Lecture 7: Memory and the Stack

CS 51P

September 25, 2019

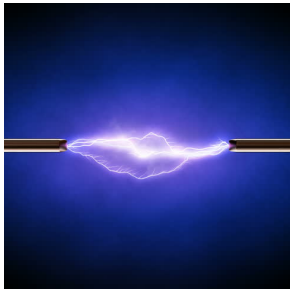
Bits

- a **bit** is a binary digit that can have two possible values

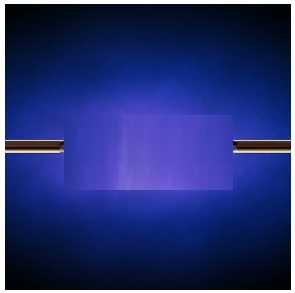
- can be physically represented with a two state device









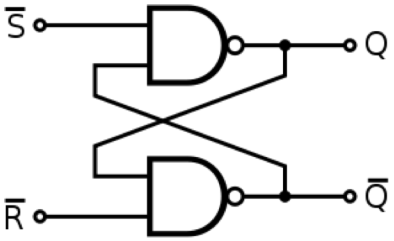



Bits


A'LA'IH, DO'NEH'LINI, DO'NEH'LINI, A'LA'IH, A'LA'IH, DO'NEH'LINI, DO'NEH'LINI, DO'NEH'LINI, A'LA'IH, A'LA'IH, DO'NEH'LINI, A'LA'IH, DO'NEH'LINI, DO'NEH'LINI, DO'NEH'LINI . . .
FOR ADDED SECURITY, AFTER WE ENCRYPT THE DATA STREAM, WE SEND IT THROUGH OUR NAVAJO CODE TALKER.
... IS HE JUST USING NAVAJO WORDS FOR "ZERO" AND "ONE"?
WHOA, HEY, KEEP YOUR VOICE DOWN!

Storing bits

- Static random access memory (SRAM): stores each bit of data in a flip-flop, a circuit with two stable states
- Dynamic Memory (DRAM): stores each bit of data in a capacitor, which stores energy in an electric field (or not)
- Magnetic Disk: regions of the platter are magnetized with either N-S polarity or S-N polarity
- Optical Disk: stores bits as tiny indentations (pits) or not (lands) that reflect light differently
- Flash Disk: electrons are stored in one of two gates separated by oxide layers



