CMSC201
Computer Science I for Majors

Lecture 22 – Project 3
and Miscellaneous Topics

Last Class We Covered

- Dictionaries
  - Creating
  - Accessing
  - Manipulating
  - Methods
- Hashing
- Dictionaries vs Lists

Any Questions from Last Time?

Today's Objectives

- To understand more about how data is represented inside the computer
  - ASCII values
- To see the benefits of short circuit evaluation

- To discuss details of Project 3
  - How many boards to have?

ASCII Values

ASCII Values

- ASCII is how text is represented in computers
  - Just like binary is how numbers are represented
- In ASCII, every character has a unique, individual numerical code
  - Lowercase and uppercase characters are separate
  - Codes go from 0 to 127
    - Why 127?

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]

A TABLE

"control" characters

uppercase letters

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]

symbols & numbers

lowercase letters

Comparing Strings

- The values of the ASCII characters are used when comparing strings together
  - Which can lead to some "weird" results

```
>>> "cat" < "dog"
True
>>> "cat" < "Dog"
False
>>> "DOG" < "dog"
True
```

?

More on Comparing Strings

- Gets even more complex when you start adding in numbers and symbols

```
>>> "2" < "one"
True
>>> "good?" < "good!"
False
>>> "UK" < "U.K."
False
```

Rules for Comparisons

- To avoid (some) of these issues:

- Always use `.lower()` for comparing strings
- Pay attention to symbols
  - *e.g.*, spaces, hyphens, punctuation, etc.
  - Either remove them, or keep them in mind as part of the order

ASCII Characters to ASCII Values

- We can convert between ASCII characters and their values using **`ord()`** and **`chr()`**

- The **`ord()`** function takes in a single character, and returns its ASCII value
- The **`chr()`** function takes in an integer, and returns its ASCII character

Using `chr()` and `ord()`

```
>>> chr(65)
'A'
>>> chr(65+32)
'a'
>>> ord('?')
63
>>> ord("d")
100
>>> ord("e")
101
```

"Short Circuit" Evaluation

Review: Complex Expressions

- We can put multiple operators together!

```
bool4 = a and (b or c)
```

- What does Python do first?
  - Computes **`(b or c)`**
  - Computes **`a and`** the result

This isn't strictly true!

Short Circuit Evaluation

- Python tries to be efficient (*i.e.*, lazy), and so it won't do any more work than necessary
  - If the remainder of an expression won't change the outcome, Python doesn't look at it

- This is called "short circuiting"
  - It's a powerful tool, and can simplify the conditionals in your programs

Short Circuit Evaluation – Rules

- For obvious reasons, short circuiting behaves differently for **`and`** and **`or`** statements

- "**`and`**" statements short circuit as soon as an expression evaluates to **`False`**
- "**`or`**" statements short circuit as soon as an expression evaluates to **`True`**

Short Circuiting – `and`

- Notice that in the expression:

```
bool1 = a and (b or c)
```

- If `a` is `False`
- The rest of the expression doesn't matter

- Python will realize this, and if `a` is `False` won't bother with the rest of the expression

Short Circuiting – `or`

- Notice that in the expression:

```
bool1 = a or (b or c)
```

- If `a` is `True`
- The rest of the expression doesn't matter

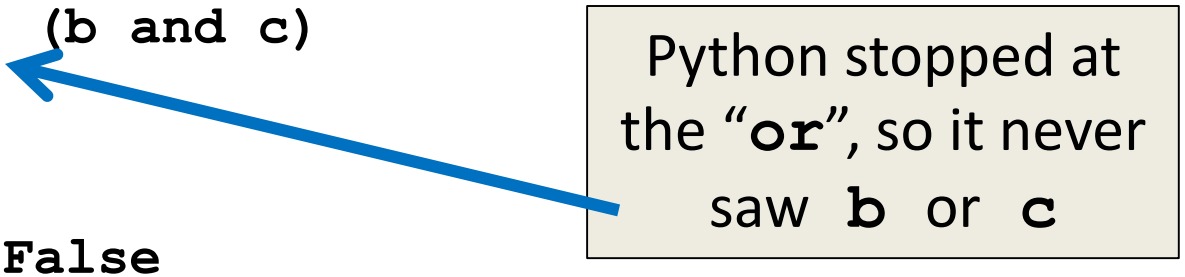
- Python will realize this, and if `a` is `True` won't bother with the rest of the expression

Causing Errors

- This can lead to "new" errors in old code

```
>>> a = True
>>> # Variables b and c not defined
>>> a or (b and c)
True

>>> a = False
>>> a or (b and c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
```



Simplifying Conditionals

- Order matters! You can use short circuiting to control what statements are reached

- While checking the validity of input, if the user can also enter a "Q" to quit

```
if num != QUIT and int(num) > MIN_VAL:
    return num
```

This will only be reached if num is not "Q", so the cast to int() won't cause a problem

Project 3

Do Not Cheat on Project 3

- Yes, this project has been given before
  - Yes, in this class
  - Yes, we have all of the old projects to compare it to
- Yes, this project has solutions on the internet
  - Yes, we have copies of all of them
  - Yes, we will go looking for new ones after it's due
- Yes, you could pay someone else to do it
  - Yes, we know of the sites where you can get this done
  - Yes, we will spot "elegant" code that you didn't write

