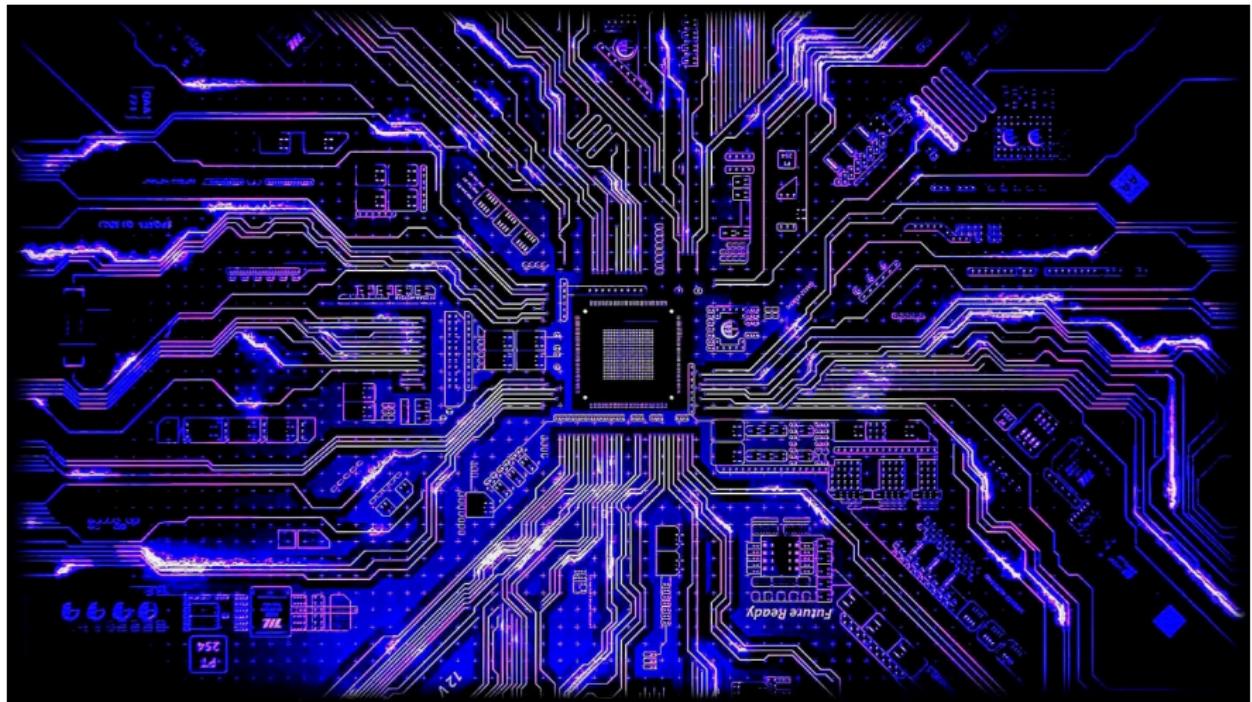


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

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 10 programming assignments
(this week we will be working on programs 1-10, which is batch 1)*

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 10 programming assignments
(this week we will be working on programs 1-10, which is batch 1)*

- **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 10 programming assignments
(this week we will be working on programs 1-10, which is batch 1)*

- **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 10 programming assignments
(this week we will be working on programs 1-10, which is batch 1)*

- **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 10 programming assignments
(this week we will be working on programs 1-10, which is batch 1)*

- **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 10 programming assignments
(this week we will be working on programs 1-10, which is batch 1)*

- **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 class quizzes, programming assignments and the final exam.

Today's Topics



- **For-loops**
- `range()`
- Variables
- Characters
- Strings

In Pairs or Triples...

Some review and some novel challenges:

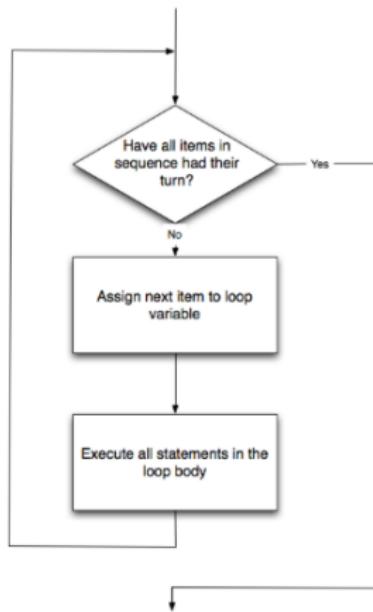
```
1 #Predict what will be printed:  
2 for i in range(4):  
3     print('The world turned upside down')  
4 for j in [0,1,2,3,4,5]:  
5     print(j)  
6 for count in range(6):  
7     print(count)  
8 for color in ['red', 'green', 'blue']:  
9     print(color)  
10    for i in range(2):  
11      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

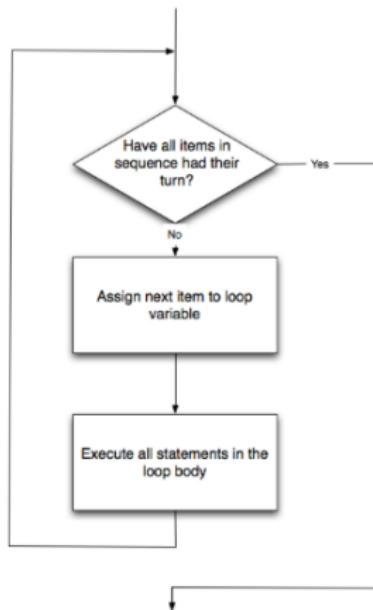
for-loop



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

How to Think Like CS, §4.5

for-loop



```
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()`.

How to Think Like CS, §4.5

Today's Topics



- For-loops
- `range()`
- Variables
- Characters
- Strings

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: $[0,1,2,3,\dots,stop-1]$
- For example, if you want the list $[0,1,2,3,\dots,100]$, you would write:

range()



Simplest version:

- `range(stop)`
- Produces a list: $[0,1,2,3,\dots,stop-1]$
- For example, if you want the list $[0,1,2,3,\dots,100]$, 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 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 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 start+k*step less than stop)
- For example, if you want 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:
 $[start, start+step, start+2*step\dots, last]$
(where last is the largest $start+k*step$ less than stop)
- For example, if you want 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

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

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	[END 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

