Python Primers

Objectives:

- Use the input function to capture data provided by a user
- Convert strings of hex values to actual hex values and binary values
- Understand how the format method works with strings

Obtaining input from the user

The input built-in function allows for the capture of strings from a user. Input takes one argument, a string. When input is called, the argument is printed to the screen followed by a prompt. The prompt, a blinking cursor, waits for the user to act. Input will accept keyboard keystrokes and terminate once the enter key is pressed. The values entered by the user prior to the enter key will be stored in whatever variable is designated to the left of the equal sign. To see the input built-in function in action, type the following and run it.

Code

```
name = input(“What is your name? ”)
print(name)
```

There was a space added between the question mark and the end quotation within the input argument. This was done so that the string entered by the user did not start immediately after the question mark. Colons also make good separators when using input.

Indexing

Each element of a string, list, dictionary, or any other iterable object can be addressed by its index position. Let’s use the string “animal” as an example. We can find out the length of the string by using the built in len() function.

Code

```
word = "animal"
print (len(word))
```

The length of this string is 6. We can also identify each character within this string by its index position. The character found at index position 2 is “i”. This is because index positions start at 0.

a	n	i	m	a	l
0	1	2	3	4	5

The character at index position 0 is the letter “a”. We can verify this by using the following code to print the index position:

Code

```
print(word[0])
```

To print the last character of this string, you can either use the index of the last character or use -1. Using -1 as the index position will address the very last index position of the iterable object you are working with. Changing the index from 0 to -1 would result in the letter “l” being printed to the screen.

Code

```
print(word[-1])
```

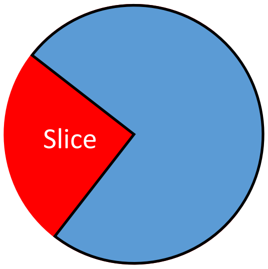
Index position -2 would be “a”, -3 would be “m”, and so forth until you reach the start of the string. If you try to address an index position that exceeds the beginning or end of the iterable object, Python will respond with an error that your index is out of range.

Code

```
print(word[6])
IndexError: string index out of range
```

Slicing

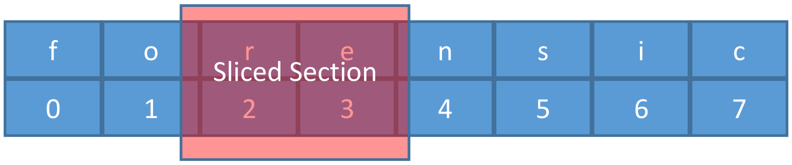
In Python, slicing is the process of removing a selection from a list, a string, or other set of iterable data to create a substring of that data. Think about slicing as taking a slice from a pizza or a pie. You want to remove a slice or section from the entire pizza to eat.



Take the word forensic as an example. There are eight characters in the word and each character has an index position as follows:

f	o	r	e	n	s	i	c
0	1	2	3	4	5	6	7

If we wanted to slice out the r and the e, we would want to remove the characters in position 2 and 3 respectively.



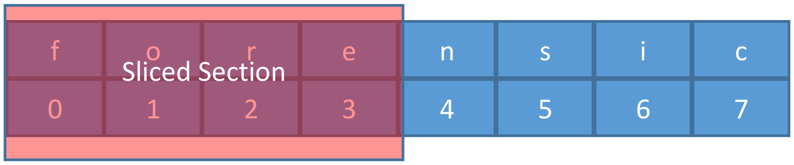
We can remove the r and e and store them in a new variable. In this example, we will use the word “selection” for the variable name. In Python, the syntax would look like the following:

Code

```
string = "forensic"
selection = string[2:4]
```

[2:4] is the slice syntax and it immediately follows the “string” variable. It means start at position 2 within the “string” variable and select up to, but not including, positon 4. Position 2 is the character “r” and position 3 is the character “e”. The slice does not include the 4th index position. The sliced result, “re”, will be stored in the variable labeled “selection”.

Leaving the first value empty in the slice will start the slice at the beginning of the string.

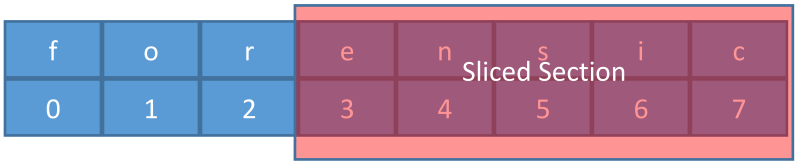


Code

```
string = "forensic"
selection = string[:4]
```

The variable, “selection”, will now contain the values from position 0 through 4, but not including 4. The variables contents will be “fore”.

Leaving the second value empty in the slice will select everything from the start of the slice to the end of the string.



