Why Assembly Language?

It’s only the foolish who never climb Mt. Fuji -- or who climb it again.

富士山に一度も登らぬバカ、二度登るバカ
Why Assembly Language?

It’s only the foolish who never climb Mt. Fuji -- or who climb it again.
Why Assembly Language?

It’s only the foolish who never climb Mt. Fuji -- or who climb it again.
How should software be *assembled*?
Python! How does Python function?

Hmmm

Jon Von Neumann, advisor

Don Gillies, assembler

Grace Hopper, admiral + author of the first high-level (human-level) language, COBOL

CS 5 Today

Instructions vs. Functions
Python! How does Python function?

Hmmm

CS 5 Today

Jon Von Neumann, advisor

Don Gillies, assembler

Grace Hopper, admiral + author of the first high-level (human-level) language, COBOL

Instructions vs. Functions

winning side - eventually
Grace Murray Hopper ’28 taught math and physics at Vassar for 12 years before joining the Navy reserves in 1943. During the second world war she programmed the Mark I, the world’s first large-scale computer, which was used to perform the calculations needed to position the Navy’s weaponry: guns, mines, rockets, and, eventually, the atomic bomb.

In 1945, she popularized the term “debugging” after finding a moth stuck in the computer’s machinery. Over the course of her career, Hopper invented the compiler to automate common computer instructions, became the first to start writing computer programs in English, and helped to develop the first “human-friendly” computer language, COBOL.
The first bug?

I'm glad it's not called demothing.
The first bug?

“The OED Supplement records sense (4b) of the noun bug (“a defect or fault in a machine, plan, or the like”) as early as 1889. In that year the Pall Mall Gazette reported (11 Mar: 1) that ‘Mr. Edison ... had been up the two previous nights discovering a ‘bug’ in his phonograph—an expression for solving a difficulty, and implying that some imaginary insect has secreted itself inside and is causing all the trouble.’....

This meaning was common enough by 1934 to be recognized in Webster’s New International Dictionary: ‘bug, n.... 3. A defect in apparatus or its operation... Slang. U.S.’” (citation)
functions vs. instructions

Functions: Python
Instructions: Hmmm
Hmmmmwork #7

**hw7pr1.hmm**  **Recursive power!**  you'll start from the *functional* factorial example we used today

**hw7pr2.py**  **Python loops**  we'll look at these next...
It figures a Python would prefer looping to jumping!
Hmmm-thinking  

Loops in Python

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

Jumps in Hmmm

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

It figures a Python would prefer looping to jumping!
Hmmm-thinking in Python

Loops in Python

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

Jumps in Hmmm

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

It figures a Python would prefer looping to jumping!
Loops in Python

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

Jumps in Hmmm

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

It figures a Python would prefer looping to jumping!
Hmmm-thinkng in Python

Loops in Python

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

Jumps in Hmmm

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

It figures a Python would prefer looping to jumping!
Hmmm-thinking *in Python*

Loops in Python

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

Jumps in Hmmm

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

It figures a Python would prefer looping to jumping!
<table>
<thead>
<tr>
<th>Loops in Python</th>
<th>Jumps in Hmmm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>def fac(x):</strong></td>
<td><strong>00 read r1</strong></td>
</tr>
<tr>
<td>result = 1</td>
<td><strong>01 setn r13 1</strong></td>
</tr>
<tr>
<td>while x != 0:</td>
<td><strong>02 jeqzn r1 6</strong></td>
</tr>
<tr>
<td>result *= x</td>
<td><strong>03 mul r13 r13 r1</strong></td>
</tr>
<tr>
<td>x -= 1</td>
<td><strong>04 addn r1 -1</strong></td>
</tr>
<tr>
<td><strong>return result</strong></td>
<td><strong>05 jumpn 02</strong></td>
</tr>
<tr>
<td></td>
<td><strong>06 write r13</strong></td>
</tr>
<tr>
<td></td>
<td><strong>07 halt</strong></td>
</tr>
</tbody>
</table>

