

# CSci 127: Introduction to Computer Science



[hunter.cuny.edu/csci](https://hunter.cuny.edu/csci)

- This lecture will be recorded

# Frequently Asked Questions

From email

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**  
*You don't submit the lab, you read the lab.*



# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

*You don't submit the lab, you **read the lab**.*

*When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)*

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

*You don't submit the lab, you read the lab.*

*When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)*

- **Can I work ahead?**

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

*You don't submit the lab, you read the lab.*

*When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)*

- **Can I work ahead?**

*Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.*

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

*You don't submit the lab, you read the lab.*

*When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)*

- **Can I work ahead?**

*Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.*

IMPORTANT: Students who work on the due dates in this class tend to miss deadlines and fall behind. If, instead, you work on programs the week of the associated lecture, you will have time to ask for help if you get stuck and still make the deadline.

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

*You don't submit the lab, you read the lab.*

*When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)*

- **Can I work ahead?**

*Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.*

IMPORTANT: Students who work on the due dates in this class tend to miss deadlines and fall behind. If, instead, you work on programs the week of the associated lecture, you will have time to ask for help if you get stuck and still make the deadline.

- **When is the midterm?**

# Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**

*You don't submit the lab, you read the lab.*

*When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)*

- **Can I work ahead?**

*Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.*

**IMPORTANT:** Students who work on the due dates in this class tend to miss deadlines and fall behind. If, instead, you work on programs the week of the associated lecture, you will have time to ask for help if you get stuck and still make the deadline.

- **When is the midterm?**

*There is no midterm. Instead there's required weekly quizzes, code reviews and programming assignments.*

# Today's Topics



- **For-loops**
- `range()`
- Variables
- Characters
- Strings
- Guests: Internships, Advising & Clubs

# In Pairs or Triples...

*Some review and some novel challenges:*

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

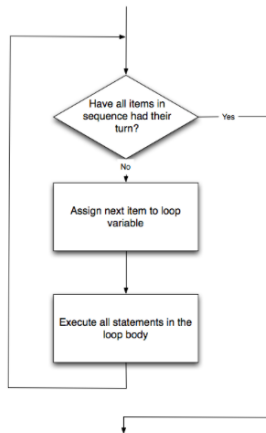


# Python Tutor

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

(Demo with pythonTutor)

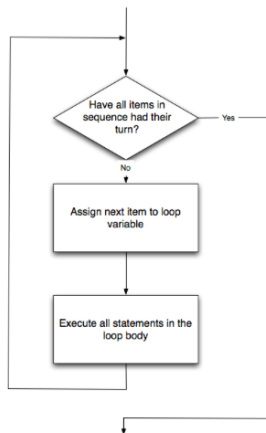
# for-loop



```
for i in list:  
    statement1  
    statement2  
    statement3
```

*How to Think Like CS, §4.5*

# for-loop



*How to Think Like CS, §4.5*

```
for i in list:  
    statement1  
    statement2  
    statement3
```

where `list` is a list of items:

- stated explicitly (e.g. `[1,2,3]`) or
- generated by a function, e.g. `range()`.

# Today's Topics



- For-loops
- **range()**
- Variables
- Characters
- Strings
- Guests: Internships, Advising & Clubs

## More on range():

```
1 #Predict what will be printed:
2
3 for num in [2,4,6,8,10]:
4     print(num)
5
6 sum = 0
7 for x in range(0,12,2):
8     print(x)
9     sum = sum + x
10
11 print(sum)
12
13 for c in "ABCD":
14     print(c)
```

# Python Tutor

```
1 #Predict what will be printed:
2
3 for num in [2,4,6,8,10]:
4     print(num)
5
6 sum = 0
7 for x in range(0,12,2):
8     print(x)
9     sum = sum + x
10
11 print(sum)
12
13 for c in "ABCD":
14     print(c)
```

(Demo with pythonTutor)

# range()



Simplest version:

- `range(stop)`

# range()



Simplest version:

- `range(stop)`
- Produces a list: `[0,1,2,3,...,stop-1]`



# range()



Simplest version:

- `range(stop)`
- Produces a list: `[0,1,2,3,...,stop-1]`
- For example, if you want the the list `[0,1,2,3,...,100]`, you would write:

# range()



Simplest version:

- `range(stop)`
- Produces a list: `[0,1,2,3,...,stop-1]`
- For example, if you want the the list `[0,1,2,3,...,100]`, you would write:

```
range(101)
```

`range()`

What if you wanted to start somewhere else:



# range()

What if you wanted to start somewhere else:

- `range(start, stop)`



# range()



What if you wanted to start somewhere else:

- `range(start, stop)`
- Produces a list:  
`[start, start+1, ..., stop-1]`

# range()



What if you wanted to start somewhere else:

- `range(start, stop)`
- Produces a list:  
`[start, start+1, ..., stop-1]`
- For example, if you want the the list  
`[10, 11, ..., 20]`  
you would write:

# range()



What if you wanted to start somewhere else:

- `range(start, stop)`
- Produces a list:  
`[start, start+1, ..., stop-1]`
- For example, if you want the the list  
`[10, 11, ..., 20]`  
you would write:

```
range(10, 21)
```

# range()

What if you wanted to count by twos, or some other number:





# range()

What if you wanted to count by twos, or some other number:

- `range(start, stop, step)`



# range()

What if you wanted to count by twos, or some other number:

- `range(start, stop, step)`
- Produces a list:  
`[start, start+step, start+2*step..., last]`  
(where last is the largest  $start+k*step$  less than stop)



# range()



What if you wanted to count by twos, or some other number:

- `range(start, stop, step)`
- Produces a list:  
`[start, start+step, start+2*step..., last]`  
(where last is the largest  $\text{start} + k * \text{step}$  less than stop)
- For example, if you want the the list `[5, 10, ..., 50]` you would write:

# range()

What if you wanted to count by twos, or some other number:

- `range(start, stop, step)`
- Produces a list:  
`[start, start+step, start+2*step..., last]`  
(where last is the largest  $\text{start} + k \cdot \text{step}$  less than stop)
- For example, if you want the the list `[5, 10, ..., 50]` you would write:

```
range(5, 51, 5)
```



In summary: `range()`



The three versions:

In summary: `range()`



The three versions:

- `range(stop)`

In summary: `range()`



The three versions:

- `range(stop)`
- `range(start, stop)`

In summary: `range()`



The three versions:

- `range(stop)`
- `range(start, stop)`
- `range(start, stop, step)`



# Today's Topics



- For-loops
- `range()`
- **Variables**
- Characters
- Strings
- Guests: Internships, Advising & Clubs

# Variables

- A **variable** is a reserved memory location for storing a value.



# Variables

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers



# Variables

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers



# Variables

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters



# Variables

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items



# Variables



- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items  
e.g. [3, 1, 4, 5, 9] or  
['violet', 'purple', 'indigo']

# Variables



- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items  
e.g. [3, 1, 4, 5, 9] or  
['violet', 'purple', 'indigo']
  - ▶ **class variables**: for complex objects, like turtles.
- In Python (unlike other languages) you don't need to specify the type; it is deduced by its value.



# Variable Names

- There's some rules about valid names for variables.



# Variable Names



- There's some rules about valid names for variables.
- Can use the underscore ('\_'), upper and lower case letters.

# Variable Names



- There's some rules about valid names for variables.
- Can use the underscore ('\_'), upper and lower case letters.
- Can also use numbers, just can't start a name with a number.

# Variable Names



- There's some rules about valid names for variables.
- Can use the underscore ('\_'), upper and lower case letters.
- Can also use numbers, just can't start a name with a number.
- Can't use symbols (like '+' or '\*') since used for arithmetic.

# Variable Names



- There's some rules about valid names for variables.
- Can use the underscore ('\_'), upper and lower case letters.
- Can also use numbers, just can't start a name with a number.
- Can't use symbols (like '+' or '\*') since used for arithmetic.
- Can't use some words that Python has reserved for itself (e.g. `for`).  
(List of reserved words in *Think CS*, §2.5.)

# Today's Topics



- For-loops
- `range()`
- Variables
- **Characters**
- Strings
- Guests: Internships, Advising & Clubs

# Standardized Code for Characters

American Standard Code for Information Interchange (ASCII), 1960.

# Standardized Code for Characters

American Standard Code for Information Interchange (ASCII), 1960.  
(New version called: Unicode).



# Standardized Code for Characters

American Standard Code for Information Interchange (ASCII), 1960.  
(New version called: Unicode).

## ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

(wiki)