Code

```
string = "forensic"
selection = string[3:]
```

The variable, “selection”, will now contain the values from position 3 to the end of the string. The variables contents will be “ensic”.

Exercise - slicing

Create a new Python document and save it as exercise_1.py.

Type the following code into your new blank document:

Code

```
decimal_value = 38
hex_value = hex(38)
print(hex_value)
```

Run the script. What is printed to the screen? ____

Modify the script so that it only prints the actual hex values to the screen.

What values did you use for the slice? [:]

Split

The split() method is used to separate targeted data within a string. Split takes a string argument and removes that set of characters from the string that the split() method is applied to.

When converting the decimal value 38 to hex, the result type was a string. We can verify that the value stored in the variable is a string by using the type() built in function:

Code

```
hex_value = hex(38)
print(type(hex_value))
```

Take, for example, the string “0x26” stored in hex_value. Working with the hex value “26” without the “0x” notation would be preferred in situations where you want to print hex values to the screen or convert the string of hex values to actual hex.

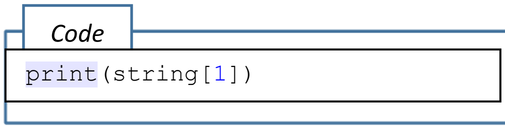
As with other string methods, such as format() and upper(), split() is used following the string with dot notation. Use the following code to remove the “0x” from this string using the split() method:

Code

```
hex_value = hex(38)
string = hex_value.split('0x')
```

Printing the variable “string” produces a list containing two elements, [‘’, ‘26’]. The “0x” is gone, but the result is still not optimal. The element we really want is in index position 1.

Code



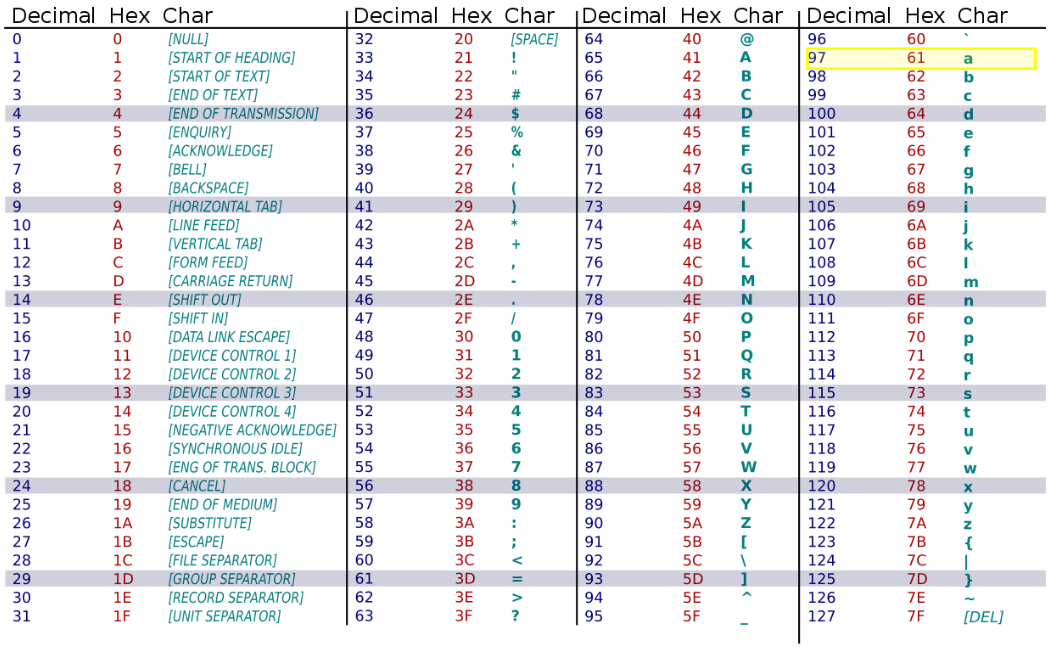
```
print(string[1])
```

Working with Strings

At some point, you may want to process data entered by a user. Python’s built-in input function will allow you to capture that data and then work with it within your script. The data you may want to process might be just strings of text to encode or hex values to perform a calculation on.

Converting ASCII characters to their decimal and hexadecimal equivalents.

The ordinal function, ord(), is used to convert a single character to its ASCII value. Take the letter ‘a’ for example. The corresponding decimal value in the ASCII chart is 97 and its hexadecimal representation is 0x61.



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]

https://upload.wikimedia.org/wikipedia/commons/thumb/1/1b/ASCII-Table-wide.svg/1280px-ASCII-Table-wide.svg.png

Use the following code to store the decimal equivalent of the letter ‘a’ to a variable labeled ‘dec’. Then print the value to the screen.