It figures a Python would prefer looping to jumping!
Hmmm-thinking *in Python*

Loops in Python

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

We get the advantages of explicit looping AND self-contained functions

All the advantages of Hmmm? I'm sold!
00 read r1
01 setn r15 42
02 call r14 5
03 jump 21
04 nop
05 jnezn r1 8
06 setn r13 1
07 jumper r14
08 storer r1
09 addn r15 1
10 storer r14
11 addn r15 1
12 addn r1 -1
13 call r14 5
14 addn r15 -1
15 loadr r14 r15
16 addn r15 -1
17 loadr r1 r15
18 mul r13 r13 r1
19 jumper r14
20 nop
21 write r13
22 halt

00 read r1
01 setn r13 1
02 jeqzn r1 6
03 mul r13 r13 r1
04 addn r1 -1
05 jumpn 02
06 write r13
07 halt

Loops!

Recursive Hmmmm factorial, hw6pr4

Looping Hmmmm factorial, similar to hw6pr2 and pr3

Hmmmm... I think I'll take Python!
Iterative design in Python

**for**

```python
for x in [40, 41, 42]:
    print(x)
```

**while**

```python
x = 42
while x > 0:
    print(x)
    x -= 1
```

Variables vary a lot!

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

But we change it as we go...

```
x = 41
x += 1
```
for loops: four examples...

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

for y in [7]*6:
    print(y)

for c in 'down with loops!':
    print(c)

for i in
    print(i)
```

How could we get this loop to run 42 times?

There are a range of answers to this one...
for!

1. x is assigned each value from this sequence

   ```python
   for x in [2, 4, 6, 8]:
   ```

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

   ```python
   print('x is', x)
   ```

3. LOOP back to the top for EACH value in the list

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

   ```python
   print('Done!')
   ```

anatomy?
empty?
x unused?
That's why they're called \textit{variables}

\begin{align*}
\text{age} &= 41 \\
\text{age} &= \text{age} + 1
\end{align*}

The "old" value (41)

The "new" value (42)

\begin{align*}
\text{Echoes from Hmmm:} & \quad 05 \quad \text{addn} \quad r1 \quad 1 \\
\text{age} &= *2 \\
\text{age} &= -= 74 \\
\text{age} &= /= 7
\end{align*}

Only in code can one's newer age be older than one's older age...!
That's why they're called *variables*

```
age = 41
age = age + 1
```

The "old" value (41)  The "new" value (42)

```
age += 1
```

**Python shortcuts**

```
hwToGo = 7
hwToGo = hwToGo - 1
```

```
hwToGo -= 1
```

```
amoebas = 21000000
amoebas = amoebas * 2
```

```
amoebas *= 2
```

```
u235 = 8400000000000000000;
u235 = u235 / 2
```

```
u235 /= 2
```

Only in code can one's newer age be older than one's older age...!
for questions for $\textbf{for}$

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

- avoid writing the whole list?
- find the sum of the list?
- showing partial sums?
- factorial function?
four questions for `for`

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

Shortcuts!
- `tab` & `shift-tab`
- `control-`/
def fac(N):
    result = 1
    for x in list(range(1,N+1)):
        result = result * x
    return result
for loop "laddering"

result = 1

for x in [2, 5, 1, 4]:
    result *= x

print(result)
These seem unexpected, but only at first... !?
Quiz

What does the loop say?

result = 1

for x in [2, 5, 1, 4]:
    result *= x

print(result)

result x

1 2 2 5
10 1 1
10
40 4
Quiz

What does the loop say?

```python
x = 0
for i in list(range(4)):
    x += 10
print(x)
```

It's ok not to use the loop variable!
Quiz

What does the loop say?

L = ['golf', 'fore!', 'club', 'tee']

for i in list(range(len(L))):
    if i % 2 == 1:
        print(L[i])

<table>
<thead>
<tr>
<th>i</th>
<th>i%2</th>
<th>L[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>'golf'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>'fore!'</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>'club'</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>'tee'</td>
</tr>
</tbody>
</table>
Quiz

What does the loop say?

S = 'time to think this over!'

result = ''

for i in list(range(len(S))):
    if S[i-1] == ' ':
        result += S[i]

print(result)
Quiz

What does the loop say?

\[ S = \text{'time to think this over!'} \]

\[ \text{result} = '' \]

\[ \text{for } i \text{ in list(range(len(S)))}: \]
\[ \quad \text{if } S[i-1] == ': \]
\[ \quad \quad \text{result} += S[i] \]

\[ \text{print(result)} \]

Looks like a four-'t' 'to' to me!

Extra! How could you change one character above to yield eoks or another to yield mns or another to yield etns!
These seem unexpected, but only at first... ?!