# Converting from Character to Code:

*(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)*

## ASCII TABLE

Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char
0		16	P	32	@	48	0
1	SOH	17	Q	33	A	49	1
2	STX	18	R	34	B	50	2
3	ETX	19	S	35	C	51	3
4	END	20	T	36	D	52	4
5	SO	21	U	37	E	53	5
6	ST	22	V	38	F	54	6
7	HT	23	W	39	G	55	7
8	LF	24	X	40	H	56	8
9	VT	25	Y	41	I	57	9
10	FF	26	Z	42	J	58	:
11		27	[	43	K	59	;
12		28	\	44	L	60	<
13	CR	29	]	45	M	61	=
14		30	^	46	N	62	>
15		31	_	47	O	63	?
16	P	32	@	48	0	64	SP
17	Q	33	A	49	1	65	a
18	R	34	B	50	2	66	b
19	S	35	C	51	3	67	c
20	T	36	D	52	4	68	d
21	U	37	E	53	5	69	e
22	V	38	F	54	6	70	f
23	W	39	G	55	7	71	g
24	X	40	H	56	8	72	h
25	Y	41	I	57	9	73	i
26	Z	42	J	58	:	74	j
27	[	43	K	59	;	75	k
28	\	44	L	60	<	76	l
29	]	45	M	61	=	77	m
30	^	46	N	62	>	78	n
31	_	47	O	63	?	79	o
32	@	48	0	64	SP	80	p
33	A	49	1	65	a	81	q
34	B	50	2	66	b	82	r
35	C	51	3	67	c	83	s
36	D	52	4	68	d	84	t
37	E	53	5	69	e	85	u
38	F	54	6	70	f	86	v
39	G	55	7	71	g	87	w
40	H	56	8	72	h	88	x
41	I	57	9	73	i	89	y
42	J	58	:	74	j	90	z
43	K	59	;	75	k	91	{
44	L	60	<	76	l	92	
45	M	61	=	77	m	93	}
46	N	62	>	78	n	94	~
47	O	63	?	79	o	95	
48	0	64	SP	80	p	96	
49	1	65	a	81	q	97	
50	2	66	b	82	r	98	
51	3	67	c	83	s	99	
52	4	68	d	84	t	100	
53	5	69	e	85	u	101	
54	6	70	f	86	v	102	
55	7	71	g	87	w	103	
56	8	72	h	88	x	104	
57	9	73	i	89	y	105	
58	:	74	j	90	z	106	
59	;	75	k	91	{	107	
60	<	76	l	92		108	
61	=	77	m	93	}	109	
62	>	78	n	94	~	110	
63	?	79	o	95		111	
64	SP	80	p	96		112	
65	a	81	q	97		113	
66	b	82	r	98		114	
67	c	83	s	99		115	
68	d	84	t	100		116	
69	e	85	u	101		117	
70	f	86	v	102		118	
71	g	87	w	103		119	
72	h	88	x	104		120	
73	i	89	y	105		121	
74	j	90	z	106		122	
75	k	91	{	107		123	
76	l	92		108		124	
77	m	93	}	109		125	
78	n	94	~	110		126	
79	o	95		111		127	DEL

# Converting from Character to Code:

*(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)*

- `ord(c)`: returns Unicode (ASCII) of the character.

