

## Two more Hmmm examples

## Try It: Write Hmmm for this code

```
x = int(input())
y = int(input())
if x > y:
    x = 3
else:
    x = y + 2
print(x)
```

```
My solution
# jump2.hmmm
0 read r1 # x is in r1
1 read r4 # y is in r4
2 sub r2 r1 r4 # test = x - y
3 jgtzn r2 7 # jump to 'if' branch
4 addn r4 2 # 'else' branch
5 copy r1 r4
6 jumpn 8
7 setn r1 3 # 'if' branch
8 write r1
9 halt
```

## Try It: Write Hmmm for this code

```
def print_even(n):
    for i in range(n+1):
        if i%2 == 0:
            print(i)

n = int(input())
print_even(n)
n = int(input())
print_even(n)
```

```
My solution
# print_even.hmmm
0 read r1
1 calln r14,7 # Call print_even(n)
2 nop
3 read r1
4 calln r14,7 # Call print_even(n)
5 nop
6 halt

# function code print_even
7 setn r13, 2 # set d = 2
8 setn r2, 0 # set i = 0
9 jltzn r1, 16 # If n < 0, jump to end
10 mod r10, r2, r13 # r10 = r1 % 2
11 jnezn r10, 13 # not an even number, skip
12 write r2 # print even number
13 addn r1, -1 # Decrement n by 1
14 addn r2, 1 # Increment i by 1
15 jumpn 9 # Repeat
16 jumpr r14 # Return to the caller
```

## For Loops in Python

## How to compute w/o recursion?

- How to compute the length of a list without recursion?
- How to compute the value of factorial without recursion?
- Many problems we solved using recursion can be solved in loops, or repetitions!

## Loops in Python

Programming languages have mechanisms for explicitly controlling / changing the state of a program:

LOOPS!

General format of a **for** loop  
**for** <variable> **in** <sequence>:  
 <commands in body of loop>

**for** loops:  
*definite, intentional*  
 iteration

```
for x in [1,2,3]:
    print(x)
```

## for loop

1 x is assigned each value from this sequence

```
for x in [2,4,6,8]:
    print('x is',x)
print('Done!')
```

2 the BODY or BLOCK of the for loop runs with that x

3 LOOP back to step 1 for EACH value in the list

4 Code AFTER the loop will not run until the loop is finished.

```
x is 2
x is 4
x is 6
x is 8
Done
```

## Hmmm ideas in Python

### Iteration in Python

```
x = int(input())
result = 1
while x != 0:
    result *= x
    x = x - 1
return result
```

### Iteration in Hmmm

```
00 read r1
01 setn r13 1
02 jeqz r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jump 02
06 write r13
07 halt
```

## Hmmm ideas in Python

### Iteration in Python

```
def fac(x):
    result = 1
    while x != 0:
        result *= x
        x = x - 1
    return result
```

We get the advantages of explicit looping, AND self-contained functions, AND reasonable name while.

## Imperative programming!

- A programming paradigm that describes computation in terms of *statements* that change program *state*.
  - Wikipedia
- Differences from *functional* programming?
  - **Recursion** – a declarative / functional approach to solving problems
  - Treats computation as the evaluation of mathematical functions
    - The notion of "state" is explicitly avoided
    - "state" is implicitly handled in the call stack!

## State?

- A program *state* is defined by the state of its variables.
- Every variable in a program has a *state*
  - state – the current value a variable takes on while a computation is carried out
- Variables change!
  - That's why they are called *variables*!

```
x = 41
x = x + 1
```

the initial value is often not the one we want in the end

But we change it as we go...

## for loops

Explicit sequence

```
for x in [2,4,6,8]:
    print(x)
```

In the interest of efficiency, these are the only **three** forms of for loops!

Sequence in expression

```
for c in [7]*6:
    print(c)
```

How could we get this loop to run 42 times?

```
for n in
    print(n)
```

There are a **range** of answers to this one...

## for loops

```
for x in [2,4,6,8]:
    print(x)
```

```
for c in [7]*6:
    print(c)
```

```
for n in range(42):
    print(n)
```

How could we get this loop to run 42 times?

There are a *range* of answers to this one...

13

## Recursion vs Iteration

```
def min(aList):
    if len(aList) == 1:
        return aList[0]
    elif aList[0] > aList[1]:
        return min(aList[1:])
    else:
        return min(aList[0:1] + aList[2:])

def min(aList):
    minSoFar = aList[0]
    for x in aList[1:]:
        if x < minSoFar:
            minSoFar = x
    return minSoFar
```

We wrote the recursive calls to "carry" the min forward until the base case.

minSoFar is an *accumulator* variable! Very common with loops

14

## Two exercises on for

```
for x in [0,1,2,3,4,5,6,7]:
    print('x is', x)
```

1. Modify the loop list to avoid writing the entire list out
2. Modify the loop to compute the sum of the numbers (HINT: you'll need an accumulator variable)

15

## Write factorial with for

```
def fac(n):
    answer = 1
    for x in range(1, n+1):
        answer = answer * x
    return answer
```

16

## sum of a list?

```
def sum(aList):
    answer = 0
    for x in aList:
        answer = answer + x
    return answer
```

17

## Loops aren't just for lists...

```
for c in 'down with CS!':
    print(c)
```

```
d
o
w
n

w
...
```

18

## Iterating through sequences

- We have mostly been using the `in` keyword with `for` to access one element at a time

```
for x in [2, 22, 222, 2222]:
    print(x)
```

- There is another common approach...

## Two kinds of `for` loops

### Element-based Loops

```
sum = 0
for x in aList:
    sum += x
```

```
aList = [ 42, -5, 10 ]
```

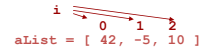
`x`



### Index-based Loops

```
sum = 0
for i in range(len(aList)):
    sum += aList[i]
```

```
aList = [ 42, -5, 10 ]
```



19

20

## Two kinds of `for` loops

### Element-based Loops

```
sum = 0
for x in aList:
    sum += x
```

```
aList = [ 42, -5, 10 ]
```

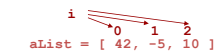
`x`



### Index-based Loops

```
sum = 0
for i in range(len(aList)):
    sum += aList[i]
```

```
aList = [ 42, -5, 10 ]
```



`aList[i]`

21

## `for` loop exercise

Write a function to print a number of "hello" based on the given parameter, using a `for` loop.

`print_hello(3)` → "1 hello 2 hello 3 hello"

```
def print_hello(n):
    str = ""
    for i in range(1, n+1):
        str += str(i) + 'hello '
    return str
```

```
def print_hello(n):
    str = ""
    for i in range(1, n):
        str += str(i) + 'hello '
    str += str(n) + 'hello'
    return str
```