```
result = 1
for x in [2, 5, 1, 4]:
    result *= x
```

```
x = 0
for i in list(range(4)):
    x += 10
print(x)
```

```
L = ['golf', 'fore!', 'club', 'tee ']
for i in list(range(len(L))):
    if i%2 == 1:
        print(L[i])
```

```
S = 'time to think this over!'
result = ''
for i in list(range(len(S))):
    if S[i-1] == ' ':
        result += S[i]
print(result)
```

Pass those in and up!

What does the loop say?
for: two types

\[ L = [3, 15, 17, 7] \]

for \( x \) in \( L \):
    print(\( x \))

Elements vs Indexes

Indices

element-based loops
for: two types

L = [3, 15, 17, 7]

```
for i in range(len(L)):
    print(L[i])
```

```
for x in L:
    print(x)
```
for: two types

\[ L = [3, 15, 17, 7] \]

for \( i \) in \( \text{range}(\text{len}(L)) \):
    print(\( L[i] \))

for \( x \) in \( L \):
    print(\( x \))

index-based loops

printing is NOT unusually common in loops – but it is good for debugging!

element-based loops
simpler vs. flexibler

\[ L = [3, 15, 17, 7] \]

```
def sum(L):
    total = 0
    for x in L:
        total += x
    return total
```

**element-based loops**

```
def sum(L):
    total = 0
    for i in range(len(L))
        total += ___
    return total
```

**index-based loops**
simpler vs. flexibler

\[ L = [3, 15, 17, 7] \]

\[ \begin{array}{c}
0 \\
1 \\
2 \\
3
\end{array} \]

\[ i \]

**Elements vs Indexes**

**Element-based loops**

```
for x in L:
    total += x
return total
```

**Index-based loops**

```
def sum(L):
    total = 0
    for i in range(len(L)):
        total += L[i]
    return total
```
What does this code do?

```python
print('It keeps on')

while 41+1 == 42:
    print('going and')

print('Phew! I\'m done!')
```

I'm whiling away my time with this one!
**Extreme Looping**

Anatomy of a `while`

```python
while 41+1 == 42:
    print('It keeps on')
    print('going and')

print('Phew! I\'m done!')
```

This won't print until the while loop finishes - in this case, it *never* prints!

I'm whiling away my time with this one!
Extreme Looping

lots of different tests...

```python
print('It keeps on')
while 42 == 42:
    print('going and')
print('Phew! I\'m done!')
```

I'm whiling away my time with this one!
Extremeloopying

lots of different tests...

print('It keeps on')

while True:
    print('going and')

print('Phew! I\'m done!')
import random
escape = 0

while escape != 42:
    print('Help! Let me out!')
    escape = random.choice([41, 42, 43])

    print('At last!')

how could we count the number of loops we run?
how could we accumulate a LIST of all the guesses?

starting value, not the final or desired value!
test to see if we keep looping

starting value, not the final or desired value!
test to see if we keep looping

watch out for infinite loops!

after the loop ends
"Birthday Room Experiment"

rand_date()

bdy_list()

design ideas?

what's easy?
what's tricky?
why?
import random

def rand_date():
    
    """returns a random date (a string) in month/day form (no leap year...)
    """

    DiM = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    month = random.choice(range(1, 12 + 1))
    day = random.choice(range(1, 31 + 1))

    randomdate = str(month) + "/" + str(day)

    return randomdate

What's DiM?

why the +1? Programs ~ executable thought...
import random

def rand_date():
    """ returns a random date (a string) in month/day form (no leap year...) """

    Dim = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 30, 31]

    month = random.choice( range(1, 12+1) )
    day = random.choice( range(1,) )

    randomdate = str(month) + "/" + str(day)

    return randomdate
import random

def rand_date():
    """ returns a random date (a string) in month/day form (no leap year...) """

    DiM = [0,31,28,31,30,31,30,31,31,30,31,30,31]

    month = random.choice( range(1,12+1) )
    day = random.choice( range(1,DiM[month]+1) )

    randomdate = str(month) + "/" + str(day)

    return randomdate
def bday_list():
    """ birthday-paradox example!
    returns the list of bdays up to and including the first repeat...
    """
    LoBs = []  # List of BDays
    
    while all_unique(LoBs) == True:
        bday = random_date()
        LoBs += [bday]

    return LoBs

How long til a repeat?
How many til a repeat?

birthday "paradox"

Name(s): __________________
Bday(s): __________________

```python
rand_date()  # is it today?
bday_list()  # gather til a repeat
len(bday_list())  # how many til a repeat?