**ASCII TABLE**

Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char
0		16	0	32	@	48	0
1		17	1	33	!	49	1
2		18	2	34	"	50	2
3		19	3	35	#	51	3
4		20	4	36	\$	52	4
5		21	5	37	%	53	5
6		22	6	38	&	54	6
7		23	7	39	'	55	7
8		24	8	40	(	56	8
9		25	9	41	)	57	9
10		26	A	42	*	58	A
11		27	B	43	+	59	B
12		28	C	44	,	60	C
13		29	D	45	-	61	D
14		30	E	46	.	62	E
15		31	F	47	/	63	F
16	0	32	@	48	0	64	0
17	1	33	!	49	1	65	1
18	2	34	"	50	2	66	2
19	3	35	#	51	3	67	3
20	4	36	\$	52	4	68	4
21	5	37	%	53	5	69	5
22	6	38	&	54	6	70	6
23	7	39	'	55	7	71	7
24	8	40	(	56	8	72	8
25	9	41	)	57	9	73	9
26	A	42	*	58	A	74	A
27	B	43	+	59	B	75	B
28	C	44	,	60	C	76	C
29	D	45	-	61	D	77	D
30	E	46	.	62	E	78	E
31	F	47	/	63	F	79	F
32	0	48	0	64	0	80	0
33	1	49	1	65	1	81	1
34	2	50	2	66	2	82	2
35	3	51	3	67	3	83	3
36	4	52	4	68	4	84	4
37	5	53	5	69	5	85	5
38	6	54	6	70	6	86	6
39	7	55	7	71	7	87	7
40	8	56	8	72	8	88	8
41	9	57	9	73	9	89	9
42	A	58	A	74	A	90	A
43	B	59	B	75	B	91	[
44	C	60	C	76	C	92	]
45	D	61	D	77	D	93	^
46	E	62	E	78	E	94	_
47	F	63	F	79	F	95	`
48	0	64	0	80	0	96	a
49	1	65	1	81	1	97	b
50	2	66	2	82	2	98	c
51	3	67	3	83	3	99	d
52	4	68	4	84	4	100	e
53	5	69	5	85	5	101	f
54	6	70	6	86	6	102	g
55	7	71	7	87	7	103	h
56	8	72	8	88	8	104	i
57	9	73	9	89	9	105	j
58	A	74	A	90	A	106	k
59	B	75	B	91	[	107	l
60	C	76	C	92	]	108	m
61	D	77	D	93	^	109	n
62	E	78	E	94	_	110	o
63	F	79	F	95	`	111	p
64	0	80	0	96	a	112	q
65	1	81	1	97	b	113	r
66	2	82	2	98	c	114	s
67	3	83	3	99	d	115	t
68	4	84	4	100	e	116	u
69	5	85	5	101	f	117	v
70	6	86	6	102	g	118	w
71	7	87	7	103	h	119	x
72	8	88	8	104	i	120	y
73	9	89	9	105	j	121	z
74	A	90	A	106	k	122	{
75	B	91	[	107	l	123	}
76	C	92	]	108	m	124	~
77	D	93	^	109	n	125	
78	E	94	_	110	o		
79	F	95	`	111	p		
80	0	96	a	112	q		
81	1	97	b	113	r		
82	2	98	c	114	s		
83	3	99	d	115	t		
84	4	100	e	116	u		
85	5	101	f	117	v		
86	6	102	g	118	w		
87	7	103	h	119	x		
88	8	104	i	120	y		
89	9	105	j	121	z		
90	A	106	k	122	{		
91	B	107	l	123	}		
92	C	108	m	124	~		
93	D	109	n	125			
94	E	110	o				
95	F	111	p				
96	0	112	q				
97	1	113	r				
98	2	114	s				
99	3	115	t				
100	4	116	u				
101	5	117	v				
102	6	118	w				
103	7	119	x				
104	8	120	y				
105	9	121	z				
106	A	122	{				
107	B	123	}				
108	C	124	~				
109	D	125					
110	E						
111	F						
112	0						
113	1						
114	2						
115	3						
116	4						
117	5						
118	6						
119	7						
120	8						
121	9						
122	A						
123	B						
124	C						
125	D						

# Converting from Character to Code:

*(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)*

**ASCII TABLE**

Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char
0		16		32		48	
1		17		33	!	49	1
2		18		34	"	50	2
3		19		35	#	51	3
4		20		36	\$	52	4
5		21		37	%	53	5
6		22		38	&	54	6
7		23		39	'	55	7
8		24		40	(	56	8
9		25		41	)	57	9
10		26		42	*	58	:
11		27		43	+	59	;
12		28		44	,	60	<
13		29		45	-	61	=
14		30		46	.	62	>
15		31		47	/	63	?
16		32	!	64	@	80	P
17		33	"	65	A	81	Q
18		34	"	66	B	82	R
19		35	"	67	C	83	S
20		36	"	68	D	84	T
21		37	"	69	E	85	U
22		38	"	70	F	86	V
23		39	"	71	G	87	W
24		40	"	72	H	88	X
25		41	"	73	I	89	Y
26		42	"	74	J	90	Z
27		43	"	75	K	91	[
28		44	"	76	L	92	\
29		45	"	77	M	93	]
30		46	"	78	N	94	^
31		47	"	79	O	95	_
32	!	48	0	80	P	96	`
33	"	49	1	81	Q	97	a
34	"	50	2	82	R	98	b
35	"	51	3	83	S	99	c
36	"	52	4	84	T	100	d
37	"	53	5	85	U	101	e
38	"	54	6	86	V	102	f
39	"	55	7	87	W	103	g
40	"	56	8	88	X	104	h
41	"	57	9	89	Y	105	i
42	"	58	:	90	Z	106	j
43	"	59	;	91	[	107	k
44	"	60	<	92	\	108	l
45	"	61	=	93	]	109	m
46	"	62	>	94	^	110	n
47	"	63	?	95	_	111	o
48	0	64	@	96	`	112	p
49	1	65	A	97	a	113	q
50	2	66	B	98	b	114	r
51	3	67	C	99	c	115	s
52	4	70	F	100	d	116	t
53	5	71	G	101	e	117	u
54	6	72	H	102	f	118	v
55	7	73	I	103	g	119	w
56	8	74	J	104	h	120	x
57	9	75	K	105	i	121	y
58	:	76	L	106	j	122	z
59	;	77	M	107	k	123	{
60	<	78	N	108	l	124	
61	=	79	O	109	m	125	}
62	>	80	P	110	n	126	~
63	?	81	Q	111	o	127	

- `ord(c)`: returns Unicode (ASCII) of the character.
- Example: `ord('a')` returns 97.

