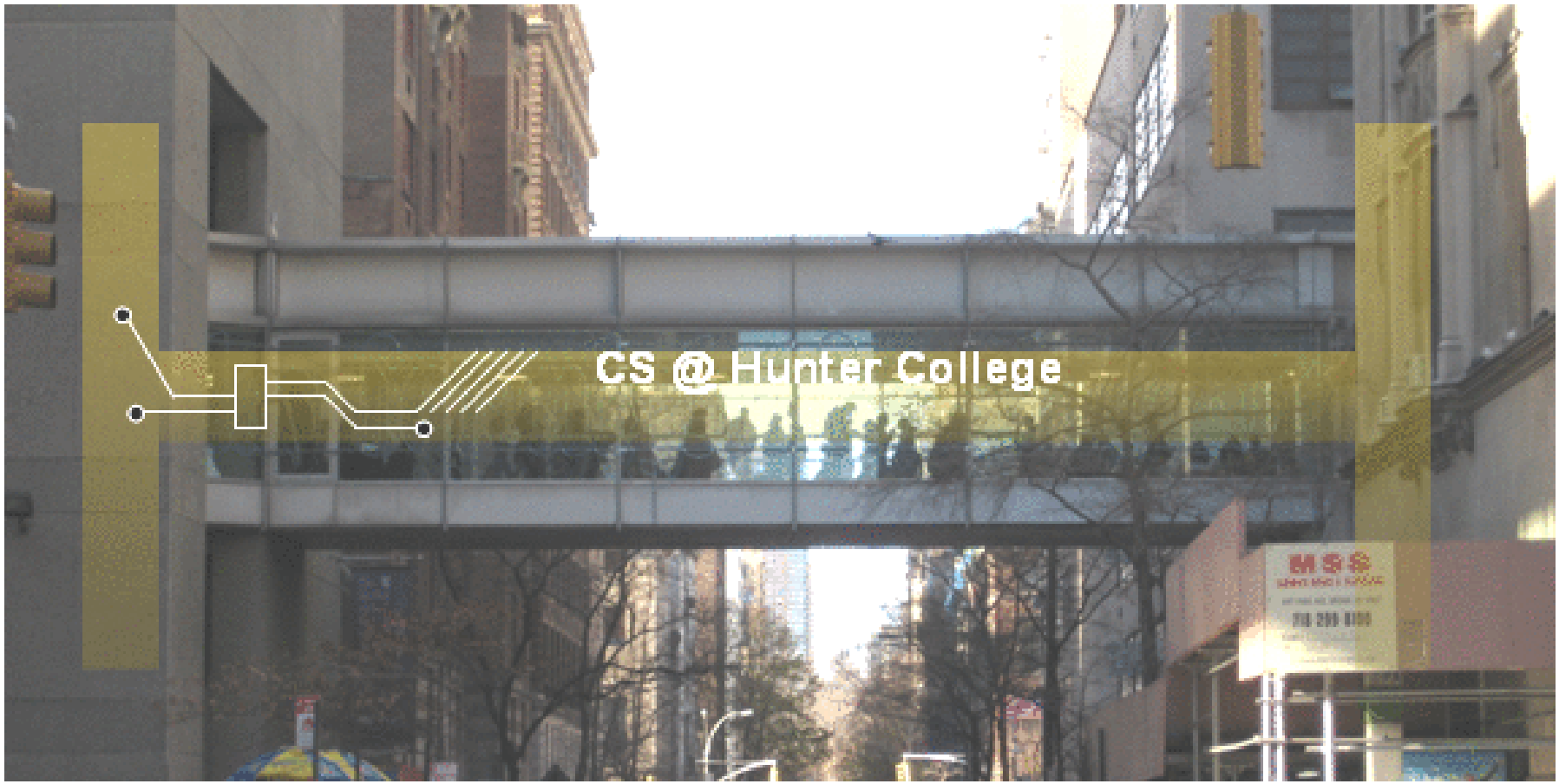
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

- This lecture will be recorded

Frequently Asked Questions

From email

Frequently Asked Questions

From email

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

Frequently Asked Questions

From email

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

Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**
  You don't submit the lab, you **read the lab.**
  When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)

Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**
  You don't submit the lab, you **read the lab.**
  When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)
- **Can I work ahead?**

Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**
  You don't submit the lab, you **read the lab.**
  When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)
- **Can I work ahead?**
  Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.

Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**
  You don't submit the lab, you **read the lab.**
  When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)
- **Can I work ahead?**
  Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.
  IMPORTANT: Students who work on the due dates in this class tend to miss deadlines and fall behind. If, instead, you work on programs the week of the associated lecture, you will have time to ask for help if you get stuck and still make the deadline.

Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**
  You don't submit the lab, you **read the lab.**
  When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)
- **Can I work ahead?**
  Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.
  IMPORTANT: Students who work on the due dates in this class tend to miss deadlines and fall behind. If, instead, you work on programs the week of the associated lecture, you will have time to ask for help if you get stuck and still make the deadline.
- **When is the midterm?**

Frequently Asked Questions

From email

- **I am not sure how to submit the Lab.**
  You don't submit the lab, you **read the lab.**
  When you are done, start working on this week's 5 programming assignments (this week we will be working on programs 6-10)
- **Can I work ahead?**
  Absolutely! Submission is open on Gradescope, 3 weeks before the deadline.
  IMPORTANT: Students who work on the due dates in this class tend to miss deadlines and fall behind. If, instead, you work on programs the week of the associated lecture, you will have time to ask for help if you get stuck and still make the deadline.
- **When is the midterm?**
  There is no midterm. Instead there's required weekly quizzes, code reviews and programming assignments.

Today's Topics

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

In Pairs or Triples...

Some review and some novel challenges:

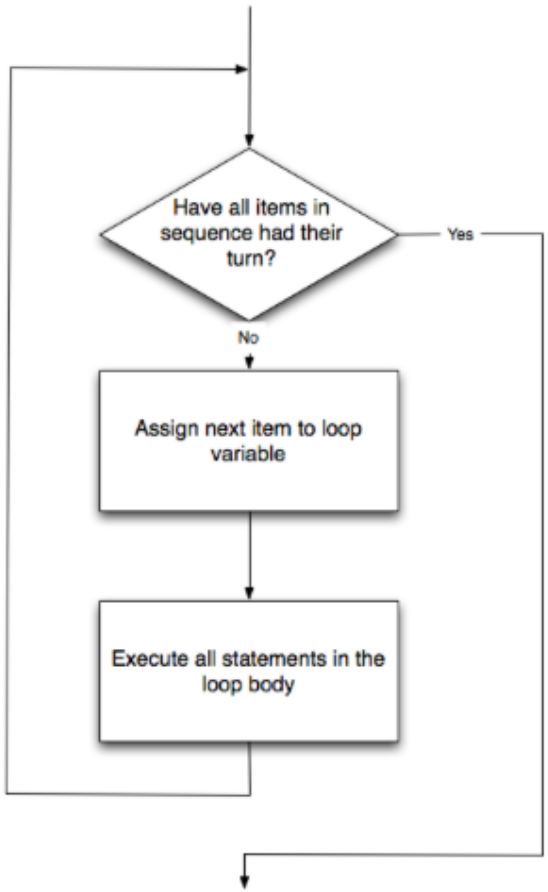
```
1  #Predict what will be printed:
2  for i in range(4):
3      print('The world turned upside down')
4  for j in [0,1,2,3,4,5]:
5      print(j)
6  for count in range(6):
7      print(count)
8  for color in ['red', 'green', 'blue']:
9      print(color)
10 for i in range(2):
11     for j in range(2):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

Python Tutor

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

(Demo with pythonTutor)

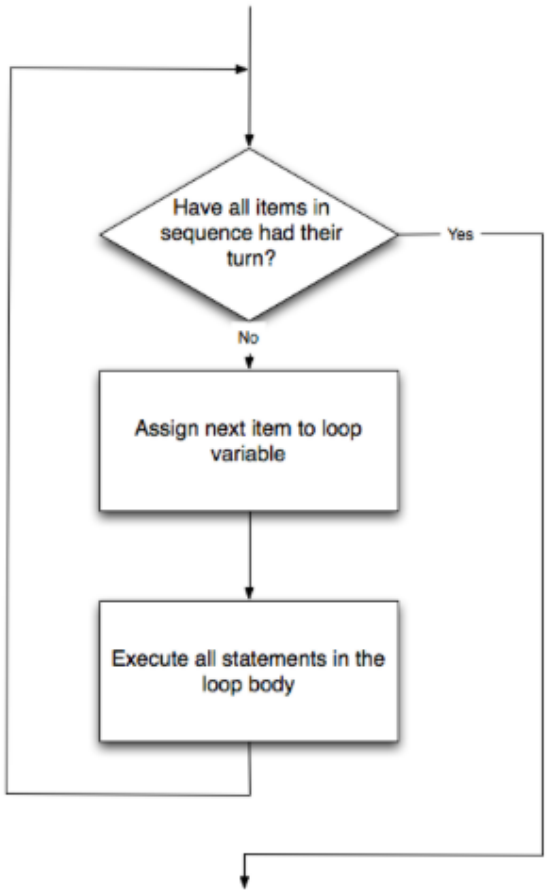
for-loop



How to Think Like CS, §4.5

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

for-loop



How to Think Like CS, §4.5

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

where `list` is a list of items:

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

Today's Topics

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

More on `range()`:

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

Python Tutor

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

(Demo with pythonTutor)

`range()`

Simplest version:

- `range(stop)`

`range()`

Simplest version:

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

range()

Simplest version:

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

range()

Simplest version:

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

