

CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From previous semesters

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must:*

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges;*

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture);*

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!);*

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*

- Can I work ahead?

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*

- Can I work ahead?

Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?

No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*

- Can I work ahead?

Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)

- When is the midterm?

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*
- Can I work ahead?
Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)
- When is the midterm?
There is no midterm. Instead there's required quizzes each class and 60 programming assignments.

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*
- Can I work ahead?
Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)
- When is the midterm?
There is no midterm. Instead there's required quizzes each class and 60 programming assignments.
- I missed class. Do you need documentation?

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*
- Can I work ahead?
Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)
- When is the midterm?
There is no midterm. Instead there's required quizzes each class and 60 programming assignments.
- I missed class. Do you need documentation?
No, but If you will miss ≥ 2 class meetings ($> 20\%$), see us about taking this in a future term. The summer semester moves VERY quickly. Missing any class is not ideal.

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*
- Can I work ahead?
Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)
- When is the midterm?
There is no midterm. Instead there's required quizzes each class and 60 programming assignments.
- I missed class. Do you need documentation?
No, but If you will miss ≥ 2 class meetings ($> 20\%$), see us about taking this in a future term. The summer semester moves VERY quickly. Missing any class is not ideal.
- I have not received any emails from this course.

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*
- Can I work ahead?
Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)
- When is the midterm?
There is no midterm. Instead there's required quizzes each class and 60 programming assignments.
- I missed class. Do you need documentation?
No, but If you will miss ≥ 2 class meetings ($> 20\%$), see us about taking this in a future term. The summer semester moves VERY quickly. Missing any class is not ideal.
- I have not received any emails from this course.
That is a big problem! *We send tons of important information through email. Please update your email on Blackboard to one you check regularly.*

Frequently Asked Questions

From previous semesters

- Am I only responsible for reading the weekly Lab?
No. *Each class you must: Attend lecture; Participate in challenges; Take a Quiz (on Gradescope, immediately after lecture); Read the weekly lab (online, see Course Outline on **course website**, find it on Blackboard!!!); Submit programming assignments to Gradescope (approx. 6 per lab)*
- Can I work ahead?
Absolutely! Submission is open on Gradescope, 2 weeks before the deadline. Start right away (after Lab 1 you can submit the first 5 problems)
- When is the midterm?
There is no midterm. Instead there's required quizzes each class and 60 programming assignments.
- I missed class. Do you need documentation?
No, but If you will miss ≥ 2 class meetings ($> 20\%$), see us about taking this in a future term. The summer semester moves VERY quickly. Missing any class is not ideal.
- I have not received any emails from this course.
That is a big problem! *We send tons of important information through email. Please update your email on Blackboard to one you check regularly.*

Today's Topics



- **For-loops**
- `range()`
- Variables
- Characters
- Strings
- CS Survey (Dr. Sakas, Computational Linguistics)

Challenge Problem...

Some review and some novel challenges:

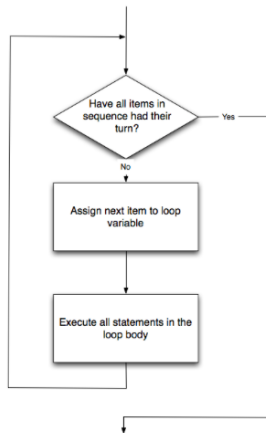
```
1 #Predict what will be printed:
2 for i in range(4):
3     print('The world turned upside down')
4 for j in [0,1,2,3,4,5]:
5     print(j)
6 for count in range(6):
7     print(count)
8 for color in ['red', 'green', 'blue']:
9     print(color)
10 for i in range(2):
11     for j in range(2):
12   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

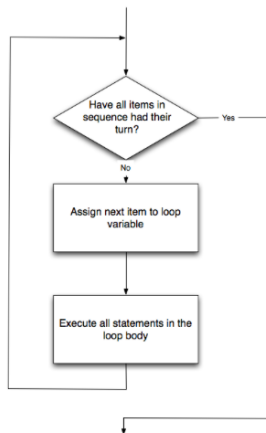
for-loop



```
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
- CS Survey (Dr. Sakas, Computational Linguistics)

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 $\text{start} + k * \text{step}$ 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 $\text{start} + k * \text{step}$ 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
- CS Survey (Dr. Sakas, Computational Linguistics)

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
- CS Survey (Dr. Sakas, Computational Linguistics)

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 an ASCII table on the back of today's lecture slip.)