# Converting from Character to Code:

*(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)*

**ASCII TABLE**

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	00		16	10	P	32	20	[	48	30	0
1	01		17	11	Q	33	21	\	49	31	1
2	02		18	12	R	34	22	]	50	32	2
3	03		19	13	S	35	23	^	51	33	3
4	04		20	14	T	36	24	_	52	34	4
5	05		21	15	U	37	25	`	53	35	5
6	06		22	16	V	38	26	{	54	36	6
7	07		23	17	W	39	27		55	37	7
8	08		24	18	X	40	28	~	56	38	8
9	09		25	19	Y	41	29		57	39	9
10	0A		26	1A	Z	42	2A		58	3A	.
11	0B		27	1B	[	43	2B		59	3B	,
12	0C		28	1C	\	44	2C		60	3C	;
13	0D		29	1D	]	45	2D		61	3D	'
14	0E		30	1E	^	46	2E		62	3E	"
15	0F		31	1F	_	47	2F		63	3F	!
16	10	@	32	20	[	48	30	0	64	40	+
17	11	A	33	21	\	49	31	1	65	41	*
18	12	B	34	22	]	50	32	2	66	42	-
19	13	C	35	23	^	51	33	3	67	43	=
20	14	D	36	24	_	52	34	4	68	44	&
21	15	E	37	25	`	53	35	5	69	45	%
22	16	F	38	26	{	54	36	6	70	46	@
23	17		39	27		55	37	7	71	47	#
24	18		40	28	~	56	38	8	72	48	\$
25	19		41	29		57	39	9	73	49	%
26	1A		42	2A		58	3A	.	74	4A	&
27	1B		43	2B		59	3B	,	75	4B	'
28	1C		44	2C		60	3C	;	76	4C	"
29	1D		45	2D		61	3D	'	77	4D	!
30	1E		46	2E		62	3E	"	78	4E	+
31	1F		47	2F		63	3F	!	79	4F	*
32	20		48	30	0	64	40	+	80	50	^
33	21	!	49	31	1	65	41	*	81	51	_
34	22	"	50	32	2	66	42	-	82	52	~
35	23	"	51	33	3	67	43	=	83	53	~
36	24	\$	52	34	4	68	44	&	84	54	~
37	25	%	53	35	5	69	45	%	85	55	~
38	26	&	54	36	6	70	46	@	86	56	~
39	27	'	55	37	7	71	47	#	87	57	~
40	28	(	56	38	8	72	48	\$	88	58	~
41	29	)	57	39	9	73	49	%	89	59	~
42	2A	*	58	3A	.	74	4A	&	90	5A	~
43	2B	+	59	3B	,	75	4B	'	91	5B	~
44	2C	,	60	3C	;	76	4C	"	92	5C	~
45	2D	-	61	3D	'	77	4D	!	93	5D	~
46	2E	.	62	3E	"	78	4E	+	94	5E	~
47	2F	/	63	3F	!	79	4F	*	95	5F	~
48	30	0	64	40	+	80	50	^	96	60	~
49	31	1	65	41	*	81	51	_	97	61	~
50	32	2	66	42	-	82	52	~	98	62	~
51	33	3	67	43	=	83	53	~	99	63	~
52	34	4	68	44	&	84	54	~	100	64	~
53	35	5	69	45	%	85	55	~			
54	36	6	70	46	@	86	56	~			
55	37	7	71	47	#	87	57	~			
56	38	8	72	48	\$	88	58	~			
57	39	9	73	49	%	89	59	~			
58	3A	.	74	4A	&	90	5A	~			
59	3B	,	75	4B	'	91	5B	~			
60	3C	;	76	4C	"	92	5C	~			
61	3D	'	77	4D	!	93	5D	~			
62	3E	"	78	4E	+	94	5E	~			
63	3F	!	79	4F	*	95	5F	~			
64	40	+	80	50	^	96	60	~			
65	41	*	81	51	_	97	61	~			
66	42	-	82	52	~	98	62	~			
67	43	=	83	53	~	99	63	~			
68	44	&	84	54	~						
69	45	%	85	55	~						
70	46	@	86	56	~						
71	47	#	87	57	~						
72	48	\$	88	58	~						
73	49	%	89	59	~						
74	4A	&	90	5A	~						
75	4B	'	91	5B	~						
76	4C	"	92	5C	~						
77	4D	!	93	5D	~						
78	4E	+	94	5E	~						
79	4F	*	95	5F	~						
80	50	^	96	60	~						
81	51	_	97	61	~						
82	52	~	98	62	~						
83	53	~	99	63	~						
84	54	~									
85	55	~									
86	56	~									
87	57	~									
88	58	~									
89	59	~									
90	5A	~									
91	5B	~									
92	5C	~									
93	5D	~									
94	5E	~									
95	5F	~									
96	60	~									
97	61	~									
98	62	~									
99	63	~									
100	64	~									

- `ord(c)`: returns Unicode (ASCII) of the character.
- Example: `ord('a')` returns 97.
- `chr(x)`: returns the character whose Unicode is x.

