x) and mysin(x)

Creating new functions – syntax

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



- Note that we must indent

- Function block ends in the line when the indentation finishes

Function Block

```
def print_lyrics():
  print “Papa kehte hain”
  print “Beta engineer banega”
print “Lyrics of an all-time favorite”
print_lyrics()
```

Function block

- The last two statements are **not** part of the function

Indentation

```
def print_lyrics():
  print “Papa kehte hain”
    print “Beta engineer banega”

print “Lyrics of an all-time favorite”
print_lyrics()
```

```
IndentationError: expected an indented block
```

- If you do not keep the indentation consistent, you will get this error

Calling a function

```
def print_lyrics():
  print “Papa kehte hain”
  print “Beta engineer banega”
print_lyrics()
```

- After a function has been defined it has to be called
- The code inside the function block is executed only when it is called

Parameters to the function

```
def print_lyrics(x):
  print “Papa kehte hain”
  print “Beta “, x, “ banega”
print_lyrics(“engineer”)
```

- Notice that there is no “x” outside the function, we only provide a value to the function, not a variable
- Best way to think about this:
  - Each variable represents a container
  - Each time the function is called, new containers are created with the variables names of the parameters

Parameters to functions

```
def print_lyrics(x):
  print “Papa kehte hain”
  print “Beta “, x, “ banega”

print_lyrics(“engineer”)
print x
```

- What do you think happens above? Why?

Local variables are temporary

- Variables defined inside functions are not available outside
  - They are created each time the function is called and destroyed immediately

```
def addone(x):
  x = x + 1
  print “x + 1 = “, x + 1
a = 5
addone(a)
addone(a + 1)
```

What will this output ?

Local variables are temporary

```
def addone(x):
   x = x + 1
   print “x = “, x
a = 5
addone(a)
addone(a)
```

What will this output ?

Scopes

- Local variables inside a function exist in a separate namespace or scope

```
def f(x):
    y = 1
    x = x + y
    print 'x = ',x
    return x
x = 3
y = 2
z = f(x)
print 'z = ',z
print 'x = ',x
print 'y = ',y
```

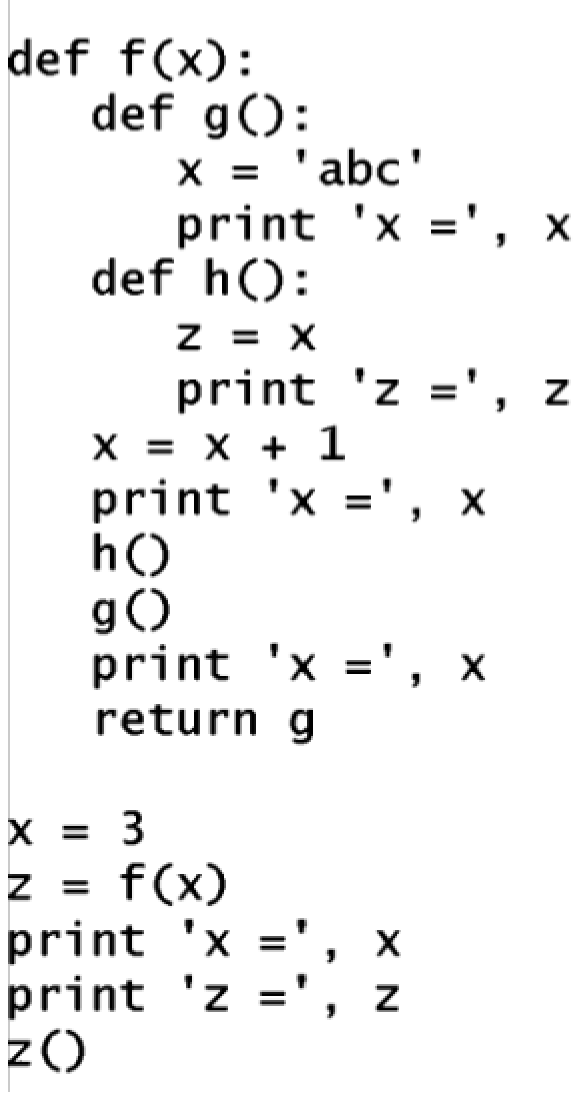
What does the code print?