Binary Numbers

4211

$$= 4 \cdot 10^3 + 2 \cdot 10^2 + 1 \cdot 10^1 + 1 \cdot 10^0$$
$$= 4211$$

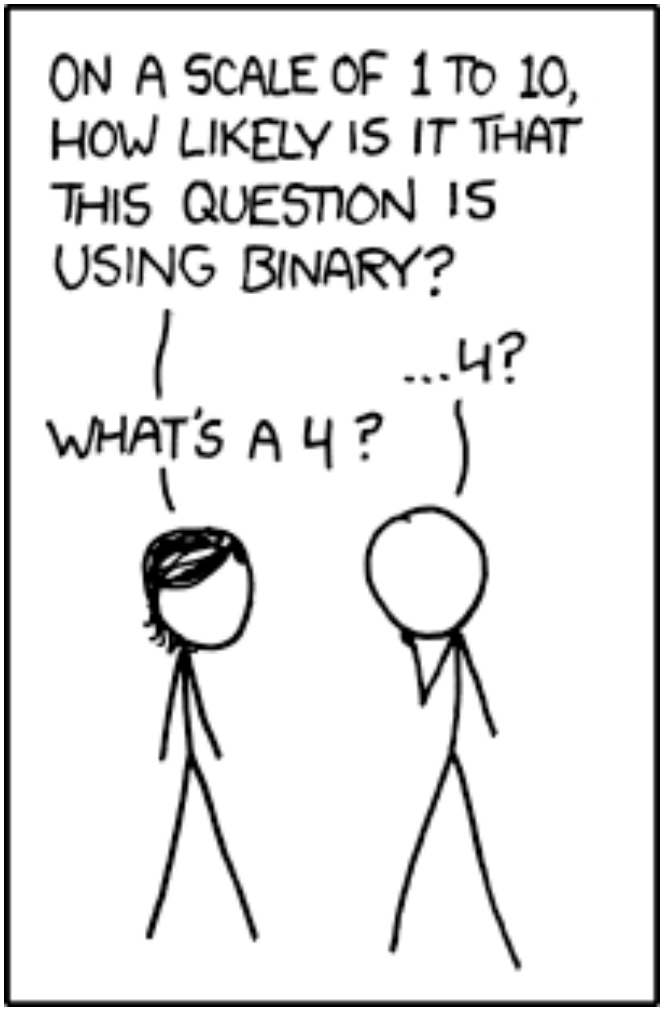
1011

$$= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$= 11$$

Exercise

- What (decimal) numbers are represented by the following (binary) values
  - 101111
  - 110011
  - 11111100011

Binary Numbers



ASCII characters

Char	Dec
(sp)	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40
)	41
*	42
+	43
,	44
-	45
.	46
/	47
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
:	58
;	59
<	60
=	61
>	62
?	63

Char	Dec
@	64
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
[	91
\	92
]	93
^	94
_	95

Char	Dec
`	96
a	97
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
j	106
k	107
l	108
m	109
n	110
o	111
p	112
q	113
r	114
s	115
t	116
u	117
v	118
w	119
x	120
y	121
z	122
{	123
|	124
}	125
~	126

Program Instructions

Python Code

```
def example1(n):
  x = n + 1
  return x
```

Binary Representation

```
10001101 01000111 00000001
11000011
```

Bits Require Interpretation

01000011 01010011 00110101 00110001

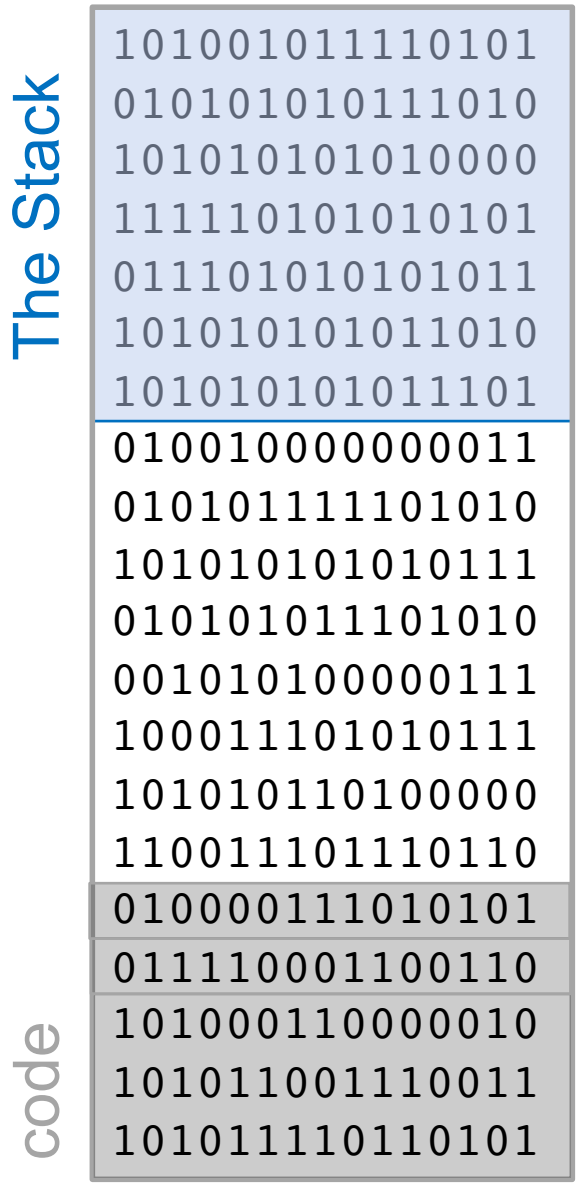
might be interpreted as

- The integer 1129526577_{10}
- A floating point number close to 211.207779
- The string “CS51”
- A portion of an image or video
- A portion of code
- An address in memory

