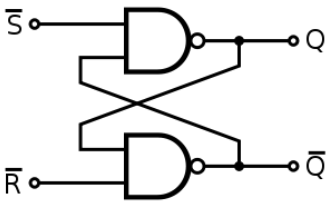
Lecture 7: Memory and the Stack

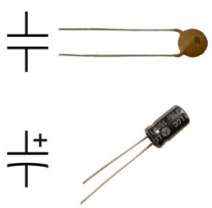
CS 51P

September 26, 2022

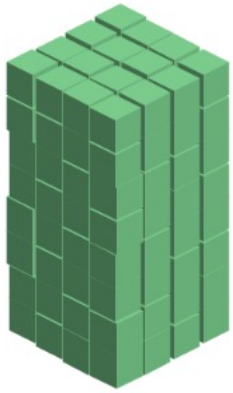
Bits

- a **bit** is a binary digit that can have two possible values
- can be physically represented with a two state device






Binary Numbers (aka Base-2 Integers)

128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)
							
0	0	0	0	0	1	0	1
0	0	1	0	1	1	1	1
1	1	1	1	1	1	1	1

ASCII characters

Ch	Dec	Binary	Ch	Dec	Binary	Ch	Dec	Binary	Ch	Dec	Binary	Ch	Dec	Binary
!	33	00100001	1	49	00110001	A	65	01000001	Q	81	01010001	a	97	01100001
"	34	00100010	2	50	00110010	B	66	01000010	R	82	01010010	b	98	01100010
#	35	00100011	3	51	00110011	C	67	01000011	S	83	01010011	c	99	01100011
$	36	00100100	4	52	00110100	D	68	01000100	T	84	01010100	d	100	01100100
%	37	00100101	5	53	00110101	E	69	01000101	U	85	01010101	e	101	01100101
&	38	00100110	6	54	00110110	F	70	01000110	V	86	01010110	f	102	01100110
'	39	00100111	7	55	00110111	G	71	01000111	W	87	01010111	g	103	01100111
(	40	00101000	8	56	00111000	H	72	01001000	X	88	01011000	h	104	01101000
)	41	00101001	9	57	00111001	I	73	01001001	Y	89	01011001	i	105	01101001
*	42	00101010	:	58	00111010	J	74	01001010	Z	90	01011010	j	106	01101010
+	43	00101011	;	59	00111011	K	75	01001011	[	91	01011011	k	107	01101011
,	44	00101100	<	60	00111100	L	76	01001100	\	92	01011100	l	108	01101100
-	45	00101101	=	61	00111101	M	77	01001101	]	93	01011101	m	109	01101101
.	46	00101110	>	62	00111110	N	78	01001110	^	94	01011110	n	110	01101110
/	47	00101111	?	63	00111111	O	79	01001111	_	95	01011111	o	111	01101111
0	48	00110000	@	64	01000000	P	80	01010000	`	96	01100000	p	112	01110000