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	Octal	Binary	Decimal	Hex	Char	Octal	Binary
0	00	\0	000	00000000	0	00	\0	000	00000000
1	01	\1	001	00000001	1	01	\1	001	00000001
2	02	\2	010	00000010	2	02	\2	010	00000010
3	03	\3	011	00000011	3	03	\3	011	00000011
4	04	\4	020	00000100	4	04	\4	020	00000100
5	05	\5	021	00000101	5	05	\5	021	00000101
6	06	\6	030	00000110	6	06	\6	030	00000110
7	07	\7	031	00000111	7	07	\7	031	00000111
8	08	\8	040	00001000	8	08	\8	040	00001000
9	09	\9	041	00001001	9	09	\9	041	00001001
10	0A	\n	050	00001010	10	0A	\n	050	00001010
11	0B	\v	051	00001011	11	0B	\v	051	00001011
12	0C	\f	060	00001100	12	0C	\f	060	00001100
13	0D	\r	061	00001101	13	0D	\r	061	00001101
14	0E	\t	070	00001110	14	0E	\t	070	00001110
15	0F	\b	071	00001111	15	0F	\b	071	00001111
16	10	\012	080	00010000	16	10	\012	080	00010000
17	11	\013	081	00010001	17	11	\013	081	00010001
18	12	\014	082	00010010	18	12	\014	082	00010010
19	13	\015	083	00010011	19	13	\015	083	00010011
20	14	\016	090	00010100	20	14	\016	090	00010100
21	15	\017	091	00010101	21	15	\017	091	00010101
22	16	\020	092	00010110	22	16	\020	092	00010110
23	17	\021	093	00010111	23	17	\021	093	00010111
24	18	\022	100	00011000	24	18	\022	100	00011000
25	19	\023	101	00011001	25	19	\023	101	00011001
26	1A	\024	102	00011010	26	1A	\024	102	00011010
27	1B	\025	103	00011011	27	1B	\025	103	00011011
28	1C	\026	110	00011100	28	1C	\026	110	00011100
29	1D	\027	111	00011101	29	1D	\027	111	00011101
30	1E	\030	120	00011110	30	1E	\030	120	00011110
31	1F	\031	121	00011111	31	1F	\031	121	00011111
32	20	\040	140	00100000	32	20	\040	140	00100000
33	21	\041	141	00100001	33	21	\041	141	00100001
34	22	\042	142	00100010	34	22	\042	142	00100010
35	23	\043	143	00100011	35	23	\043	143	00100011
36	24	\044	150	00100100	36	24	\044	150	00100100
37	25	\045	151	00100101	37	25	\045	151	00100101
38	26	\046	152	00100110	38	26	\046	152	00100110
39	27	\047	153	00100111	39	27	\047	153	00100111
40	28	\050	160	00101000	40	28	\050	160	00101000
41	29	\051	161	00101001	41	29	\051	161	00101001
42	2A	\052	162	00101010	42	2A	\052	162	00101010
43	2B	\053	163	00101011	43	2B	\053	163	00101011
44	2C	\054	170	00101100	44	2C	\054	170	00101100
45	2D	\055	171	00101101	45	2D	\055	171	00101101
46	2E	\056	172	00101110	46	2E	\056	172	00101110
47	2F	\057	173	00101111	47	2F	\057	173	00101111
48	30	\060	180	00110000	48	30	\060	180	00110000
49	31	\061	181	00110001	49	31	\061	181	00110001
50	32	\062	182	00110010	50	32	\062	182	00110010
51	33	\063	183	00110011	51	33	\063	183	00110011
52	34	\064	190	00110100	52	34	\064	190	00110100
53	35	\065	191	00110101	53	35	\065	191	00110101
54	36	\066	192	00110110	54	36	\066	192	00110110
55	37	\067	193	00110111	55	37	\067	193	00110111
56	38	\070	200	00111000	56	38	\070	200	00111000
57	39	\071	201	00111001	57	39	\071	201	00111001
58	3A	\072	202	00111010	58	3A	\072	202	00111010
59	3B	\073	203	00111011	59	3B	\073	203	00111011
60	3C	\074	210	00111100	60	3C	\074	210	00111100
61	3D	\075	211	00111101	61	3D	\075	211	00111101
62	3E	\076	212	00111110	62	3E	\076	212	00111110
63	3F	\077	213	00111111	63	3F	\077	213	00111111
64	40	\080	220	01000000	64	40	\080	220	01000000
65	41	\081	221	01000001	65	41	\081	221	01000001
66	42	\082	222	01000010	66	42	\082	222	01000010
67	43	\083	223	01000011	67	43	\083	223	01000011
68	44	\084	230	01000100	68	44	\084	230	01000100
69	45	\085	231	01000101	69	45	\085	231	01000101
70	46	\086	232	01000110	70	46	\086	232	01000110
71	47	\087	233	01000111	71	47	\087	233	01000111
72	48	\090	240	01001000	72	48	\090	240	01001000
73	49	\091	241	01001001	73	49	\091	241	01001001
74	4A	\092	242	01001010	74	4A	\092	242	01001010
75	4B	\093	243	01001011	75	4B	\093	243	01001011
76	4C	\094	250	01001100	76	4C	\094	250	01001100
77	4D	\095	251	01001101	77	4D	\095	251	01001101
78	4E	\096	252	01001110	78	4E	\096	252	01001110
79	4F	\097	253	01001111	79	4F	\097	253	01001111
80	50	\0A0	260	01010000	80	50	\0A0	260	01010000
81	51	\0A1	261	01010001	81	51	\0A1	261	01010001
82	52	\0A2	262	01010010	82	52	\0A2	262	01010010
83	53	\0A3	263	01010011	83	53	\0A3	263	01010011
84	54	\0A4	270	01010100	84	54	\0A4	270	01010100
85	55	\0A5	271	01010101	85	55	\0A5	271	01010101
86	56	\0A6	272	01010110	86	56	\0A6	272	01010110
87	57	\0A7	273	01010111	87	57	\0A7	273	01010111
88	58	\0B0	280	01011000	88	58	\0B0	280	01011000
89	59	\0B1	281	01011001	89	59	\0B1	281	01011001
90	5A	\0B2	282	01011010	90	5A	\0B2	282	01011010
91	5B	\0B3	283	01011011	91	5B	\0B3	283	01011011
92	5C	\0B4	290	01011100	92	5C	\0B4	290	01011100
93	5D	\0B5	291	01011101	93	5D	\0B5	291	01011101
94	5E	\0B6	292	01011110	94	5E	\0B6	292	01011110
95	5F	\0B7	293	01011111	95	5F	\0B7	293	01011111
96	60	\0C0	300	01100000	96	60	\0C0	300	01100000
97	61	\0C1	301	01100001	97	61	\0C1	301	01100001
98	62	\0C2	302	01100010	98	62	\0C2	302	01100010
99	63	\0C3	303	01100011	99	63	\0C3	303	01100011
100	64	\0C4	310	01100100	100	64	\0C4	310	01100100
101	65	\0C5	311	01100101	101	65	\0C5	311	01100101
102	66	\0C6	312	01100110	102	66	\0C6	312	01100110
103	67	\0C7	313	01100111	103	67	\0C7	313	01100111
104	68	\0D0	320	01101000	104	68	\0D0	320	01101000
105	69	\0D1	321	01101001	105	69	\0D1	321	01101001
106	6A	\0D2	322	01101010	106	6A	\0D2	322	01101010
107	6B	\0D3	323	01101011	107	6B	\0D3	323	01101011
108	6C	\0D4	330	01101100	108	6C	\0D4	330	01101100
109	6D	\0D5	331	01101101	109	6D	\0D5	331	01101101
110	6E	\0D6	332	01101110	110	6E	\0D6	332	01101110
111	6F	\0D7	333	01101111	111	6F	\0D7	333	01101111
112	70	\0E0	340	01110000	112	70	\0E0	340	01110000
113	71	\0E1	341	01110001	113	71	\0E1	341	01110001
114	72	\0E2	342	01110010	114	72	\0E2	342	01110010
115	73	\0E3	343	01110011	115	73	\0E3	343	01110011
116	74	\0E4	350	01110100	116	74	\0E4	350	01110100
117	75	\0E5	351	01110101	117	75	\0E5	351	01110101
118	76	\0E6	352	01110110	118	76	\0E6	352	01110110
119	77	\0E7	353	01110111	119	77	\0E7	353	01110111
120	78	\0F0	360	01111000	120	78	\0F0	360	01111000
121	79	\0F1	361	01111001	121	79	\0F1	361	01111001
122	7A	\0F2	362	01111010	122	7A	\0F2	362	01111010
123	7B	\0F3	363	01111011	123	7B	\0F3	363	01111011
124	7C	\0F4	370	01111100	124	7C	\0F4	370	01111100
125	7D	\0F5	371	01111101	125	7D	\0F5	371	01111101
126	7E	\0F6	372	01111110	126	7E	\0F6	372	01111110
127	7F	\0F7	373	01111111	127	7F	\0F7	373	01111111