```
range(101)
```

`range()`

What if you wanted to start somewhere else:

`range()`

What if you wanted to start somewhere else:

- `range(start, stop)`

`range()`

What if you wanted to start somewhere else:

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

`range()`

What if you wanted to start somewhere else:

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

range()

What if you wanted to start somewhere else:

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

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

`range()`

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

range()

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

- `range(start, stop, step)`

range()

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

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

range()

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

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

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

In summary: range()

The three versions:

In summary: range()

The three versions:

- `range(stop)`

In summary: range()

The three versions:

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

In summary: range()

The three versions:

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

Today's Topics

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

Variables

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

Variables

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

Variables

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

Variables

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

Variables

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

Variables

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

Variables

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

Variable Names

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

Variable Names

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

Variable Names

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

Variable Names

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

Variable Names

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

Today's Topics

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

Standardized Code for Characters

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

Standardized Code for Characters

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

(New version called: Unicode).

Standardized Code for Characters

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

(New version called: Unicode).

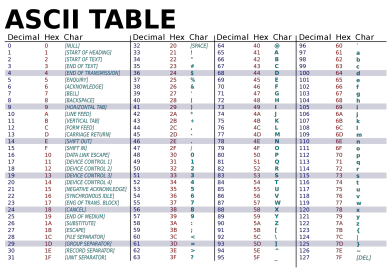
ASCII TABLE

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

(wiki)

Converting from Character to Code:

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

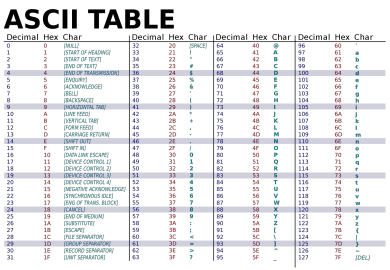


ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	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]

Converting from Character to Code:

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

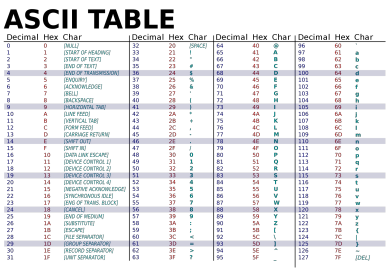


Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	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]

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

Converting from Character to Code:

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



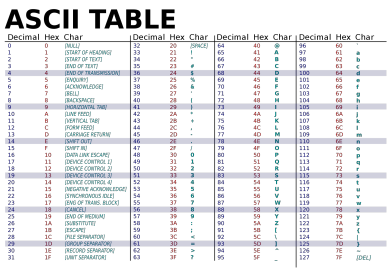
ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	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]

- `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'.)

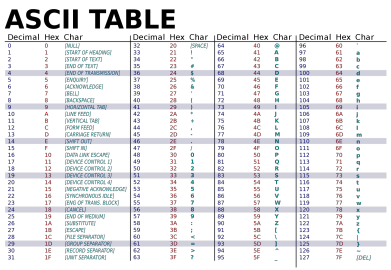


Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	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]

- `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:

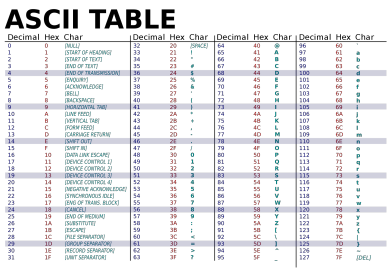
(There is a link to the ASCII table on the course webpage, under 'Useful Links'.)



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

Converting from Character to Code:

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



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]

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

In Pairs or Triples...

Some review and some novel challenges:

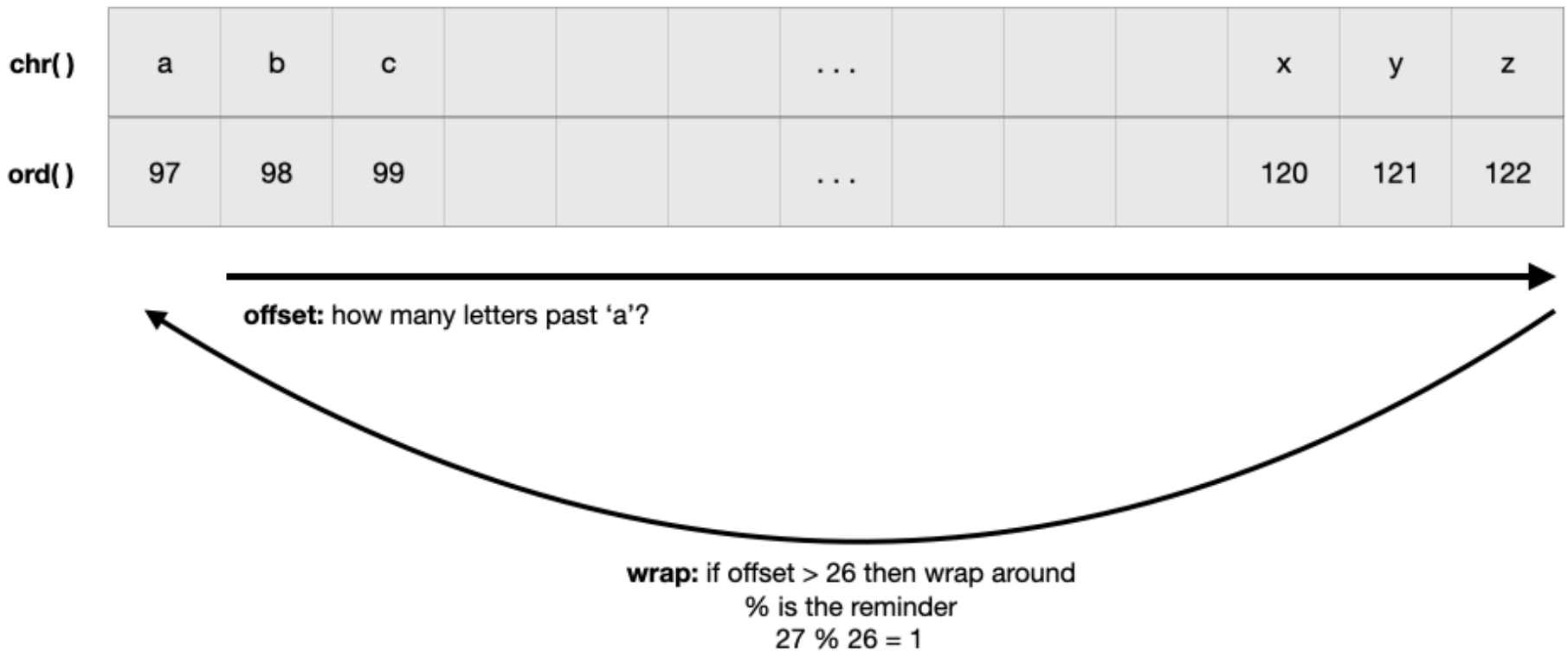
```
1  #Predict what will be printed:
2
3  for c in range(65,90):
4      print(chr(c))
5
6  message = "I love Python"
7  newMessage = ""
8  for c in message:
9      print(ord(c))   #Print the Unicode of each number
10     print(chr(ord(c)+1))    #Print the next character
11     newMessage = newMessage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