# Converting from Character to Code:

*(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)*

**ASCII TABLE**

Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char
0		16	P	32	@	48	0
1	SOH	17	Q	33	A	49	1
2	STX	18	R	34	B	50	2
3	ETX	19	S	35	C	51	3
4	END	20	T	36	D	52	4
5	SO	21	U	37	E	53	5
6	SI	22	V	38	F	54	6
7	DEL	23	W	39	G	55	7
8		24	X	40	H	56	8
9		25	Y	41	I	57	9
10	LF	26	Z	42	J	58	:
11		27	[	43	K	59	;
12	FF	28	\	44	L	60	<
13	CR	29	]	45	M	61	=
14		30	^	46	N	62	>
15		31	_	47	O	63	?

- `ord(c)`: returns Unicode (ASCII) of the character.
- Example: `ord('a')` returns 97.
- `chr(x)`: returns the character whose Unicode is x.
- Example: `chr(97)` returns 'a'.

# Converting from Character to Code:

*(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)*

**ASCII TABLE**

Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char	Decimal	Hex Char
0		16	P	32	@	48	0
1	SOH	17	Q	33	A	49	1
2	STX	18	R	34	B	50	2
3	ETX	19	S	35	C	51	3
4	END	20	T	36	D	52	4
5	SO	21	U	37	E	53	5
6	ST	22	V	38	F	54	6
7	HT	23	W	39	G	55	7
8	LF	24	X	40	H	56	8
9	VT	25	Y	41	I	57	9
10	FF	26	Z	42	J	58	:
11		27	[	43	K	59	;
12		28	\	44	L	60	<
13	CR	29	]	45	M	61	=
14		30	^	46	N	62	>
15		31	_	47	O	63	?

- `ord(c)`: returns Unicode (ASCII) of the character.
- Example: `ord('a')` returns 97.
- `chr(x)`: returns the character whose Unicode is x.
- Example: `chr(97)` returns 'a'.
- What is `chr(33)`?

# In Pairs or Triples...

*Some review and some novel challenges:*

```
1 #Predict what will be printed:
2
3 for c in range(65,90):
4     print(chr(c))
5
6 message = "I love Python"
7 newMessage = ""
8 for c in message:
9     print(ord(c))    #Print the Unicode of each number
10    print(chr(ord(c)+1))    #Print the next character
11    newMessage = newMessage + chr(ord(c)+1) #add to the new message
12 print("The coded message is", newMessage)
13
14 word = "zebra"
15 codedWord = ""
16 for ch in word:
17     offset = ord(ch) - ord('a') + 1 #how many letters past 'a'
18     wrap = offset % 26 #if larger than 26, wrap back to 0
19     newChar = chr(ord('a') + wrap) #compute the new letter
20     print(wrap, chr(ord('a') + wrap))    #print the wrap & new lett
21     codedWord = codedWord + newChar #add the newChar to the coded w
22
23 print("The coded word (with wrap) is", codedWord)
```



# Python Tutor

```
1 #Predict what will be printed:
2
3 for c in range(65,90):
4     print(chr(c))
5
6 message = "I love Python"
7 newMessage = ""
8 for c in message:
9     print(ord(c))    #Print the Unicode of each number
10    print(chr(ord(c)+1))    #Print the next character
11    newMessage = newMessage + chr(ord(c)+1) #Add to the new message
12 print("The coded message is", newMessage)
13
14 word = "zebra"
15 codedWord = ""
16 for ch in word:
17     offset = ord(ch) - ord('a') + 1 #How many letters past 'a'
18     wrap = offset % 26 #if larger than 26, wrap back to 0
19     newChar = chr(ord('a') + wrap) #compute the new letter
20     print(wrap, chr(ord('a') + wrap))    #Print the wrap & new lett
21     codedWord = codedWord + newChar #add the newChar to the coded w
22
23 print("The coded word (with wrap) is", codedWord)
```

(Demo with pythonTutor)

# Wrap

<code>chr()</code>	a	b	c				...				x	y	z
<code>ord()</code>	97	98	99				...				120	121	122

**offset:** how many letters past 'a'?

**wrap:** if offset > 26 then wrap around  
% is the remainder  
 $27 \% 26 = 1$

# User Input

*Covered in detail in Lab 2:*

---

```
➔ 1 mess = input('Please enter a message: ')\n   2 print("You entered", mess)
```

---

(Demo with pythonTutor)

## Side Note: '+' for numbers and strings



- `x = 3 + 5` stores the number 8 in memory location `x`.

