

Lecture 8: Memory and the Stack

CS 51P

October 2, 2023

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
```

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

The Stack

```
101001011110101
010101010111010
101010101010000
111110101010101
011101010101011
101010101011010
101010101011101
```

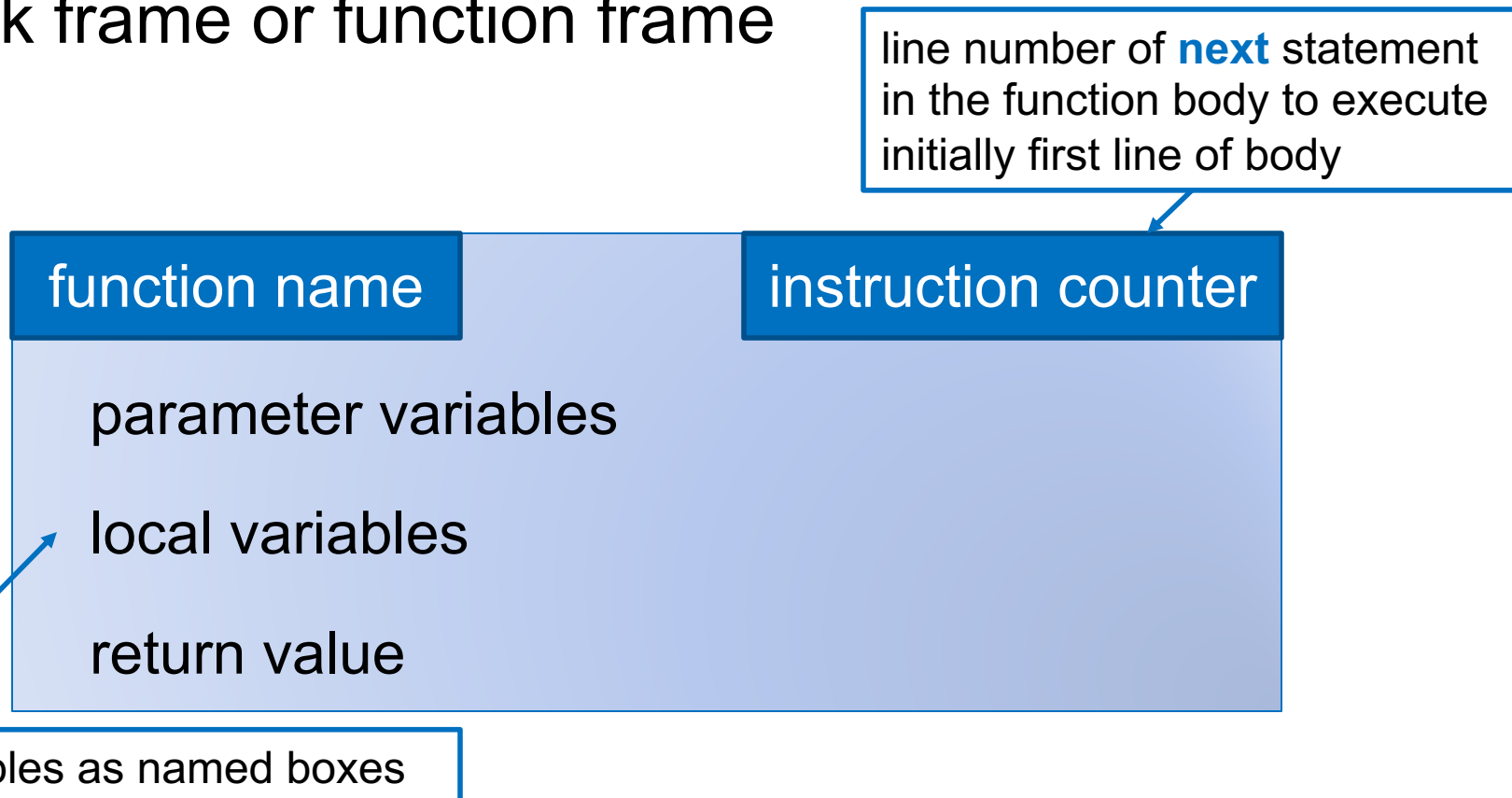
```
010010000000011
010101111101010
101010101010111
010101011101010
001010100000111
100011101010111
101010110100000
110011101110110
010000111010101
```

code

```
011110001100110
101000110000010
101011001110011
101011110110101
```

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

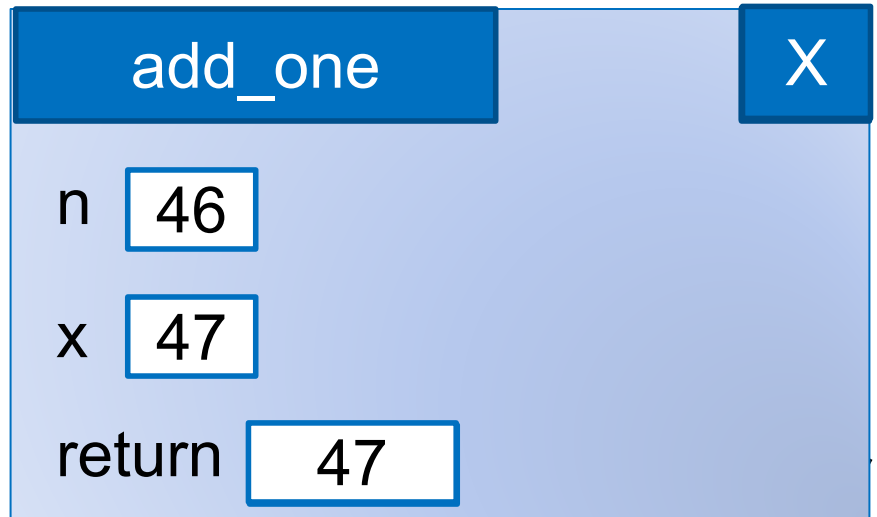
```
def add_one(n):
```

```
1  x = n + 1
```

```
2  return x
```

```
num = add_one(46)
```

num 47



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"
-
- Draw the state of the stack immediately before Python executes line 9