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	Octal	Binary	Hex	Char	Octal	Binary	Hex	Char	Octal	Binary
0	00	\0	000	0000000000000000	00	\0	000	0000000000000000	00	\0	000	0000000000000000
1	01	\1	001	0000000000000001	01	\1	001	0000000000000001	01	\1	001	0000000000000001
2	02	\2	010	0000000000000010	02	\2	010	0000000000000010	02	\2	010	0000000000000010
3	03	\3	011	0000000000000011	03	\3	011	0000000000000011	03	\3	011	0000000000000011
4	04	\4	100	0000000000000100	04	\4	100	0000000000000100	04	\4	100	0000000000000100
5	05	\5	101	0000000000000101	05	\5	101	0000000000000101	05	\5	101	0000000000000101
6	06	\6	110	0000000000000110	06	\6	110	0000000000000110	06	\6	110	0000000000000110
7	07	\7	111	0000000000000111	07	\7	111	0000000000000111	07	\7	111	0000000000000111
8	08	\8	1000	0000000000001000	08	\8	1000	0000000000001000	08	\8	1000	0000000000001000
9	09	\9	1001	0000000000001001	09	\9	1001	0000000000001001	09	\9	1001	0000000000001001
10	0A	\a	1010	0000000000001010	0A	\a	1010	0000000000001010	0A	\a	1010	0000000000001010
11	0B	\b	1011	0000000000001011	0B	\b	1011	0000000000001011	0B	\b	1011	0000000000001011
12	0C	\c	1100	0000000000001100	0C	\c	1100	0000000000001100	0C	\c	1100	0000000000001100
13	0D	\d	1101	0000000000001101	0D	\d	1101	0000000000001101	0D	\d	1101	0000000000001101
14	0E	\e	1110	0000000000001110	0E	\e	1110	0000000000001110	0E	\e	1110	0000000000001110
15	0F	\f	1111	0000000000001111	0F	\f	1111	0000000000001111	0F	\f	1111	0000000000001111
16	10	\n	10000	0000000000010000	10	\n	10000	0000000000010000	10	\n	10000	0000000000010000
17	11	\v	10001	0000000000010001	11	\v	10001	0000000000010001	11	\v	10001	0000000000010001
18	12	\t	10010	0000000000010010	12	\t	10010	0000000000010010	12	\t	10010	0000000000010010
19	13	\r	10011	0000000000010011	13	\r	10011	0000000000010011	13	\r	10011	0000000000010011
20	14	\012	10100	0000000000010100	14	\012	10100	0000000000010100	14	\012	10100	0000000000010100
21	15	\015	10101	0000000000010101	15	\015	10101	0000000000010101	15	\015	10101	0000000000010101
22	16	\016	10110	0000000000010110	16	\016	10110	0000000000010110	16	\016	10110	0000000000010110
23	17	\017	10111	0000000000010111	17	\017	10111	0000000000010111	17	\017	10111	0000000000010111
24	18	\020	11000	0000000000011000	18	\020	11000	0000000000011000	18	\020	11000	0000000000011000
25	19	\021	11001	0000000000011001	19	\021	11001	0000000000011001	19	\021	11001	0000000000011001
26	1A	\022	11010	0000000000011010	1A	\022	11010	0000000000011010	1A	\022	11010	0000000000011010
27	1B	\023	11011	0000000000011011	1B	\023	11011	0000000000011011	1B	\023	11011	0000000000011011
28	1C	\024	11100	0000000000011100	1C	\024	11100	0000000000011100	1C	\024	11100	0000000000011100
29	1D	\025	11101	0000000000011101	1D	\025	11101	0000000000011101	1D	\025	11101	0000000000011101
30	1E	\026	11110	0000000000011110	1E	\026	11110	0000000000011110	1E	\026	11110	0000000000011110
31	1F	\027	11111	0000000000011111	1F	\027	11111	0000000000011111	1F	\027	11111	0000000000011111
32	20	\040	100000	0000000000100000	20	\040	100000	0000000000100000	20	\040	100000	0000000000100000
33	21	\041	100001	0000000000100001	21	\041	100001	0000000000100001	21	\041	100001	0000000000100001
34	22	\042	100010	0000000000100010	22	\042	100010	0000000000100010	22	\042	100010	0000000000100010
35	23	\043	100011	0000000000100011	23	\043	100011	0000000000100011	23	\043	100011	0000000000100011
36	24	\044	100100	0000000000100100	24	\044	100100	0000000000100100	24	\044	100100	0000000000100100
37	25	\045	100101	0000000000100101	25	\045	100101	0000000000100101	25	\045	100101	0000000000100101
38	26	\046	100110	0000000000100110	26	\046	100110	0000000000100110	26	\046	100110	0000000000100110
39	27	\047	100111	0000000000100111	27	\047	100111	0000000000100111	27	\047	100111	0000000000100111
40	28	\050	101000	0000000000101000	28	\050	101000	0000000000101000	28	\050	101000	0000000000101000
41	29	\051	101001	0000000000101001	29	\051	101001	0000000000101001	29	\051	101001	0000000000101001
42	2A	\052	101010	0000000000101010	2A	\052	101010	0000000000101010	2A	\052	101010	0000000000101010
43	2B	\053	101011	0000000000101011	2B	\053	101011	0000000000101011	2B	\053	101011	0000000000101011
44	2C	\054	101100	0000000000101100	2C	\054	101100	0000000000101100	2C	\054	101100	0000000000101100
45	2D	\055	101101	0000000000101101	2D	\055	101101	0000000000101101	2D	\055	101101	0000000000101101
46	2E	\056	101110	0000000000101110	2E	\056	101110	0000000000101110	2E	\056	101110	0000000000101110
47	2F	\057	101111	0000000000101111	2F	\057	101111	0000000000101111	2F	\057	101111	0000000000101111
48	30	\060	110000	0000000001000000	30	\060	110000	0000000001000000	30	\060	110000	0000000001000000
49	31	\061	110001	0000000001000001	31	\061	110001	0000000001000001	31	\061	110001	0000000001000001
50	32	\062	110010	0000000001000010	32	\062	110010	0000000001000010	32	\062	110010	0000000001000010
51	33	\063	110011	0000000001000011	33	\063	110011	0000000001000011	33	\063	110011	0000000001000011
52	34	\064	110100	0000000001000100	34	\064	110100	0000000001000100	34	\064	110100	0000000001000100
53	35	\065	110101	0000000001000101	35	\065	110101	0000000001000101	35	\065	110101	0000000001000101
54	36	\066	110110	0000000001000110	36	\066	110110	0000000001000110	36	\066	110110	0000000001000110
55	37	\067	110111	0000000001000111	37	\067	110111	0000000001000111	37	\067	110111	0000000001000111
56	38	\070	111000	0000000001001000	38	\070	111000	0000000001001000	38	\070	111000	0000000001001000
57	39	\071	111001	0000000001001001	39	\071	111001	0000000001001001	39	\071	111001	0000000001001001
58	3A	\072	111010	0000000001001010	3A	\072	111010	0000000001001010	3A	\072	111010	0000000001001010
59	3B	\073	111011	0000000001001011	3B	\073	111011	0000000001001011	3B	\073	111011	0000000001001011
60	3C	\074	111100	0000000001001100	3C	\074	111100	0000000001001100	3C	\074	111100	0000000001001100
61	3D	\075	111101	0000000001001101	3D	\075	111101	0000000001001101	3D	\075	111101	0000000001001101
62	3E	\076	111110	0000000001001110	3E	\076	111110	0000000001001110	3E	\076	111110	0000000001001110
63	3F	\077	111111	0000000001001111	3F	\077	111111	0000000001001111	3F	\077	111111	0000000001001111
64	40	\080	1000000	0000000010000000	40	\080	1000000	0000000010000000	40	\080	1000000	0000000010000000
65	41	\081	1000001	0000000010000001	41	\081	1000001	0000000010000001	41	\081	1000001	0000000010000001
66	42	\082	1000010	0000000010000010	42	\082	1000010	0000000010000010	42	\082	1000010	0000000010000010
67	43	\083	1000011	0000000010000011	43	\083	1000011	0000000010000011	43	\083	1000011	0000000010000011
68	44	\084	1000100	0000000010000100	44	\084	1000100	0000000010000100	44	\084	1000100	0000000010000100
69	45	\085	1000101	0000000010000101	45	\085	1000101	0000000010000101	45	\085	1000101	0000000010000101
70	46	\086	1000110	0000000010000110	46	\086	1000110	0000000010000110	46	\086	1000110	0000000010000110
71	47	\087	1000111	0000000010000111	47	\087	1000111	0000000010000111	47	\087	1000111	0000000010000111
72	48	\090	1001000	0000000010001000	48	\090	1001000	0000000010001000	48	\090	1001000	0000000010001000
73	49	\091	1001001	0000000010001001	49	\091	1001001	0000000010001001	49	\091	1001001	0000000010001001
74	4A	\092	1001010	0000000010001010	4A	\092	1001010	0000000010001010	4A	\092	1001010	0000000010001010
75	4B	\093	1001011	0000000010001011	4B	\093	1001011	0000000010001011	4B	\093	1001011	0000000010001011
76	4C	\094	1001100	0000000010001100	4C	\094	1001100	0000000010001100	4C	\094	1001100	0000000010001100
77	4D	\095	1001101	0000000010001101	4D	\095	1001101	0000000010001101	4D	\095	1001101	0000000010001101
78	4E	\096	1001110	0000000010001110	4E	\096	1001110	0000000010001110	4E	\096	1001110	0000000010001110
79	4F	\097	1001111	0000000010001111	4F	\097	1001111	0000000010001111	4F	\097	1001111	0000000010001111
80	50	\100	101									