## Side Note: '+' for numbers and strings



- $x = 3 + 5$  stores the number 8 in memory location  $x$ .
- $x = x + 1$  increases  $x$  by 1.

## Side Note: '+' for numbers and strings



- `x = 3 + 5` stores the number 8 in memory location `x`.
- `x = x + 1` increases `x` by 1.
- `s = "hi" + "Mom"` stores "hiMom" in memory locations `s`.

## Side Note: '+' for numbers and strings



- `x = 3 + 5` stores the number 8 in memory location `x`.
- `x = x + 1` increases `x` by 1.
- `s = "hi" + "Mom"` stores "hiMom" in memory locations `s`.
- `s = s + "A"` adds the letter "A" to the end of the strings `s`.

# Lecture Quiz

- Log-in to Gradescope
- Find LECTURE 2 Quiz
- Take the quiz
- **You have 3 minutes**



# Today's Topics



- For-loops
- `range()`
- Variables
- Characters
- **Strings**
- Guests: Internships, Advising & Clubs

# More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"

# More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"
- There are many useful functions for strings (more in Lab 2).

# More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"
- There are many useful functions for strings (more in Lab 2).
- `s.count(x)` will count the number of times the pattern, `x`, appears in `s`.

# More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"
- There are many useful functions for strings (more in Lab 2).
- `s.count(x)` will count the number of times the pattern, `x`, appears in `s`.
  - ▶ `s.count("s")` counts the number of lower case `s` that occurs.

## More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"
- There are many useful functions for strings (more in Lab 2).
- `s.count(x)` will count the number of times the pattern, `x`, appears in `s`.
  - ▶ `s.count("s")` counts the number of lower case `s` that occurs.
  - ▶ `num = s.count("s")` stores the result in the variable `num`, for later.

# More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"
- There are many useful functions for strings (more in Lab 2).
- `s.count(x)` will count the number of times the pattern, `x`, appears in `s`.
  - ▶ `s.count("s")` counts the number of lower case `s` that occurs.
  - ▶ `num = s.count("s")` stores the result in the variable `num`, for later.
  - ▶ What would `print(s.count("sS"))` output?

## More on Strings: String Methods

```
s = "FridaysSaturdaysSundays"  
num = s.count("s")
```

- The first line creates a variable, called `s`, that stores the string: "FridaysSaturdaysSundays"
- There are many useful functions for strings (more in Lab 2).
- `s.count(x)` will count the number of times the pattern, `x`, appears in `s`.
  - ▶ `s.count("s")` counts the number of lower case `s` that occurs.
  - ▶ `num = s.count("s")` stores the result in the variable `num`, for later.
  - ▶ What would `print(s.count("sS"))` output?
  - ▶ What about:

```
mess = "10 20 21 9 101 35"  
mults = mess.count("0 ")  
print(mults)
```



## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

# More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[0]` is

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[0]` is 'F'.

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[1]` is

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[1]` is `'r'`.



## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[-1]` is

# More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[-1]` is 's'.

# More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[3:6]` is

# More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[3:6]` is 'day'.

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[:3]` is

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[:3]` is 'Fri'.

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[:-1]` is

## More on Strings: Indexing & Substrings

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- Strings are made up of individual characters (letters, numbers, etc.)
- Useful to be able to refer to pieces of a string, either an individual location or a “substring” of the string.

0	1	2	3	4	5	6	7	8	...	16	17	18	19	20	21	22
F	r	i	d	a	y	s	S	a	...	S	u	n	d	a	y	s
												...	-4	-3	-2	-1

- `s[:-1]` is 'FridaysSaturdaysSunday'.  
(no trailing 's' at the end)



## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.

# More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

"Friday~~s~~Saturday~~s~~Sunday"

## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

```
"FridayXSaturdayXSunday"  
days = ['Friday', 'Saturday', 'Sunday']
```

## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

```
"FridayXSaturdayXSunday"  
days = ['Friday', 'Saturday', 'Sunday']
```

- Different delimiters give different lists:

## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

```
"FridayXSaturdayXSunday"  
days = ['Friday', 'Saturday', 'Sunday']
```

- Different delimiters give different lists:

```
days = s[:-1].split("day")
```

## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

```
"FridaysSaturdaysSunday"  
days = ['Friday', 'Saturday', 'Sunday']
```

- Different delimiters give different lists:

```
days = s[:-1].split("day")  
"FridaysSaturdaysSunday"
```