Information is Bits + Context

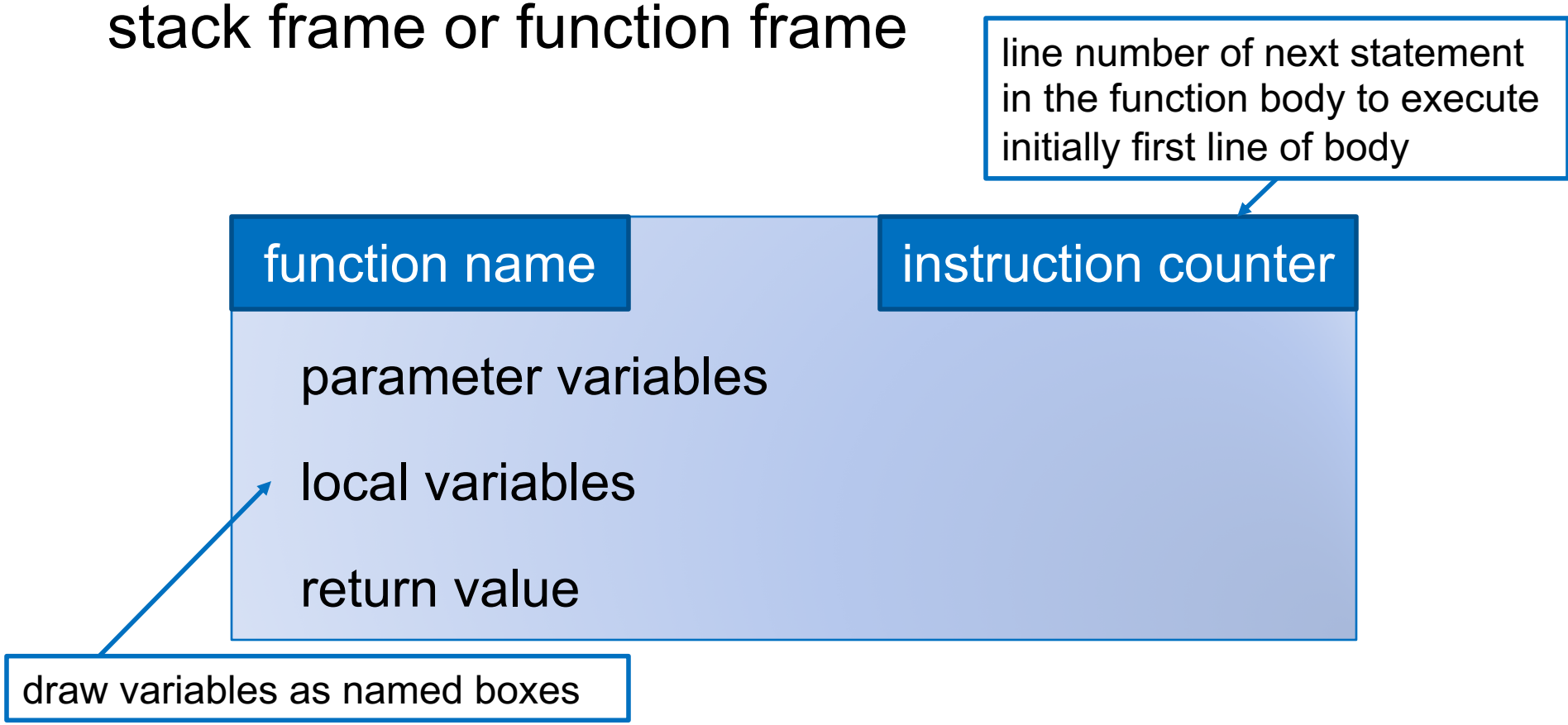
Memory

- memory is a sequence of bytes
- different "sections" of memory are used for different purposes
- code section stores your programs
- the stack is used to store variables to keep track of functions



Stack Frames

- each time a function is called, that function call gets its own section of the stack, known as a stack frame or function frame