- `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	Binary	Hex	Char									
	00000000	00		00000001	01		00000010	02		00000011	03	
	00000002	04		00000003	05		00000004	06		00000005	07	
	00000006	08		00000007	09		00000008	0A		00000009	0B	
	0000000A	0C		0000000B	0D		0000000C	0E		0000000D	0F	
	0000000E	10		0000000F	11		00000010	12		00000011	13	
	00000012	1C		00000013	1D		00000014	1E		00000015	1F	
	00000016	20		00000017	21		00000018	22		00000019	23	
	0000001A	24		0000001B	25		0000001C	26		0000001D	27	
	0000001E	2C		0000001F	2D		00000020	2E		00000021	2F	
	00000022	30		00000023	31		00000024	32		00000025	33	
	00000026	34		00000027	35		00000028	36		00000029	37	
	0000002A	3C		0000002B	3D		0000002C	3E		0000002D	3F	
	0000002E	40		0000002F	41		00000030	42		00000031	43	
	00000032	44		00000033	45		00000034	46		00000035	47	
	00000036	4C		00000037	4D		00000038	4E		00000039	4F	
	0000003A	50		0000003B	51		0000003C	52		0000003D	53	
	0000003E	54		0000003F	55		00000040	56		00000041	57	
	00000042	5C		00000043	5D		00000044	5E		00000045	5F	
	00000046	60		00000047	61		00000048	62		00000049	63	
	0000004A	64		0000004B	65		0000004C	66		0000004D	67	
	0000004E	6C		0000004F	6D		00000050	6E		00000051	6F	
	00000052	70		00000053	71		00000054	72		00000055	73	
	00000056	74		00000057	75		00000058	76		00000059	77	
	0000005A	7C		0000005B	7D		0000005C	7E		0000005D	7F	
	0000005E	80		0000005F	81		00000060	82		00000061	83	
	00000062	84		00000063	85		00000064	86		00000065	87	
	00000066	8C		00000067	8D		00000068	8E		00000069	8F	
	0000006A	90		0000006B	91		0000006C	92		0000006D	93	
	0000006E	94		0000006F	95		00000070	96		00000071	97	
	00000072	9C		00000073	9D		00000074	9E		00000075	9F	
	00000076	A0		00000077	A1		00000078	A2		00000079	A3	
	0000007A	A4		0000007B	A5		0000007C	A6		0000007D	A7	
	0000007E	A8		0000007F	A9		00000080	AA		00000081	AB	
	00000082	AC		00000083	AD		00000084	AE		00000085	AF	
	00000086	B0		00000087	B1		00000088	B2		00000089	B3	
	0000008A	B4		0000008B	B5		0000008C	B6		0000008D	B7	
	0000008E	B8		0000008F	B9		00000090	BA		00000091	BB	
	00000092	BC		00000093	BD		00000094	BE		00000095	BF	
	00000096	C0		00000097	C1		00000098	C2		00000099	C3	
	0000009A	C4		0000009B	C5		0000009C	C6		0000009D	C7	
	0000009E	C8		0000009F	C9		000000A0	CA		000000A1	CB	
	000000A2	CC		000000A3	CD		000000A4	CE		000000A5	CF	
	000000A6	D0		000000A7	D1		000000A8	D2		000000A9	D3	
	000000A8	D4		000000A9	D5		000000AA	D6		000000AB	D7	
	000000AC	D8		000000AD	D9		000000AE	DA		000000AF	DB	
	000000B0	E0		000000B1	E1		000000B2	E2		000000B3	E3	
	000000B4	E4		000000B5	E5		000000B6	E6		000000B7	E7	
	000000B8	E8		000000B9	E9		000000BA	EA		000000B1	EB	
	000000B2	EC		000000B3	ED		000000B4	EE		000000B5	EF	
	000000B6	F0		000000B7	F1		000000B8	F2		000000B9	F3	
	000000B4	F4		000000B5	F5		000000B6	F6		000000B7	F7	
	000000B8	F8		000000B9	F9		000000BA	FA		000000B1	FB	
	000000B2	FC		000000B3	FD		000000B4	FE		000000B5	FF	

