15-110 Principles of Computing – F21

Lecture 16: Strings 2

Teacher:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

String formatting using escape sequences

- print("He said, "What's there?" ") → SyntaxError: Invalid syntax
- print('He said, "What's there?" ') → SyntaxError: Invalid syntax

➢ Use **Escape sequence**

✓ An escape sequence **starts with a backslash** \ such that what follows is interpreted differently from usual (it is *protected*)

- print("He said, \"What's there? \" ") → Ok
- print('He said, "What\'s there?" ') → Ok

String formatting using escape sequences

- \n : **new line feed** is inserted print(" Hello!\nThis goes on a new line ")
- \t : **tabular space** is inserted print(" Hello!\t\tThis gets two tab spaces ")
- \\ : this allows to write file/folder **paths in windows** print("C:\\Python64\\Lib")
- \a : this **rings a bell**! print(" This rings a bell\a")

String formatting using escape sequences

Escape Sequence	Description
\newline	Backslash and newline ignored
\\	Backslash
\'	Single quote
\"	Double quote
\a	ASCII Bell
\b	ASCII Backspace
\f	ASCII Formfeed
\n	ASCII Linefeed
\r	ASCII Carriage Return
\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab
\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH

ASCII encoding for characters

- **Encoding:** Character → Integer number → Binary representation
- ASCII (American Standard Code for Information Interchange) standard code, defined in 1968 (and extended later on), assigns a numeric code (that can be hold in **8 bits = 1 byte**) to a subset of standard characters
- **1 byte: basic unit of storage in computer memory!**

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	[START OF HEADING]	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	[START OF TEXT]	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	[END OF TEXT]	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	[END OF TRANSMISSION]	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	[ENQUIRY]	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	[ACKNOWLEDGE]	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	[BELL]	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	[BACKSPACE]	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	[HORIZONTAL TAB]	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	[LINE FEED]	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	[VERTICAL TAB]	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	[FORM FEED]	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	[CARRIAGE RETURN]	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	[SHIFT OUT]	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	[SHIFT IN]	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	[DATA LINK ESCAPE]	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	[DEVICE CONTROL 1]	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	[DEVICE CONTROL 2]	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	[DEVICE CONTROL 3]	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	[DEVICE CONTROL 4]	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	[SYNCHRONOUS IDLE]	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	[ENG OF TRANS. BLOCK]	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	[CANCEL]	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	[END OF MEDIUM]	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	[SUBSTITUTE]	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	[ESCAPE]	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	[FILE SEPARATOR]	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	[GROUP SEPARATOR]	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	[RECORD SEPARATOR]	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	[UNIT SEPARATOR]	79	4F	1001111	117	O	127	7F	1111111	177	[DEL]
32	20	100000	40	[SPACE]	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Numeric encoding for characters

chr(*i*)

Return the string representing a character whose Unicode code point is the integer *i*. For example, `chr(97)` returns the string `'a'`, while `chr(8364)` returns the string `'€'`. This is the inverse of `ord()`.

The valid range for the argument is from 0 through 1,114,111 (0x10FFFF in base 16). `ValueError` will be raised if *i* is outside that range.

ord(*c*)

Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, `ord('a')` returns the integer `97` and `ord('€')` (Euro sign) returns `8364`. This is the inverse of `chr()`.

Comparison between strings

- Since each character is encoded as a number, we can compare two strings / characters!

```
'a' > 'z'
False
```

- Numeric encoding of 'a' is 96
- Numeric encoding of 'z' is 122
- ✓ 96 is not greater than 122!

```
'Hello' > 'Goodbye'
True
```

- Numeric encoding of 'H' is 72
- Numeric encoding of 'G' is 71
- ✓ The string starting with 'H' is greater than that starting with 'G'

Sorting on strings

- We can *sort* strings! → Get a sorted list of individual characters

```
s = 'I am a string'

sorted(s)
[' ', ' ', ' ', 'I', 'a', 'a', 'g', 'i', 'm', 'n', 'r', 's', 't']
```

- We can sort lists with string elements

```
L = ['Hello', 'Hola', 'Ciao']

sorted(L)
['Ciao', 'Hello', 'Hola']
```

```
L = ['Hello', 'Hola', 'Ciao', 3]
```

Error!