Python Tutor

```
#Predict what will be printed:

for c in range(65,90):
    print(chr(c))

message = "I love Python"
newMessage = ""
for c in message:
    print(ord(c))   #Print the Unicode of each number
    print(chr(ord(c)+1))    #Print the next character
    newMessage = newMessage + chr(ord(c)+1) #add to the new message
print("The coded message is", newMessage)

word = "zebra"
codedWord = ""
for ch in word:
    offset = ord(ch) - ord('a') + 1 #how many letters past 'a'
    wrap = offset % 26  #if larger than 26, wrap back to 0
    newChar = chr(ord('a') + wrap)  #compute the new letter
    print(wrap, chr(ord('a') + wrap))    #print the wrap & new lett
    codedWord = codedWord + newChar #add the newChar to the coded w

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
offset: how many letters past ‘a’?
wrap: if offset > 26 then wrap around
% is the reminder
27 % 26 = 1

User Input

Covered in detail in Lab 2:

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

(Demo with `pythonTutor`)

Side Note: '+' for numbers and strings

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

Side Note: '+' for numbers and strings

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

Side Note: '+' for numbers and strings

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

Side Note: '+' for numbers and strings

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

Lecture Quiz

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

Today's Topics

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

More on Strings: String Methods

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

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

More on Strings: String Methods

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

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

More on Strings: String Methods

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

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

More on Strings: String Methods

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

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

More on Strings: String Methods

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

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

More on Strings: String Methods

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

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

More on Strings: String Methods

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

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

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

More on Strings: Indexing & Substrings

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

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

More on Strings: Indexing & Substrings

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

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

More on Strings: Indexing & Substrings

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

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

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

More on Strings: Indexing & Substrings

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

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

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

More on Strings: Indexing & Substrings

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

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

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

- `s[0]` is

More on Strings: Indexing & Substrings

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

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

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

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

More on Strings: Indexing & Substrings

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

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

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

- `s[1]` is

More on Strings: Indexing & Substrings

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

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

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

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

More on Strings: Indexing & Substrings

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

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

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

- `s[-1]` is

More on Strings: Indexing & Substrings

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

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

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

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

More on Strings: Indexing & Substrings

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

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

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

- `s[3:6]` is

More on Strings: Indexing & Substrings

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

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

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

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

More on Strings: Indexing & Substrings

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

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

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

- `s[:3]` is

More on Strings: Indexing & Substrings

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

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

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

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

More on Strings: Indexing & Substrings

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

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

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

- `s[:-1]` is

More on Strings: Indexing & Substrings

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

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

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

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

More on Strings: Splits

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

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

More on Strings: Splits

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

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

More on Strings: Splits

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

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

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

More on Strings: Splits

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

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