- `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'.)

Decimal Num Char	Octal Num Char	Hex Num Char	Decimal Num Char	Octal Num Char	Hex Num Char
'\0'	'000'	'000'	'\n'	'000'	'000'
'\t'	'009'	'009'	'\r'	'00D'	'00D'
'\v'	'012'	'012'	'\f'	'014'	'014'
'\b'	'010'	'010'	'\a'	'007'	'007'
'\x00'	'000'	'000'	'\x0A'	'00A'	'00A'
'\x0D'	'00D'	'00D'	'\x0C'	'014'	'014'
'\x08'	'010'	'010'	'\x0B'	'011'	'011'
'\x09'	'009'	'009'	'\x0E'	'012'	'012'
'\x0A'	'00A'	'00A'	'\x0F'	'013'	'013'
'\x0C'	'014'	'014'	'\x10'	'015'	'015'
'\x08'	'010'	'010'	'\x11'	'016'	'016'
'\x09'	'009'	'009'	'\x12'	'017'	'017'
'\x0A'	'00A'	'00A'	'\x13'	'018'	'018'
'\x0D'	'00D'	'00D'	'\x14'	'019'	'019'
'\x08'	'010'	'010'	'\x15'	'01A'	'01A'
'\x09'	'009'	'009'	'\x16'	'01B'	'01B'
'\x0A'	'00A'	'00A'	'\x17'	'01C'	'01C'
'\x0D'	'00D'	'00D'	'\x18'	'01D'	'01D'
'\x08'	'010'	'010'	'\x19'	'01E'	'01E'
'\x09'	'009'	'009'	'\x1A'	'01F'	'01F'
'\x0A'	'00A'	'00A'	'\x1B'	'020'	'020'
'\x0D'	'00D'	'00D'	'\x1C'	'021'	'021'
'\x08'	'010'	'010'	'\x1D'	'022'	'022'
'\x09'	'009'	'009'	'\x1E'	'023'	'023'
'\x0A'	'00A'	'00A'	'\x1F'	'024'	'024'
'\x0D'	'00D'	'00D'	'\x20'	'025'	'025'
'\x08'	'010'	'010'	'\x21'	'026'	'026'
'\x09'	'009'	'009'	'\x22'	'027'	'027'
'\x0A'	'00A'	'00A'	'\x23'	'028'	'028'
'\x0D'	'00D'	'00D'	'\x24'	'029'	'029'
'\x08'	'010'	'010'	'\x25'	'02A'	'02A'
'\x09'	'009'	'009'	'\x26'	'02B'	'02B'
'\x0A'	'00A'	'00A'	'\x27'	'02C'	'02C'
'\x0D'	'00D'	'00D'	'\x28'	'02D'	'02D'
'\x08'	'010'	'010'	'\x29'	'02E'	'02E'
'\x09'	'009'	'009'	'\x2A'	'02F'	'02F'
'\x0A'	'00A'	'00A'	'\x2B'	'030'	'030'
'\x0D'	'00D'	'00D'	'\x2C'	'031'	'031'
'\x08'	'010'	'010'	'\x2D'	'032'	'032'
'\x09'	'009'	'009'	'\x2E'	'033'	'033'
'\x0A'	'00A'	'00A'	'\x2F'	'034'	'034'
'\x0D'	'00D'	'00D'	'\x30'	'035'	'035'
'\x08'	'010'	'010'	'\x31'	'036'	'036'
'\x09'	'009'	'009'	'\x32'	'037'	'037'
'\x0A'	'00A'	'00A'	'\x33'	'038'	'038'
'\x0D'	'00D'	'00D'	'\x34'	'039'	'039'
'\x08'	'010'	'010'	'\x35'	'03A'	'03A'
'\x09'	'009'	'009'	'\x36'	'03B'	'03B'
'\x0A'	'00A'	'00A'	'\x37'	'03C'	'03C'
'\x0D'	'00D'	'00D'	'\x38'	'03D'	'03D'
'\x08'	'010'	'010'	'\x39'	'03E'	'03E'
'\x09'	'009'	'009'	'\x3A'	'03F'	'03F'
'\x0A'	'00A'	'00A'	'\x3B'	'040'	'040'
'\x0D'	'00D'	'00D'	'\x3C'	'041'	'041'
'\x08'	'010'	'010'	'\x3D'	'042'	'042'
'\x09'	'009'	'009'	'\x3E'	'043'	'043'
'\x0A'	'00A'	'00A'	'\x3F'	'044'	'044'
'\x0D'	'00D'	'00D'	'\x40'	'045'	'045'
'\x08'	'010'	'010'	'\x41'	'046'	'046'
'\x09'	'009'	'009'	'\x42'	'047'	'047'
'\x0A'	'00A'	'00A'	'\x43'	'048'	'048'
'\x0D'	'00D'	'00D'	'\x44'	'049'	'049'
'\x08'	'010'	'010'	'\x45'	'04A'	'04A'
'\x09'	'009'	'009'	'\x46'	'04B'	'04B'
'\x0A'	'00A'	'00A'	'\x47'	'04C'	'04C'
'\x0D'	'00D'	'00D'	'\x48'	'04D'	'04D'
'\x08'	'010'	'010'	'\x49'	'04E'	'04E'
'\x09'	'009'	'009'	'\x4A'	'04F'	'04F'
'\x0A'	'00A'	'00A'	'\x4B'	'050'	'050'
'\x0D'	'00D'	'00D'	'\x4C'	'051'	'051'
'\x08'	'010'	'010'	'\x4D'	'052'	'052'
'\x09'	'009'	'009'	'\x4E'	'053'	'053'