## More on Strings: Splits

```
s = "FridaysSaturdaysSundays"  
days = s[:-1].split("s")
```

- `split()` divides a string into a list.
- Cross out the delimiter, and the remaining items are the list.

```
"FridaysSaturdaysSunday"  
days = ['Friday', 'Saturday', 'Sunday']
```

- Different delimiters give different lists:

```
days = s[:-1].split("day")  
"FridaysSaturdaysSunday"  
days = ['Fri', 'sSatur', 'sSun']
```



# Today's Topics



- For-loops
- `range()`
- Variables
- Characters
- Strings
- **Guests: Internships, Advising & Clubs**

# Guest Speakers

- Announcement on Blackboard:
  - ▶ Advising
  - ▶ Programs and Clubs Handout
  - ▶ Internships Handout
  - ▶ Hunter CS Handbook
  - ▶ PreTech Center (formerly CUNY2X) Newsletter

# Recap

- In Python, we introduced:

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

# Recap

- In Python, we introduced:
  - For-loops

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

# Recap

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

# Recap

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`
- ▶ Variables: ints and strings

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

# Recap

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`
- ▶ Variables: ints and strings
- ▶ Some arithmetic

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

# Recap

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`
- ▶ Variables: ints and strings
- ▶ Some arithmetic
- ▶ String concatenation

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```



# Recap

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`
- ▶ Variables: ints and strings
- ▶ Some arithmetic
- ▶ String concatenation
- ▶ Functions: `ord()` and `chr()`

# Recap

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`
- ▶ Variables: ints and strings
- ▶ Some arithmetic
- ▶ String concatenation
- ▶ Functions: `ord()` and `chr()`
- ▶ String Manipulation

# Recap

```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12         print('Look around,')
13     print('How lucky we are to be alive!')
```

- In Python, we introduced:

- ▶ For-loops
- ▶ `range()`
- ▶ Variables: ints and strings
- ▶ Some arithmetic
- ▶ String concatenation
- ▶ Functions: `ord()` and `chr()`
- ▶ String Manipulation

# Practice Quiz & Final Questions



- Since you must pass the final exam to pass the course, we end every lecture with final exam review.

# Practice Quiz & Final Questions



- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - ▶ write as much you can for 60 seconds;
  - ▶ followed by answer; and
  - ▶ repeat.
- Past exams are on the webpage (under [Final Exam Information](#)).
- We're starting with Spring 2018, Mock Exam.

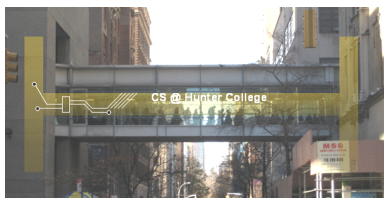
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab

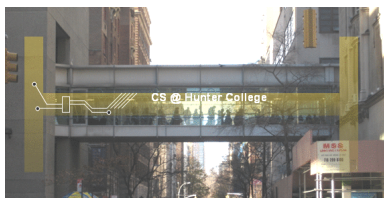
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North

# Weekly Reminders!

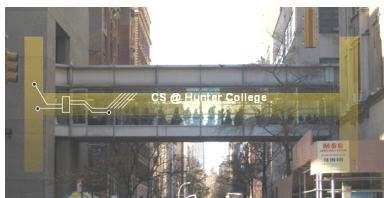


Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North



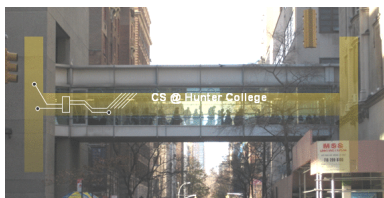
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's [5 programming assignments](#) (programs 6-10)

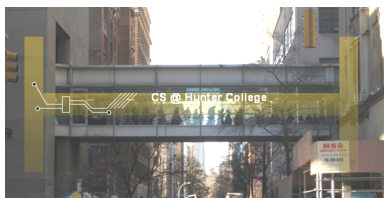
# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's [5 programming assignments](#) (programs 6-10)
- If you need help, [schedule an appointment for Tutoring](#) in lab 1001E 11am-5pm

# Weekly Reminders!



Before next lecture, don't forget to:

- Work on this week's Online Lab
- [Schedule an appointment to take the Quiz](#) in lab 1001E Hunter North
- If you haven't already, [schedule an appointment to take the Code Review](#) (**one every two weeks**) in lab 1001E Hunter North
- Submit this week's [5 programming assignments](#) (programs 6-10)
- If you need help, [schedule an appointment for Tutoring](#) in lab 1001E 11am-5pm
- Take the Lecture Preview on Blackboard on Monday (or no later than 10am on Tuesday)