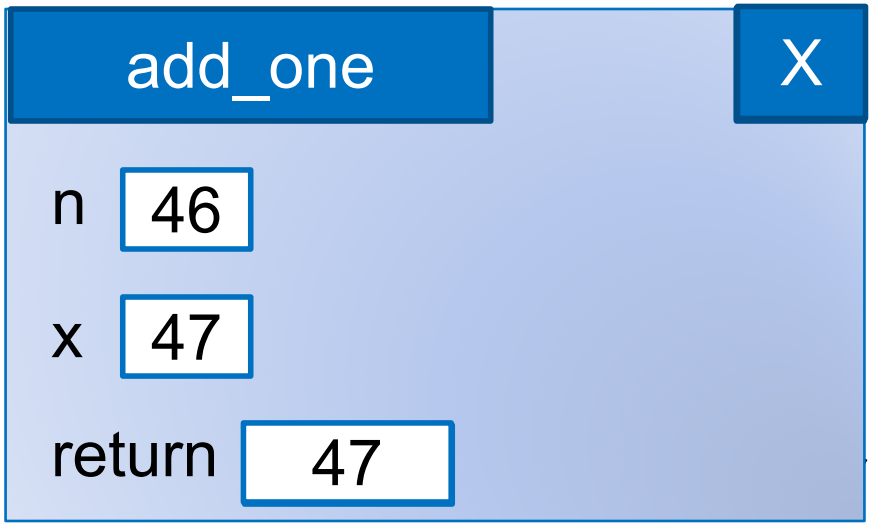


Example

```
def add_one(n):
1   x = n + 1
2   return x
```

num 47

num = add_one(46)



Exercise

```
def foo(a, b):
1   x = a + b
2   y = 2 * b
3   return 2 * x + y
```

foo(2, 3)

Control Flow and Nested Functions

```
def square(n):
1   if n <= 0:
2     return 0
3   else:
4     return n**2

def sum_squares(n):
1   sum = 0
2   for i in range(n):
3     sum += square(i)
4   return sum
```

sum_squares(2)

Exercise

```
def is_pos_int(s):
1   if str.isdigit(s):
2     return int(s) > 0
3   else:
4     return False

def get_pos_int():
1   done = False
2   while not Done:
3     s = input()
4     done = is_pos_int(s)
5   return s
```

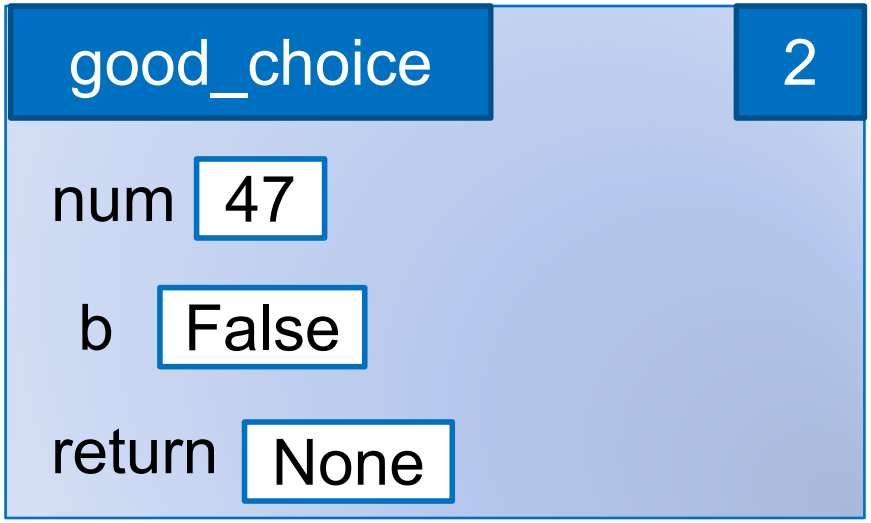
- get_pos_int()
- assume user enters
  - hello
  - 47

Global Variables

global variables are outside of any stack frame. They are in a different section of memory!


```
fav = 13

def good_choice(num):
1    b = (num == fav)
2    return b
```



Scope

```
fav = 13

def good_choice(num):
1    b = (num == fav)
2    return b

def main():
1    in_str = input()
2    fav = int(in_str)
3    if good_choice(fav):
4      print("yay")
5    else:
6      print("boo")
```

- Storing a value in a variable:
  - If there is a variable with that name in the current function's stack frame, store the value in that variable
  - Otherwise create a new variable in the current function's stack frame and store the value there
- Using a variable
  - Check for a local variable with that name. If it exists, use the value stored in that variable
  - Else if there exists a global variable with that name, use the value stored in that global variable
  - Otherwise get a NameError

Exercise

```
def print_example(s4,s5):
    s1 = 3*s4
    s2 = s4+s5
    print(s1)
    print(s2)
    return s1+s2

s1 = '!'
s2 = '?'
print(s1)
s3 = print_example(s1,s2)
print(s2)
print(s3)
print(s4)
```