'\x0A'	'00A'	'00A'	'\x4F'	'054'	'054'
'\x0D'	'00D'	'00D'	'\x50'	'055'	'055'
'\x08'	'010'	'010'	'\x51'	'056'	'056'
'\x09'	'009'	'009'	'\x52'	'057'	'057'
'\x0A'	'00A'	'00A'	'\x53'	'058'	'058'
'\x0D'	'00D'	'00D'	'\x54'	'059'	'059'
'\x08'	'010'	'010'	'\x55'	'05A'	'05A'
'\x09'	'009'	'009'	'\x56'	'05B'	'05B'
'\x0A'	'00A'	'00A'	'\x57'	'05C'	'05C'
'\x0D'	'00D'	'00D'	'\x58'	'05D'	'05D'
'\x08'	'010'	'010'	'\x59'	'05E'	'05E'
'\x09'	'009'	'009'	'\x5A'	'05F'	'05F'
'\x0A'	'00A'	'00A'	'\x5B'	'060'	'060'
'\x0D'	'00D'	'00D'	'\x5C'	'061'	'061'
'\x08'	'010'	'010'	'\x5D'	'062'	'062'
'\x09'	'009'	'009'	'\x5E'	'063'	'063'
'\x0A'	'00A'	'00A'	'\x5F'	'064'	'064'
'\x0D'	'00D'	'00D'	'\x60'	'065'	'065'
'\x08'	'010'	'010'	'\x61'	'066'	'066'
'\x09'	'009'	'009'	'\x62'	'067'	'067'
'\x0A'	'00A'	'00A'	'\x63'	'068'	'068'
'\x0D'	'00D'	'00D'	'\x64'	'069'	'069'
'\x08'	'010'	'010'	'\x65'	'06A'	'06A'
'\x09'	'009'	'009'	'\x66'	'06B'	'06B'
'\x0A'	'00A'	'00A'	'\x67'	'06C'	'06C'
'\x0D'	'00D'	'00D'	'\x68'	'06D'	'06D'
'\x08'	'010'	'010'	'\x69'	'06E'	'06E'
'\x09'	'009'	'009'	'\x6A'	'06F'	'06F'
'\x0A'	'00A'	'00A'	'\x6B'	'070'	'070'
'\x0D'	'00D'	'00D'	'\x6C'	'071'	'071'
'\x08'	'010'	'010'	'\x6D'	'072'	'072'
'\x09'	'009'	'009'	'\x6E'	'073'	'073'
'\x0A'	'00A'	'00A'	'\x6F'	'074'	'074'
'\x0D'	'00D'	'00D'	'\x70'	'075'	'075'
'\x08'	'010'	'010'	'\x71'	'076'	'076'
'\x09'	'009'	'009'	'\x72'	'077'	'077'
'\x0A'	'00A'	'00A'	'\x73'	'078'	'078'
'\x0D'	'00D'	'00D'	'\x74'	'079'	'079'
'\x08'	'010'	'010'	'\x75'	'07A'	'07A'
'\x09'	'009'	'009'	'\x76'	'07B'	'07B'
'\x0A'	'00A'	'00A'	'\x77'	'07C'	'07C'
'\x0D'	'00D'	'00D'	'\x78'	'07D'	'07D'
'\x08'	'010'	'010'	'\x79'	'07E'	'07E'
'\x09'	'009'	'009'	'\x7A'	'07F'	'07F'
'\x0A'	'00A'	'00A'	'\x7B'	'080'	'080'
'\x0D'	'00D'	'00D'	'\x7C'	'081'	'081'
'\x08'	'010'	'010'	'\x7D'	'082'	'082'
'\x09'	'009'	'009'	'\x7E'	'083'	'083'
'\x0A'	'00A'	'00A'	'\x7F'	'084'	'084'
'\x0D'	'00D'	'00D'	'\x80'	'085'	'085'
'\x08'	'010'	'010'	'\x81'	'086'	'086'
'\x09'	'009'	'009'	'\x82'	'087'	'087'
'\x0A'	'00A'	'00A'	'\x83'	'088'	'088'
'\x0D'	'00D'	'00D'	'\x84'	'089'	'089'
'\x08'	'010'	'010'	'\x85'	'08A'	'08A'
'\x09'	'009'	'009'	'\x86'	'08B'	'08B'
'\x0A'	'00A'	'00A'	'\x87'	'08C'	'08C'
'\x0D'	'00D'	'00D'	'\x88'	'08D'	'08D'
'\x08'	'010'	'010'	'\x89'	'08E'	'08E'
'\x09'	'009'	'009'	'\x8A'	'08F'	'08F'
'\x0A'	'00A'	'00A'	'\x8B'	'090'	'090'
'\x0D'	'00D'	'00D'	'\x8C'	'091'	'091'
'\x08'	'010'	'010'	'\x8D'	'092'	'092'
'\x09'	'009'	'009'	'\x8E'	'093'	'093'
'\x0A'	'00A'	'00A'	'\x8F'	'094'	'094'
'\x0D'	'00D'	'00D'	'\x90'	'095'	'095'
'\x08'	'010'	'010'	'\x91'	'096'	'096'
'\x09'	'009'	'009'	'\x92'	'097'	'097'
'\x0A'	'00A'	'00A'	'\x93'	'098'	'098'
'\x0D'	'00D'	'00D'	'\x94'	'099'	'099'
'\x08'	'010'	'010'	'\x95'	'0A0'	'0A0'
'\x09'	'009'	'009'	'\x96'	'0A1'	'0A1'
'\x0A'	'00A'	'00A'	'\x97'	'0A2'	'0A2'
'\x0D'	'00D'	'00D'	'\x98'	'0A3'	'0A3'
'\x08'	'010'	'010'	'\x99'	'0A4'	'0A4'
'\x09'	'009'	'009'	'\x9A'	'0A5'	'0A5'