ASCII TABLE

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

Converting from Character to Code:

(There is an ASCII table on the back of today's lecture slip.)

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

Converting from Character to Code:

(There is an ASCII table on the back of today's lecture slip.)

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	
64	@	82	R	112	p		
65	A	83	S	113	q		
66	B	84	T	114	r		
67	C	85	U	115	s		
68	D	86	V	116	t		
69	E	87	W	117	u		
70	F	88	X	118	v		
71	G	89	Y	119	w		
72	H	90	Z	120	x		
73	I	91	[121	y		
74	J	92	\	122	z		
75	K	93]				
76	L	94	^				
77	M	95	_				
78	N	96	`				
79	O	97	a				
80	P	98	b				
81	Q	99	c				
82	R	100	d				
83	S	101	e				
84	T	102	f				
85	U	103	g				
86	V	104	h				
87	W	105	i				
88	X	106	j				
89	Y	107	k				
90	Z	108	l				
91	[109	m				
92	\	110	n				
93]	111	o				
94	^	112	p				
95	_	113	q				
96	`	114	r				
97	a	115	s				
98	b	116	t				
99	c	117	u				
100	d	118	v				
101	e	119	w				
102	f	120	x				
103	g	121	y				
104	h	122	z				
105	i						
106	j						
107	k						
108	l						
109	m						
110	n						
111	o						
112	p						
113	q						
114	r						
115	s						
116	t						
117	u						
118	v						
119	w						
120	x						
121	y						
122	z						
123	{						
124							
125	}						
126	~						
127							

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

Converting from Character to Code:

(There is an ASCII table on the back of today's lecture slip.)

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	A
28	1C		44	2C		60	3C	;	76	4C	M
29	1D		45	2D		61	3D	"	77	4D	Y
30	1E		46	2E		62	3E	'	78	4E	P
31	1F		47	2F		63	3F	~	79	4F	B
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	A	91	5B	
44	2C	,	60	3C	;	76	4C	M	92	5C	
45	2D	-	61	3D	"	77	4D	Y	93	5D	
46	2E	.	62	3E	'	78	4E	P	94	5E	
47	2F	/	63	3F	~	79	4F	B	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				
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	A	91	5B				
60	3C	;	76	4C	M	92	5C				
61	3D	"	77	4D	Y	93	5D				
62	3E	'	78	4E	P	94	5E				
63	3F	~	79	4F	B	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	A	91	5B							
76	4C	M	92	5C							
77	4D	Y	93	5D							
78	4E	P	94	5E							
79	4F	B	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										