Program Instructions

Python Code

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

Binary Representation

```
10001101 01000111 00000001
11000011
```

Bits Require Interpretation

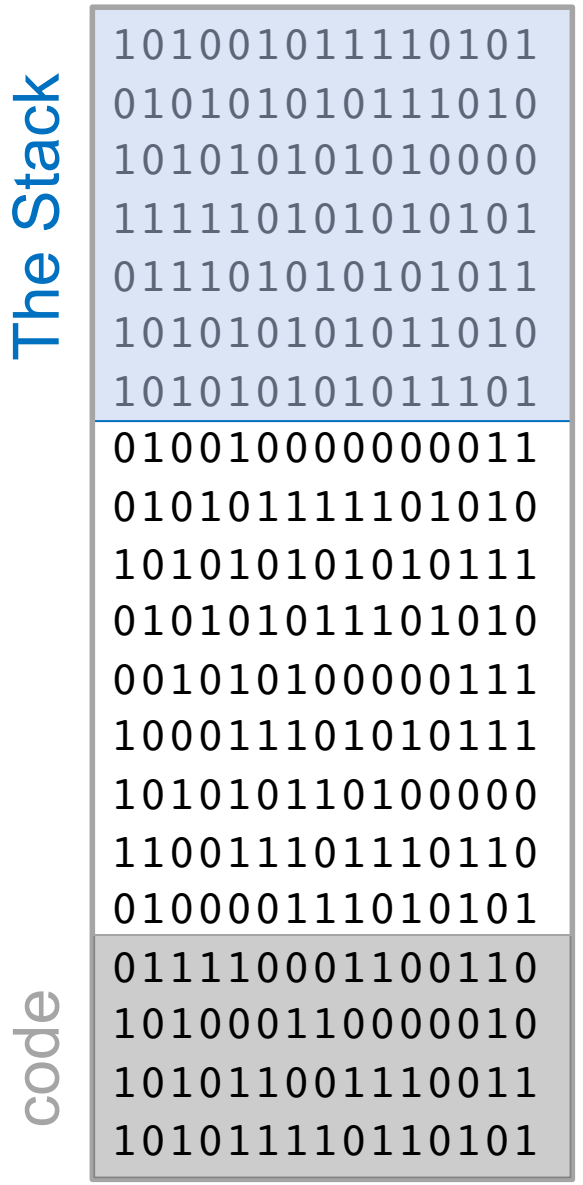
01000011 01010011 00110101 00110001

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

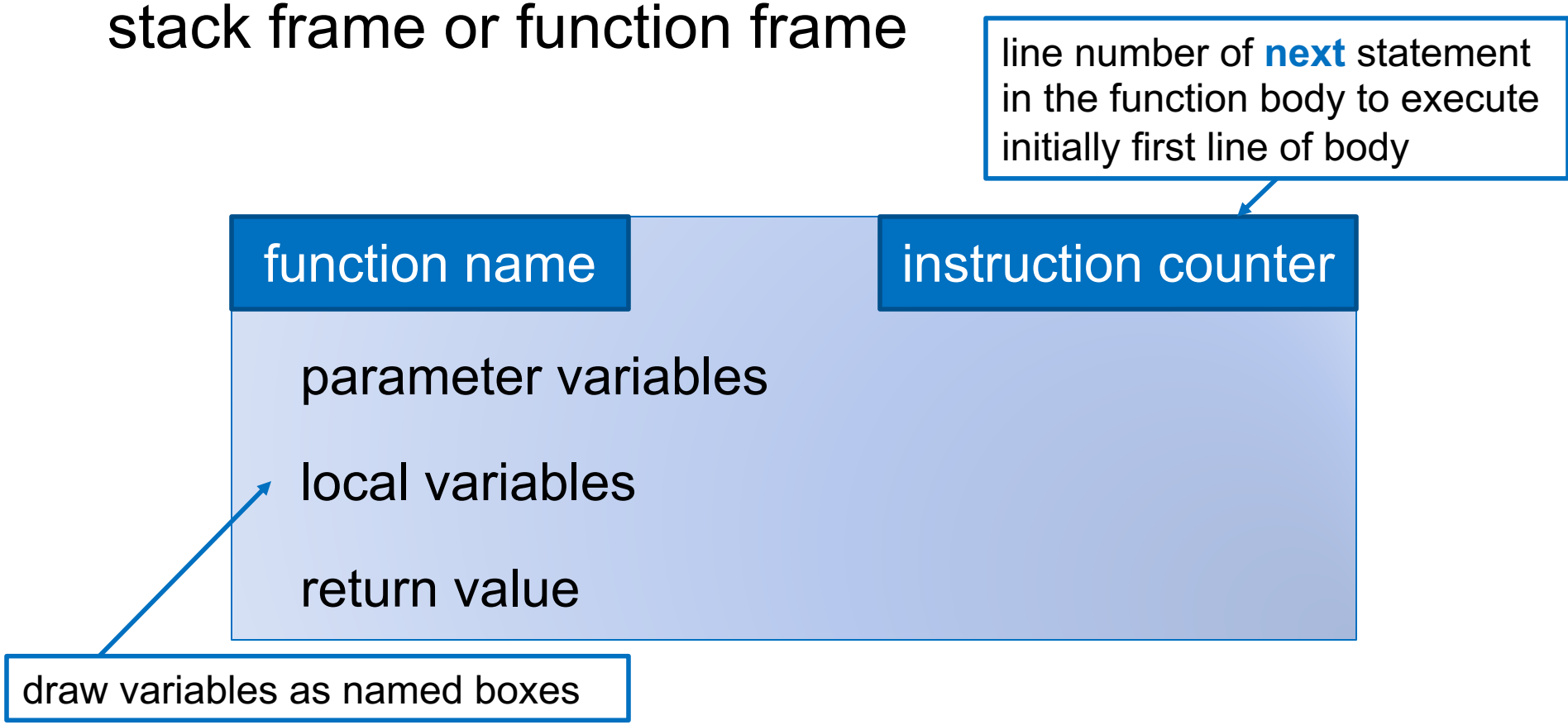
Memory

- memory is a sequence of **bytes** (8-bit segments)
- 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