Converting (anything) to a string

str(x): Returns **x** converted as a string, **x** can be virtually anything …

str(3.2) → '3.2'

str(10) → '10'

str([1,3]) → '[1,3]'

str(True) → 'True'

Print the digits

Print out, one by one, all the digits of a (whatever) number n:

```
n = 762.95
n_str = str(n)
for i in n_str:
    print(i)
```

Compute the squared sum of all the digits of an integer number n.
E.g., $n = 567 \rightarrow 5^2 + 6^2 + 7^2 = 25 + 36 + 49$

```
n = 567
n_str = str(n)
sq_sum = 0
for d in n_str:
    sq_sum += int(d)**2
print('Sum of squared digits:', sq_sum)
```

Remove all whitespaces from a string

```
def remove_all_whitespaces(s):
    s_list = s.split()
    s = ''.join(s_list)
    return s
```

Reverse string without reversing words

```
def reverseWithoutReversing(s):
    '''Returns the input string with all words in reverse order but whithout
       reversing the single words, and removing all extra spaces.
       Words are separated by spaces and there
       might be multiple and/or lead/train spaces that need to be removed.
       E.g., " Hello  I like this   course " should be retuned as
       "course this like I Hello".
    '''
    s = s.strip()
    word_list = s.split()
    word_string = ' '.join(word_list[-1::-1])
    return word_string
```

s.strip() is not necessary (but it's a useful method!)

Is a pangram?

```
def isPangram(s):
    '''Check if ALL the letters of the alphabet appears at least once in
       the string s. Upper or lower case doesn't matter'''

    for letter_code in range(ord('a'), ord('z')):
        letter = chr(letter_code)
        if letter.upper() not in s and letter.lower() not in s:
            return False
    return True
```

Extract parts of a string

Implement the function `evens(s)` that takes a string `s` as input and returns another string composed only by the characters at even positions. For example, `evens('abcde')` should return "ace".

However, if the middle character of `s` is the same as the first and last characters of `s`, the function shall return a string of the same length as `s` but with all characters being the same as the middle character of `s`. For example, `evens('GATTGGAAHTAG')` should return `GGGGGGGGGGGG`.

Note that the *middle point* of a sequence of n elements depends on whether n is even or odd. For instance, if $n = 11$, the middle point is the element at position 5 (counting from 0). Instead, if $n = 10$ (even) the notion of middle point is not precisely defined, because it should be "between" the elements at positions 4 and 5. In these cases, we will consider the middle point being the element at position $\frac{n}{2}$. For examples, if `s` is the string `'0123456789'`, which consists of 10 characters, the middle point character is `'5'`. If `s` is the string `'1234567890*'`, the middle point character is `'6'`.

Construct valid file names

Users like to give their files all sorts of creative names. Unfortunately, computer systems can be limited in what they understand as a file name. Suppose that a system only allows file names that are composed of two parts, a *name* and an *extension* (`filename.ext`), and that follow the rules below for defining the name and the extension:

1. There must be one and only one "dot" (.).
2. The dot must separate the name and the extension.
3. The extension must be formed by exactly three characters.
4. There shall be **no** white spaces.
5. The name must have at least one character.
6. The name cannot start with a number.
7. The name can only contain alphanumeric characters (letters and digits from 0 to 9).

For instance, `speech.doc`, `speech2.txt` are valid filenames, while `speech-2.x`, `2speech.txt`, `speech and report.docx`, `my_file`, are all examples of invalid filenames.

Implement the function `is_valid_filename(s)` that takes a string as input and returns `True` if this string is a valid file name according to the rules above, or `False` otherwise.