- `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 an ASCII table on the back of today's lecture slip.)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	00		16	10	P	32	20	R	48	30	T
1	01	SOH	17	11	Q	33	21	S	49	31	U
2	02	STX	18	12	R	34	22	T	50	32	V
3	03	ETX	19	13	S	35	23	U	51	33	W
4	04	END	20	14	T	36	24	V	52	34	X
5	05	SO	21	15	U	37	25	W	53	35	Y
6	06	SI	22	16	V	38	26	X	54	36	Z
7	07	BS	23	17	W	39	27	Y	55	37	[
8	08	HT	24	18	X	40	28	Z	56	38	\
9	09	LF	25	19	Y	41	29	[57	39]
10	0A	VT	26	1A	Z	42	2A	\	58	3A	^
11	0B	FF	27	1B	[43	2B]	59	3B	_
12	0C	DEL	28	1C	\	44	2C	^	60	3C	`
13	0D		29	1D]	45	2D	_	61	3D	a
14	0E		30	1E	^	46	2E	`	62	3E	b
15	0F		31	1F	_	47	2F	a	63	3F	c
16	10	SP	32	20	R	48	30	T	64	40	d
17	11	P	33	21	S	49	31	U	65	41	e
18	12	Q	34	22	T	50	32	V	66	42	f
19	13	R	35	23	U	51	33	W	67	43	g
20	14	S	36	24	V	52	34	X	68	44	h
21	15	T	37	25	W	53	35	Y	69	45	i
22	16	U	38	26	X	54	36	Z	70	46	j
23	17	V	39	27	Y	55	37	[71	47	k
24	18	W	40	28	Z	56	38	\	72	48	l
25	19	X	41	29	[57	39]	73	49	m
26	1A	Z	42	2A	\	58	3A	^	74	4A	n
27	1B	[43	2B]	59	3B	_	75	4B	o
28	1C	\	44	2C	^	60	3C	`	76	4C	p
29	1D]	45	2D	_	61	3D	a	77	4D	q
30	1E	^	46	2E	`	62	3E	b	78	4E	r
31	1F	_	47	2F	a	63	3F	c	79	4F	s
32	20	R	48	30	T	64	40	d	80	50	t
33	21	S	49	31	U	65	41	e	81	51	u
34	22	T	50	32	V	66	42	f	82	52	v
35	23	U	51	33	W	67	43	g	83	53	w
36	24	V	52	34	X	68	44	h	84	54	x
37	25	W	53	35	Y	69	45	i	85	55	y
38	26	X	54	36	Z	70	46	j	86	56	z
39	27	Y	55	37	[71	47	k	87	57	{
40	28	Z	56	38	\	72	48	l	88	58	
41	29	[57	39]	73	49	m	89	59	~
42	2A	\	58	3A	^	90	5A	DEL			
43	2B]	59	3B	_						
44	2C	^	60	3C	`						
45	2D	_	61	3D	a						
46	2E	`	62	3E	b						
47	2F	a	63	3F	c						
48	30	T	64	40	d						
49	31	U	65	41	e						
50	32	V	66	42	f						
51	33	W	67	43	g						
52	34	X	68	44	h						
53	35	Y	69	45	i						
54	36	Z	70	46	j						
55	37	[71	47	k						
56	38	\	72	48	l						
57	39]	73	49	m						
58	3A	^	74	4A	n						
59	3B	_	75	4B	o						
60	3C	`	76	4C	p						
61	3D	a	77	4D	q						
62	3E	b	78	4E	r						
63	3F	c	79	4F	s						
64	40	d	80	50	t						
65	41	e	81	51	u						
66	42	f	82	52	v						
67	43	g	83	53	w						
68	44	h	84	54	x						
69	45	i	85	55	y						
70	46	j	86	56	z						
71	47	k	87	57	{						
72	48	l	88	58							
73	49	m	89	59	~						
74	4A	n	90	5A	DEL						
75	4B	o									
76	4C	p									
77	4D	q									
78	4E	r									
79	4F	s									
80	50	t									
81	51	u									
82	52	v									
83	53	w									
84	54	x									
85	55	y									
86	56	z									
87	57	{									
88	58										
89	59	~									
90	5A	DEL									

- `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 an ASCII table on the back of today's lecture slip.)