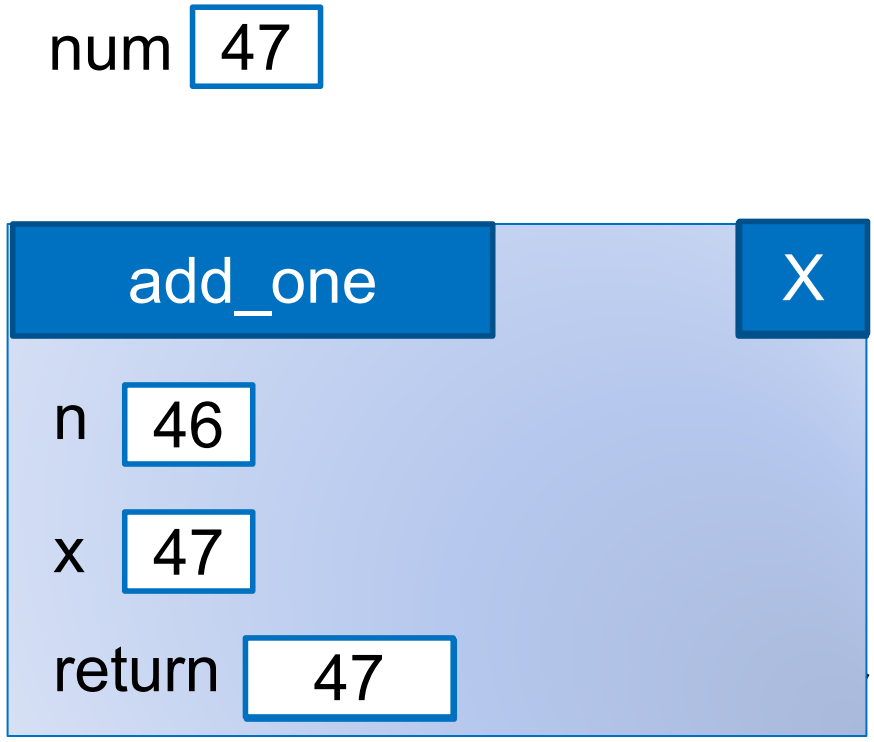
- each time a function is called, that function call gets its own section of the stack, known as a stack frame or function frame



Example

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

num = add_one(46)
```



Exercise

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

n = 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):
5   sum = 0
6   for i in range(n):
7     sum += square(i)
8   return sum

def main():
9   total = sum_squares(2)

main()
```