'\x0A'	'00A'	'00A'	'\x9B'	'0A6'	'0A6'
'\x0D'	'00D'	'00D'	'\x9C'	'0A7'	'0A7'
'\x08'	'010'	'010'	'\x9D'	'0A8'	'0A8'
'\x09'	'009'	'009'	'\x9E'	'0A9'	'0A9'
'\x0A'	'00A'	'00A'	'\x9F'	'0AA'	'0AA'
'\x0D'	'00D'	'00D'	'\xA0'	'0AB'	'0AB'
'\x08'	'010'	'010'	'\xA1'	'0AC'	'0AC'
'\x09'	'009'	'009'	'\xA2'	'0AD'	'0AD'
'\x0A'	'00A'	'00A'	'\xA3'	'0AE'	'0AE'
'\x0D'	'00D'	'00D'	'\xA4'	'0AF'	'0AF'
'\x08'	'010'	'010'	'\xA5'	'0B0'	'0B0'
'\x09'	'009'	'009'	'\xA6'	'0B1'	'0B1'
'\x0A'	'00A'	'00A'	'\xA7'	'0B2'	'0B2'
'\x0D'	'00D'	'00D'	'\xA8'	'0B3'	'0B3'
'\x08'	'010'	'010'	'\xA9'	'0B4'	'0B4'
'\x09'	'009'	'009'	'\xAA'	'0B5'	'0B5'
'\x0A'	'00A'	'00A'	'\xAB'	'0B6'	'0B6'
'\x0D'	'00D'	'00D'	'\xAC'	'0B7'	'0B7'
'\x08'	'010'	'010'	'\xAD'	'0B8'	'0B8'
'\x09'	'009'	'009'	'\xAE'	'0B9'	'0B9'
'\x0A'	'00A'	'00A'	'\xAF'	'0BA'	'0BA'
'\x0D'	'00D'	'00D'	'\xA0'	'0B10'	'0B10'
'\x08'	'010'	'010'	'\xA1'	'0B11'	'0B11'
'\x09'	'009'	'009'	'\xA2'	'0B12'	'0B12'
'\x0A'	'00A'	'00A'	'\xA3'	'0B13'	'0B13'
'\x0D'	'00D'	'00D'	'\xA4'	'0B14'	'0B14'
'\x08'	'010'	'010'	'\xA5'	'0B15'	'0B15'
'\x09'	'009'	'009'	'\xA6'	'0B16'	'0B16'
'\x0A'	'00A'	'00A'	'\xA7'	'0B17'	'0B17'
'\x0D'	'00D'	'00D'	'\xA8'	'0B18'	'0B18'
'\x08'	'010'	'010'	'\xA9'	'0B19'	'0B19'
'\x09'	'009'	'009'	'\xAA'	'0B1A'	'0B1A'
'\x0A'	'00A'	'00A'	'\xAB'	'0B1B'	'0B1B'
'\x0D'	'00D'	'00D'	'\xA0'	'0B1C'	'0B1C'
'\x08'	'010'	'010'	'\xA1'	'0B1D'	'0B1D'
'\x09'	'009'	'009'	'\xA2'	'0B1E'	'0B1E'
'\x0A'	'00A'	'00A'	'\xA3'	'0B1F'	'0B1F'
'\x0D'	'00D'	'00D'	'\xA4'	'0B20'	'0B20'
'\x08'	'010'	'010'	'\xA5'	'0B21'	'0B21'
'\x09'	'009'	'009'	'\xA6'	'0B22'	'0B22'
'\x0A'	'00A'	'00A'	'\xA7'	'0B23'	'0B23'
'\x0D'	'00D'	'00D'	'\xA8'	'0B24'	'0B24'
'\x08'	'010'	'010'	'\xA9'	'0B25'	'0B25'
'\x09'	'009'	'009'	'\xA0'	'0B26'	'0B26'
'\x0A'	'00A'	'00A'	'\xA1'	'0B27'	'0B27'
'\x0D'	'00D'	'00D'	'\xA2'	'0B28'	'0B28'
'\x08'	'010'	'010'	'\xA3'	'0B29'	'0B29'
'\x09'	'009'	'009'	'\xA4'	'0B2A'	'0B2A'
'\x0A'	'00A'	'00A'	'\xA5'	'0B2B'	'0B2B'
'\x0D'	'00D'	'00D'	'\xA6'	'0B2C'	'0B2C'
'\x08'	'010'	'010'	'\xA7'	'0B2D'	'0B2D'
'\x09'	'009'	'009'	'\xA8'	'0B2E'	'0B2E'
'\x0A'	'00A'	'00A'	'\xA9'	'0B2F'	'0B2F'
'\x0D'	'00D'	'00D'	'\xA0'	'0B30'	'0B30'
'\x08'	'010'	'010'	'\xA1'	'0B31'	'0B31'
'\x09'	'009'	'009'	'\xA2'	'0B32'	'0B32'
'\x0A'	'00A'	'00A'	'\xA3'	'0B33'	'0B33'
'\x0D'	'00D'	'00D'	'\xA4'	'0B34'	'0B34'
'\x08'	'010'	'010'	'\xA5'	'0B35'	'0B35'
'\x09'	'009'	'009'	'\xA6'	'0B36'	'0B36'
'\x0A'	'00A'	'00A'	'\xA7'	'0B37'	'0B37'
'\x0D'	'00D'	'00D'	'\xA8'	'0B38'	'0B38'
'\x08'	'010'	'010'	'\xA9'	'0B39'	'0B39'
'\x09'	'009'	'009'	'\xA0'	'0B3A'	'0B3A'
'\x0A'	'00A'	'00A'	'\xA1'	'0B3B'	'0B3B'
'\x0D'	'00D'	'00D'	'\xA2'	'0B3C'	'0B3C'
'\x08'	'010'	'010'	'\xA3'	'0B3D'	'0B3D'
'\x09'	'009'	'009'	'\xA4'	'0B3E'	'0B3E'
'\x0A'	'00A'	'00A'	'\xA5'	'0B3F'	'0B3F'
'\x0D'	'00D'	'00D'	'\xA6'	'0B40'	'0B40'
'\x08'	'010'	'010'	'\xA7'	'0B41'	'0B41'
'\x09'	'009'	'009'	'\xA8'	'0B42'	'0B42'
'\x0A'	'00A'	'00A'	'\xA9'	'0B43'	'0B43'
'\x0					