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	`
49	1	65	1	81	1	97	a
50	2	66	2	82	2	98	b
51	3	67	3	83	3	99	c
52	4	68	4	84	4	100	d
53	5	69	5	85	5	101	e
54	6	70	6	86	6	102	f
55	7	71	7	87	7	103	g
56	8	72	8	88	8	104	h
57	9	73	9	89	9	105	i
58	A	74	A	90	A	106	j
59	B	75	B	91	[107	k
60	C	76	C	92	\	108	l
61	D	77	D	93]	109	m
62	E	78	E	94	^	110	n
63	F	79	F	95	_	111	o
64	0	80	0	96	`	112	p
65	1	81	1	97	a	113	q
66	2	82	2	98	b	114	r
67	3	83	3	99	c	115	s
68	4	84	4	100	d	116	t
69	5	85	5	101	e	117	u
70	6	86	6	102	f	118	v
71	7	87	7	103	g	119	w
72	8	88	8	104	h	120	x
73	9	89	9	105	i	121	y
74	A	90	A	106	j	122	z
75	B	91	[107	k	123	{
76	C	92	\	108	l	124	}
77	D	93]	109	m	125	~
78	E	94	^	110	n		
79	F	95	_	111	o		
80	0	96	`	112	p		
81	1	97	a	113	q		
82	2	98	b	114	r		
83	3	99	c	115	s		
84	4	100	d	116	t		
85	5	101	e	117	u		
86	6	102	f	118	v		
87	7	103	g	119	w		
88	8	104	h	120	x		
89	9	105	i	121	y		
90	A	106	j	122	z		
91	[107	k				
92	\	108	l				
93]	109	m				
94	^	110	n				
95	_	111	o				
96	`	112	p				
97	a	113	q				
98	b	114	r				
99	c	115	s				
100	d	116	t				
101	e	117	u				
102	f	118	v				
103	g	119	w				
104	h	120	x				
105	i	121	y				
106	j	122	z				
107	k						
108	l						
109	m						
110	n						
111	o						
112	p						
113	q						
114	r						
115	s						
116	t						
117	u						
118	v						
119	w						
120	x						
121	y						
122	z						
123	{						
124	}						
125	~						

- `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)`?

Challenge Problem...

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)

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`.

Today's Topics



- For-loops
- `range()`
- Variables
- Characters
- **Strings**
- CS Survey (Dr. Sakas, Computational Linguistics)

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.

```
"FridaysSaturdaysSunday"  
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")  
"FrixxxsSaturxxxsSundxxx"
```


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")  
"FrixxxsSaturxxxsSunxxx"  
days = ['Fri', 'sSatur', 'sSun']
```

Today's Topics



- For-loops
- `range()`
- Variables
- Characters
- Strings
- **CS Survey (Dr. Sakas, Computational Linguistics)**

CS Survey: Prof. Sakas, Computational Computational Linguistics



Language is Hard for Computers

Learning Language is Easy for my 3-year-old twins

CSCI 12700 Guest Bullet Talk

William Gregory Sakas



*M.A./Ph.D. Program in Linguistics
@ The City University of New York*

CS Survey: Prof. Sakas, Computational Computational Linguistics



Language is Hard

- *Buffalo buffalo, Buffalo buffalo buffalo, buffalo, Buffalo buffalo*
- *Someone shot the servant of the actress who was on the balcony. Who was on the balcony?*
- *Who do you think Mary kissed?*
- *Who do you think that Mary kissed?*
- *Who do you think bought a radio?*
- ** Who do you think that bought a radio?*



CS Survey: Prof. Sakas, Computational Computational Linguistics



So how to explain language?

Treat Language as a **scientific field** - like **Physics**.

Example: A scientific principle about sentences:

Given $\langle p \rangle = [\alpha [H \ \beta]]$,
where $\alpha = \text{edge}(\text{Spec's})$ β then:
the head H of $\langle p \rangle$ is inert after the phase is
completed, triggering no further grammatical
operations.

Language is complex!!!
Understanding how language works is hard!!!

Unless you're 3.



CS Survey: Prof. Sakas, Computational Computational Linguistics



Linguistic experts!

Challenge Problem



Linguistic experts!



Design a program that **counts** the number of plural nouns in a **list** of nouns. Think about:

- what the input is,
- what the output is, and
- how you can determine if a noun is plural.

Note: To simplify the problem, assume all plural nouns end in “s”.

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

```
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

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