Boards in Project 3

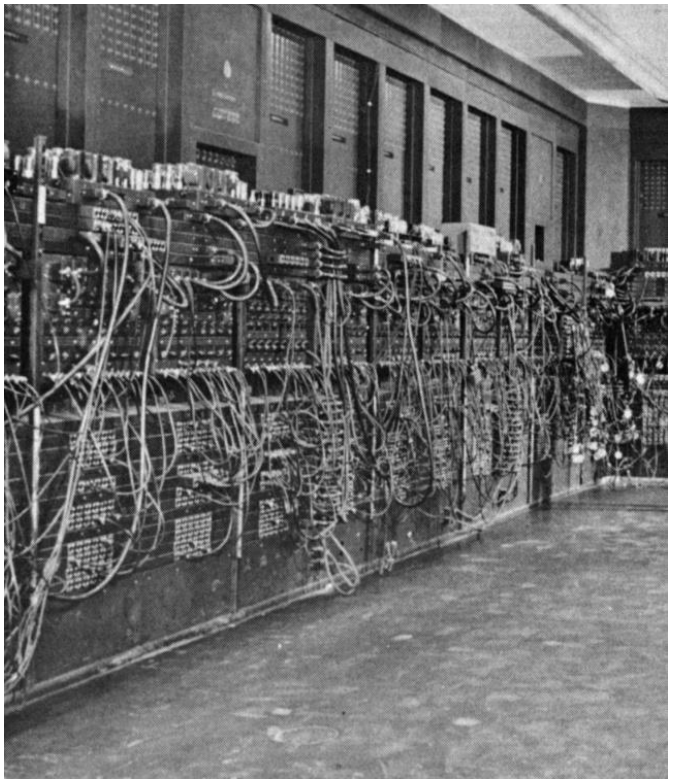
- Discussed in class
- ¯_(ツ)_/¯

Daily CS History

- John von Neumann
  - Creator of merge sort
    - We'll learn this soon!
  - Helped develop what is now known as "von Neumann architecture" (not all his work)
  - Created a rigorous framework for quantum mechanics
  - Developed implosion mechanism for nuclear bombs

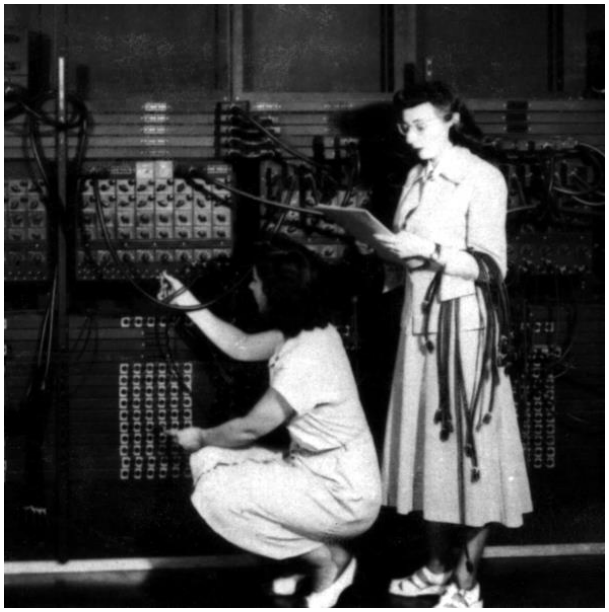
More Daily CS History

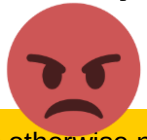
- ENIAC
  - Completed in 1946 at UPenn
    - Decommissioned in 1956
  - Computations were 2,400 times faster than humans
  - Cost $6.7 million to build
  - Meant to create artillery firing tables for the US Army
  - Also used for studying thermonuclear feasibility



Even More Daily CS History

- ENIAC Programmers
  - Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Meltzer, Fran Bilas, and Ruth Lichterman
  - These women turned abstract ideas into working, bug-free code
    - First program run on ENIAC had a million individual punchcards
  - Programming was seen back then as "easy" work, akin to typing up a handwritten letter





Announcements

- Project 3 design is due on Friday, May 4th
  - Project itself is due on Friday, May 11th
- Survey #3 out on Monday, May 7th
  - Final exam metacognition quiz out on BB same day

- Course evaluations are (not out yet)
- Final exam is when?
  - Friday, May 18th from 6 to 8 PM

Final Exam Locations

- Find your room ahead of time!
- **Engineering 027** - Sections 8, 9, 10, 11, 12
  Section 6
- **Meyerhoff 030** - Sections 2, 3, 4, 5
  Sections 14, 15, 16, 17, 30

Image Sources

- ASCII table (adapted from):
  - https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg
- Generic kitten:
  - http://www.publicdomainpictures.net/view-image.php?image=87454
- Generic puppy:
  - http://www.publicdomainpictures.net/view-image.php?image=192231
- John von Neumann:
  - https://en.wikipedia.org/wiki/File:JohnvonNeumann-LosAlamos.gif
- ENIAC (adapted from):
  - https://commons.wikimedia.org/wiki/File:Eniac.jpg
- ENIAC programmers (adapted from):
  - https://commons.wikimedia.org/wiki/File:Reprogramming_ENIAC.png
- Mad emoji (adapted from):
  - https://commons.wikimedia.org/wiki/File:Twemoji_1f620.svg