Converting from Character to Code:

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

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	00	\0	32	20		64	40	!
1	01	\1	33	21	!	65	41	A
2	02	\2	34	22	”	66	42	B
3	03	\3	35	23	”	67	43	C
4	04	\4	36	24	”	68	44	D
5	05	\5	37	25	”	69	45	E
6	06	\6	38	26	”	70	46	F
7	07	\7	39	27	”	71	47	G
8	08	\8	40	28	”	72	48	H
9	09	\9	41	29	”	73	49	I
10	0A	\n	42	2A	”	74	4A	J
11	0B	\r	43	2B	”	75	4B	K
12	0C	\t	44	2C	”	76	4C	L
13	0D	\v	45	2D	”	77	4D	M
14	0E	\f	46	2E	”	78	4E	N
15	0F	\u000F	47	2F	”	79	4F	O
16	10	\u0010	48	30	”	80	50	P
17	11	\u0011	49	31	”	81	51	Q
18	12	\u0012	4A	32	”	82	52	R
19	13	\u0013	4B	33	”	83	53	S
20	14	\u0014	4C	34	”	84	54	T
21	15	\u0015	4D	35	”	85	55	U
22	16	\u0016	4E	36	”	86	56	V
23	17	\u0017	4F	37	”	87	57	W
24	18	\u0018	50	38	”	88	58	X
25	19	\u0019	51	39	”	89	59	Y
26	1A	\u001A	52	3A	”	90	5A	Z
27	1B	\u001B	53	3B	”	91	5B	[\u001B]
28	1C	\u001C	54	3C	”	92	5C	\u001C
29	1D	\u001D	55	3D	”	93	5D	\u001D
30	1E	\u001E	56	3E	”	94	5E	\u001E
31	1F	\u001F	57	3F	”	95	5F	\u001F
32	20	\u0020	58	40		96	60	\u0020
33	21	\u0021	59	41	!	97	61	'
34	22	\u0022	60	42	"	98	62	"
35	23	\u0023	61	43	#	99	63	%
36	24	\u0024	62	44	\$	100	64	@
37	25	\u0025	63	45	\u0025	101	65	\u0025
38	26	\u0026	64	46	\u0026	102	66	\u0026
39	27	\u0027	65	47	\u0027	103	67	\u0027
40	28	\u0028	66	48	\u0028	104	68	\u0028
41	29	\u0029	67	49	\u0029	105	69	\u0029
42	2A	\u002A	68	4A	*	106	6A	*
43	2B	\u002B	69	4B	+	107	6B	+
44	2C	\u002C	70	4C	,	108	6C	,
45	2D	\u002D	71	4D	-	109	6D	-
46	2E	\u002E	72	4E	.	110	6E	.
47	2F	\u002F	73	4F	/	111	6F	/
48	30	\u0030	74	50	0	112	70	0
49	31	\u0031	75	51	1	113	71	1
50	32	\u0032	76	52	2	114	72	2
51	33	\u0033	77	53	3	115	73	3
52	34	\u0034	78	54	4	116	74	4
53	35	\u0035	79	55	5	117	75	5
54	36	\u0036	80	56	6	118	76	6
55	37	\u0037	81	57	7	119	77	7
56	38	\u0038	82	58	8	120	78	8
57	39	\u0039	83	59	9	121	79	9
58	3A	\u003A	84	5A	:	122	7A	:
59	3B	\u003B	85	5B	>	123	7B	>
60	3C	\u003C	86	5C	<	124	7C	<
61	3D	\u003D	87	5D	=	125	7D	=
62	3E	\u003E	88	5E	>=	126	7E	>=
63	3F	\u003F	89	5F	<=	127	7F	<=
64	40	\u0040	90	60	\u0040	128	80	\u0040
65	41	\u0041	91	61	A	129	81	A
66	42	\u0042	92	62	B	130	82	B
67	43	\u0043	93	63	C	131	83	C
68	44	\u0044	94	64	D	132	84	D
69	45	\u0045	95	65	E	133	85	E
70	46	\u0046	96	66	F	134	86	F
71	47	\u0047	97	67	G	135	87	G
72	48	\u0048	98	68	H	136	88	H
73	49	\u0049	99	69	I	137	89	I
74	4A	\u004A	100	6A	J	138	90	J
75	4B	\u004B	101	6B	K	139	91	K
76	4C	\u004C	102	6C	L	140	92	L
77	4D	\u004D	103	6D	M	141	93	M
78	4E	\u004E	104	6E	N	142	94	N
79	4F	\u004F	105	6F	O	143	95	O
80	50	\u0050	106	70	P	144	96	P
81	51	\u0051	107	71	Q	145	97	Q
82	52	\u0052	108	72	R	146	98	R
83	53	\u0053	109	73	S	147	99	S
84	54	\u0054	110	74	T	148	100	T
85	55	\u0055	111	75	U	149	101	U
86	56	\u0056	112	76	V	150	102	V
87	57	\u0057	113	77	W	151	103	W
88	58	\u0058	114	78	X	152	104	X
89	59	\u0059	115	79	Y	153	105	Y
90	5A	\u005A	116	7A	Z	154	106	Z
91	5B	\u005B	117	7B	[\u005B]	155	107	[\u005B]
92	5C	\u005C	118	7C	\u005C	156	108	\u005C
93	5D	\u005D	119	7D	\u005D	157	109	\u005D
94	5E	\u005E	120	7E	\u005E	158	110	\u005E
95	5F	\u005F	121	7F	\u005F	159	111	\u005F

- `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:

.../images/csci127/caesarCipher.png

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 the offset is 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     print(newChar, codedWord + newChar) #add the newChar to the coded w  
22  
23 print("The coded word (with wrap) is", codedWord)
```

(Demo with pythonTutor)

Wrap

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



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

User Input

Covered in detail in Lab 2:

./images/csci127/inputMessage.png

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**

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~~X~~Saturday~~X~~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.

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

- Different delimiters give different lists:

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

```
"FriXXXsSaturdayXXXsSunday"
```

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")
```

```
"FriXXXySaturXXXySunXXX"  
days = ['Fri', 'sSatur', 'sSun']
```

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

```
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

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()`

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

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

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

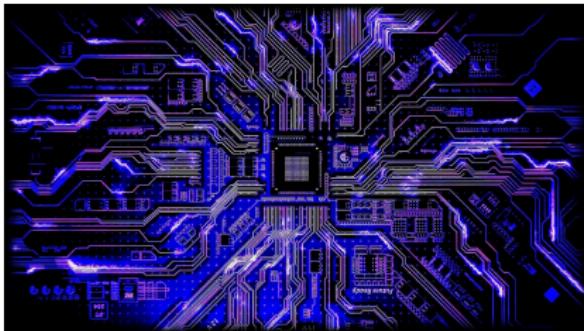
Recap

```
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

Weekly Reminders!



Before next lecture, don't forget to:

- Read and work through LAB 2!
- Submit this week's 10 programming assignments (programs 1-10)
- If you need help, email cscisummer24@gmail.com with questions or to sign up for Friday online tutoring.