Code

```
dec = ord(“a”)
print(dec)
```

The value 97 is printed to the screen. But rather than dealing with decimal values, we will need to work with the hexadecimal equivalents when printing output to the screen or using other functions to encode and decode the data.

To convert a decimal value to its hexadecimal equivalent, the hex() built-in function is used. The hex() function takes one argument, an integer, and converts it to hexadecimal.

Code

```
hexvalue = hex(ord(“a”))
print(hexvalue)
```

The output is not optimal. The result is ‘0x61’. The actual value we will want to work with is ‘61’ by itself. We can use the slice() function to specify what data we actually want to work with. In this case, we want to start at index position 2, which is the ‘6’ and continue on until the end of the string. And the variable ‘hexvalue’ is a string and can be verified to be a string using the type() function.

The following code will remove the ‘0x’ and place the hexadecimal value alone into a variable called ‘hexvalue’:

Code

```
hexvalue = hex(ord(“a”))[2:]
print(hexvalue)
```

This works fine with one byte strings, but to convert a string of values input by the user, we will need to use a for loop to loop through each character in the string and convert each and either print to the screen directly or store the end result in a variable.

Code

```
def stringtohex(string):
        result = “”
        for i in string:
                result += ((hex(ord(i))[2:]).upper())
        return result

string = “forensic”
result = stringtohex(string)
print(result)
```

Strings to Binary

It may be desired to print to screen or a file the binary equivalent of hex values. This can be achieved using the fromhex method in the bytes function.

Code

```
binary = bytes.fromhex(hex(ord(“a”))[2:])
print(binary)
```

The problem now is that the letter “a” is still printed to the screen, even though it is preceded with binary notation. The solution was able to give us a value in binary, but like decimal and hex, if the value has a corresponding ASCII character, that character is printed to the screen instead. We need a way to force the binary value to be printed to the screen.

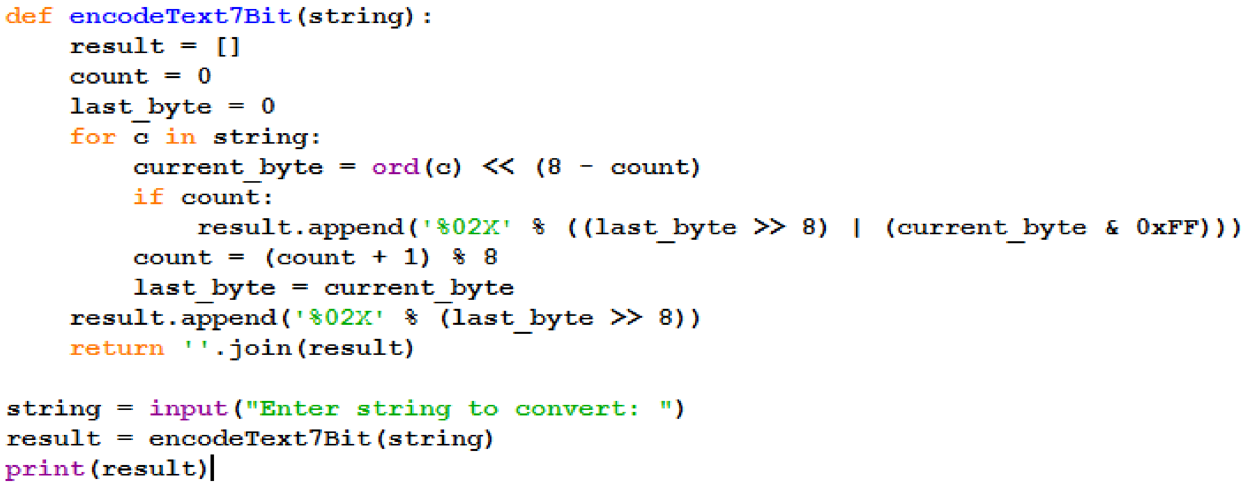
This is where we can use the string format method along with the bin function to force the bits to be printed to the screen:

Code

```
def binview(value):
        return ("{0:08}".format(int(bin(value).split("0b")[1])))

value = stringtohex(“forensic”)
value = bytes.fromhex(value)
for i in value:
        print(binview(i), end = “”)
```

GSM7 Bit Encoder



```
def encodeText7Bit(string):
    result = []
    count = 0
    last_byte = 0
    for c in string:
        current_byte = ord(c) << (8 - count)
        if count:
            result.append('%02X' % ((last_byte >> 8) | (current_byte & 0xFF)))
        count = (count + 1) % 8
        last_byte = current_byte
    result.append('%02X' % (last_byte >> 8))
    return ''.join(result)

string = input("Enter string to convert: ")
result = encodeText7Bit(string)
print(result)
```