```
"FridaysSaturdaysSunday"
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.

```
"FridaysSaturdaysSunday"
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.

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

- Different delimiters give different lists:

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

More on Strings: Splits

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

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

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

- Different delimiters give different lists:

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

Today's Topics

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

Guest Speakers

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

Recap

- In Python, we introduced:

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

Recap

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

- In Python, we introduced:
  - For-loops

Recap

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

- In Python, we introduced:
  - For-loops
  - `range()`

Recap

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

- In Python, we introduced:
  - For-loops
  - `range()`
  - Variables: ints and strings

Recap

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

- In Python, we introduced:
  - For-loops
  - `range()`
  - Variables: ints and strings
  - Some arithmetic

Recap

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

- In Python, we introduced:
  - For-loops
  - `range()`
  - Variables: ints and strings
  - Some arithmetic
  - String concatenation

Recap

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

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

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

- In Python, we introduced:
  - For-loops
  - `range()`
  - Variables: ints and strings
  - Some arithmetic
  - String concatenation
  - Functions: `ord()` and `chr()`
  - String Manipulation

Practice Quiz & Final Questions



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

Practice Quiz & Final Questions



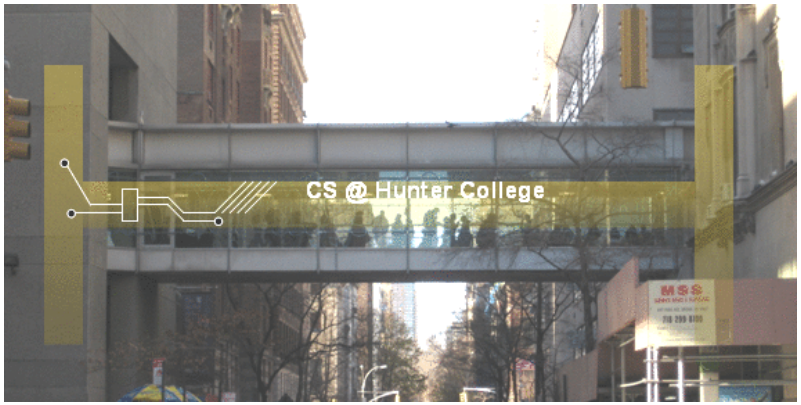
- Since you must pass the final exam to pass the course, we end every lecture with final exam review.
- Pull out something to write on (not to be turned in).
- Lightning rounds:
  - write as much you can for 60 seconds;
  - followed by answer; and
  - repeat.
- Past exams are on the webpage (under Final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

Weekly Reminders!

Before next lecture, don't forget to:

- Work on this week's Online Lab

Weekly Reminders!



Before next lecture, don't forget to:

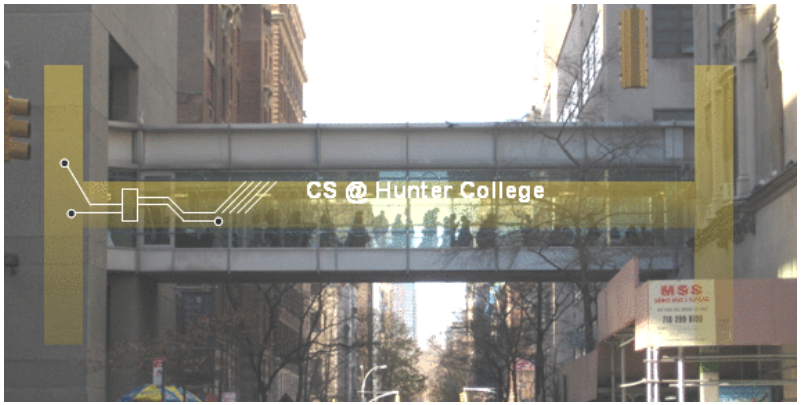
- Work on this week's Online Lab
- Schedule an appointment to take the Quiz in lab 1001E Hunter North

Weekly Reminders!

Before next lecture, don't forget to:

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

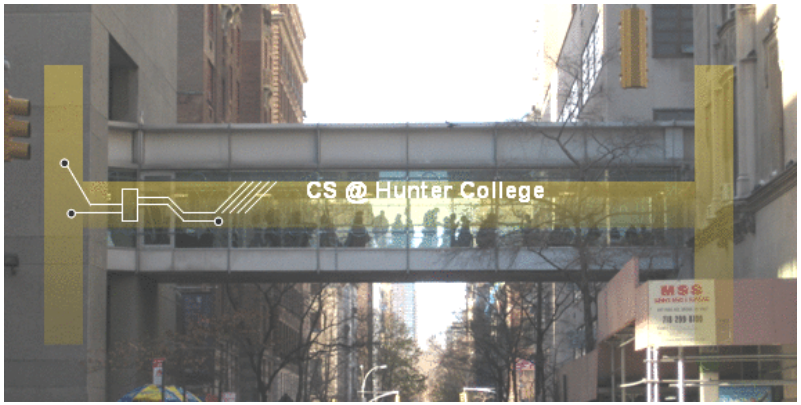
Weekly Reminders!



Before next lecture, don't forget to:

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

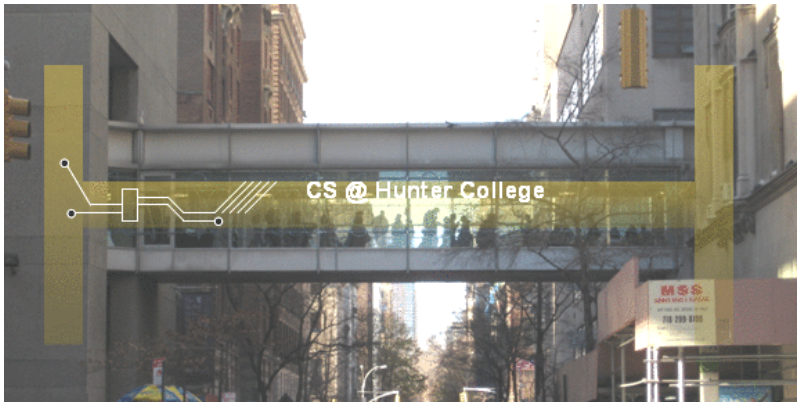
Weekly Reminders!



Before next lecture, don't forget to:

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

Weekly Reminders!



Before next lecture, don't forget to:

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