Scopes in terms of stack frames

- At the top level, there is a symbol table that stores the names and memory bindings of all variables
- When each function is called, a new symbol table (also called stack frame) is created
  - Contains all local variables and the parameters passed
  - This is destroyed when the function exits
  - If another function is called from this function, another stack frame is created

Creating stack frames

- What stack frames are created for the following code?



```
def f(x):
   def g():
      x = 'abc'
      print 'x =', x
   def h():
      z = x
      print 'z =', z
   x = x + 1
   print 'x =', x
   h()
   g()
   print 'x =', x
   return g

x = 3
z = f(x)
print 'x =', x
print 'z =', z
z()
```

More about scopes: locals vs. globals

```
def somefunc(a):
    a = 2
    print "a = ", a + 1
a = 5
somefunc(a)
print a
a = a + 1
somefunc(a)
```

Local variables vs. global

```
def somefunc(a):
    a = 2
    print "a = ", a
a = 5
somefunc(a)
a = a + 1
somefunc(a)
```

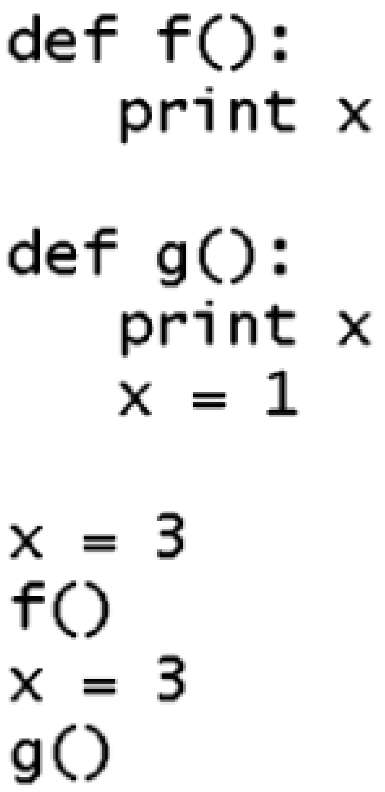
When the function `somefunc` is called, there is already a global variable named `a`

However, if there is a variable of same name, the function always uses that

In the above example, the parameter `a` creates a local variable that is used by the function

More about scopes

- Understanding when variables are createed in symbol tables is often important to avoid confusion



```
def f():
    print x

def g():
    print x
    x = 1

x = 3
f()
x = 3
g()
```

What does this code print?

Accessing global variables

```
def g():
    global x         # accesses global symboltable for X
    print x
    x = 1
x = 3
g()
```

Here, because of the global keyword, the global variable X is being accessed.

Scopes

- Important to remember concepts:
  - Separate symbol table created for each function call (not definition)
  - An entry for a variable name exists only if it appears in the LHS of an expression (i.e. is being assigned)
  - Local symbol table searched first before global symbol table, local definitions override global definitions
  - global keyword forces looking into global symbol table

Return values

- Functions can often return a value

```
def myfunction ( x ):
    a = x + 1
    return a

Val = myfunction(5)
print “Val = “,Val
```

```
def myfunction2 ( x ):
    return x + 1

Val = myfunction(5)
print “Val = “,Val
```

- `return` statement returns the values after it

Return values

- Functions can often return a value

```
def myfunction ( x ):
    a = x + 1
    return a

Val = myfunction(5)
print “Val = “,Val
```

```
def myfunction2 ( x ):
    return x + 1

Val = myfunction(5)
print “Val = “,Val
```

- `return` statement returns the values after it
- If it is a mathematical expression, then the resulting value is returned

Return values

```
def myfunction ( x ):
    return x + 1

Val = myfunction(myfunction(5)) + myfunction(6)
print “Val = “,Val
```

- What is printed as Val?

Exercise

- Write down the following program
  - Input `n` numbers `a1,a2,....,an`
  - Has a function `am(list1)` that calculates the AM of the numbers in the list `list1`
  - Has a function `gm(list1)` that calculates the GM of the numbers in `list1`
  - The program calculates AM and GM of the given numbers and prints them out

Exercise

We will write some to spot-check whether a function is a bijection. Write down a python function to implement the function f : (0,1) → R where

$$f(x) = (x - {}^{1}\!/_{2}) / (x - x^2)$$

- Write down the code for the inverse function g of the above function

- Write down a program that does the following:
  - Takes an input x in (0, 1) from the user
  - Calculate y = f(x)
  - Calculate z = g(y)
  - Checks whether z is equal to x (beware of floating point comparisons), also print the values of x, f(x) and g(f(x))