LoLs = [ len(bday_list()) for i in range(100) ]

min(LoLs)  # ____ what might this be?
sum(LoLs)/len(LoLs)  # ____ the average?
max(LoLs)  # ____ "ballpark" this...

LoLs = [ len(bday_list()) for i in range(10000) ]

min(LoLs)  # ____ what might this be?
sum(LoLs)/len(LoLs)  # ____ and the average?
max(LoLs)  # ____ "ballpark" this?

L = [ rand_date() for i in range(330) ]  # ____ How many repeats would you "ballpark" to be in L?
How long til a **repeat**?

**Sooner than you might think...**

---

**Understanding the Birthday Paradox**

by Kalid Azad · 186 comments · [Tweet](https://twitter.com) 219

23 **people**. In a room of just 23 people there’s a 50-50 chance of two people having the same birthday. In a room of 75 there’s a 99.9% chance of two people matching.

Put down the calculator and pitchfork, I don’t speak heresy. The birthday paradox is strange, counter-intuitive, and **completely true**. It’s only a “paradox” because our brains can’t handle the compounding power of exponents. We expect probabilities to be linear and only consider the scenarios we’re involved in (both faulty assumptions, by the way).

Try these…

What do these two loops return?

Let WORD = 'forty-two'

```python
def count( WORD ):
    n = 0
    for c in WORD:
        if c not in 'aeiou':
            n += 1
    return n
```

Finish this loop to find and return the \texttt{min} of a list, \( L \)

L will be a non-empty list of numbers.

```python
def min( L ):
    result = L[0]
    for x in L:
        if [ ]:
            result = x
    return result
```

Extra: Write a loop so that this function returns \texttt{True} if the input \( n \) is prime and \texttt{False} otherwise

Let \( n = 12 \)  Let \( n = 8 \)

```python
def mystery( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True
```

def isPrime( n ):

Hint: check all possible divisors to see if they "work"...

Challenge: for what inputs \( n \) does \texttt{mystery} return \texttt{True}?
Let WORD = 'forty-two'

```python
def count( WORD ):
    n = 0
    for c in WORD:
        if c not in 'aeiou':
            n += 1
    return n
```

Let n = 12 (then 8)

```python
def mystery( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True
```

What do these two loops return?

Challenge: what are inputs for which `mystery` returns True?
def min(L):
    result = L[0]
    for x in L:
        if x < result:
            result = x
    return result

An example list to consider...

[50, 55, 43, 99, 45, 42]

def isPrime(n):
    return int(sqrt(n))
    for d in range(2, n):
        if n%d == 0:
            return n/2
    return

Hint: check all possible divisors to see if they "work"...
Loop on! What do these two loops return?

Let WORD = 'forty-two'

def count( WORD ):
    n = 0
    for c in WORD:
        if c not in 'aeiou':
            n += 1
    return n

Finish this loop to find and return the min of a list, L

L will be a non-empty list of numbers.

def min( L ):
    result = L[0]
    for x in L:
        if
            : return result

Extra: Write a loop so that this function returns True if the input n is prime and False otherwise

Let n = 12, 8

def mystery( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True

We'll keep revisiting for and while loops over the upcoming weeks!

12 8

Challenge: for what inputs n does mystery return True?

Hint: check all possible divisors to see if they "work"...
and this was before watches – or glasses...

Loop of life, XKCD's take:

- Stare blankly at screen
- Absently check smaller device
- Open news site
- Get bored
- Start reading

Challenge:

use Hmmm code somewhere in your first **Writ1** or **SpecRel** assignment...

5, 42, 10, 21, 12, 7