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():
5   done = False
6   while not done:
7     s = input()
8     done = is_pos_int(s)
9   return s

def main():
10  x = get_pos_int()

main()
```

assume user enters

- "hello"
- "47"

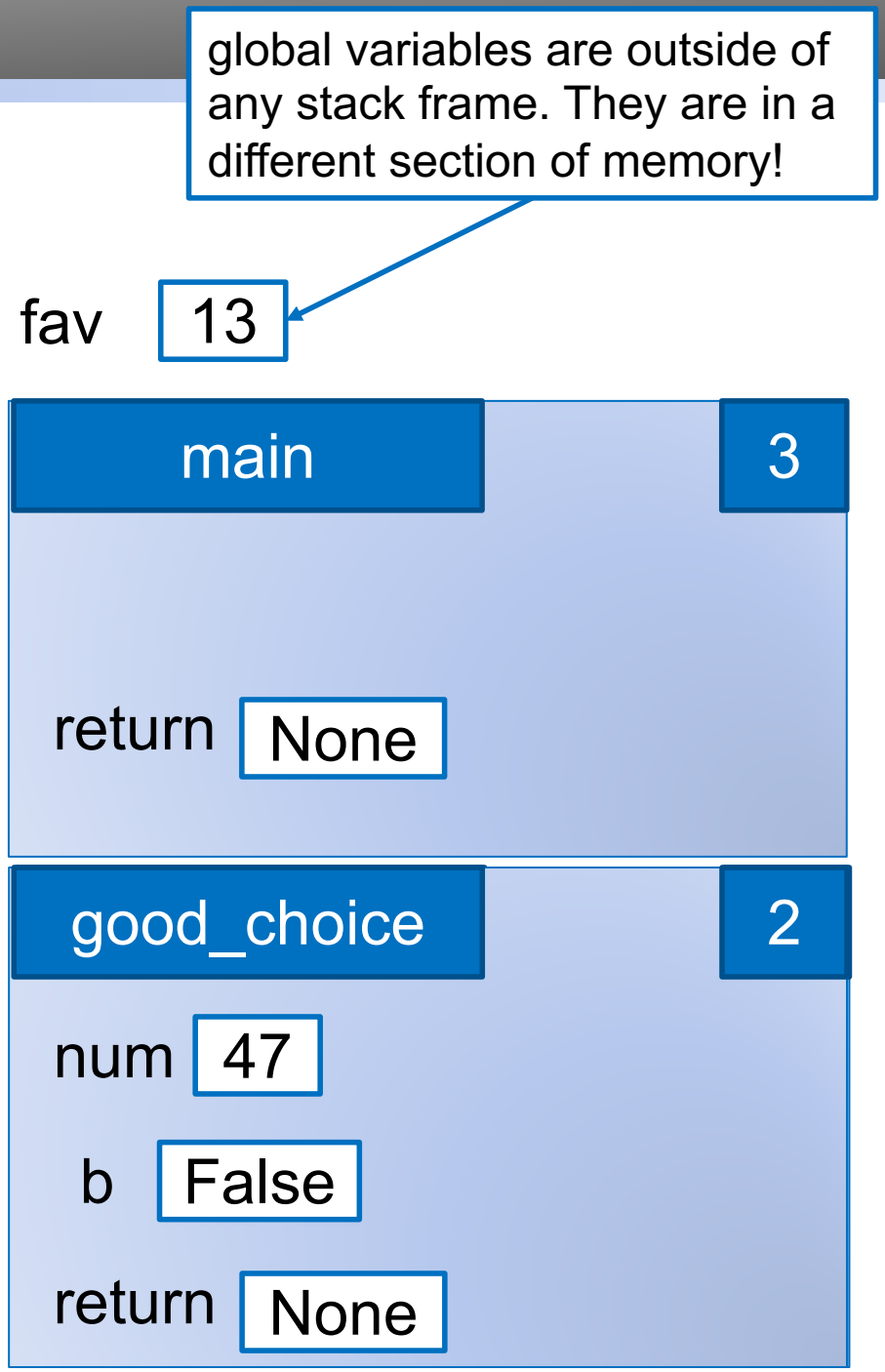
Global Variables

```
fav = 13

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

def main():
3    x = good_choice(47)

main()
```



Scope

```
fav = 13

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

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

main()
```

- 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(s0)
    print(s1)
    print(s2)
    return s1+s2

s0 = '!'
s1 = '?'
print(s0)
s3 = print_example(s0, s1)
print(s1)
print(s3)
print(s4)
```