15-110 Principles of Computing – S19

Lecture 4:

Binary representations, String data types

Teacher:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Scalar vs. Non-scalar objects

- Scalar type objects:
  - `int`
  - `float`
  - `complex`
  - `bool`
  - `None`

Indivisible

- Non-Scalar type literal objects:
  - `str`: **String of characters (text):**
    - “Hi” , ‘Hello!’, “Number 5”
  - `tuple`
  - `list`
  - `set`
  - `dict`

Internal structure

- Made of multiple components
- Individual or subsets of components can be addressed for read/write operations

➢ Scalar vs. Non-scalar terminology, from *math*

✓ It is termed a scalar any *real number*, or *any quantity* that can be measured using a **single real number**

✓ A vector is made of **multiple scalar components** (represents a point in a multi-dimensional space)

Basic examples of using string objects: single and double quotes

- “Hi”
- ‘Hello!’
- “Number 5”
- “abc”
- “Hello!”
- ‘z’,
- ‘abc’
- ‘_wow_’
- “I’m Joe”
- ‘ Say “hello!” to her’

Double quotes

Single quotes

Double and single quotes together

```
str1 = “This is”
str2 = “spam!”
print( str1, str2)
```

```
This is spam!
```

```
name = “ ”
introduction = “My name is”
name = “Gianni”
print( introduction, name)
```

```
My name is Gianni
```

Basic examples of using string objects: triple quotes

- long_str = '''Hi this is a veeeeeeery long string of text that I would like to write over multiple lines '''
- long_str = """Hi this is a veeeeeeery long string of text that I would like to write over multiple lines """

Triple (single or double) quotes

```
long_str = '''Hi this is a veeeeeeery long string of text
that I would like to write over multiple lines '''
print(long_str)
```

```
Hi this is a veeeeeeery long string of text
that I would like to write over multiple lines
```

String objects

- A string is a *sequence* of characters
  - ✓ Sequence → *Ordering*, indexing
  - ✓ Characters → Which type of characters are allowed? → **Unicode set**

- Sequence:

"Hello Joe"

H	e	l	l	o		J	o	e
0	1	2	3	4	5	6	7	8

Indexing of the positions of the individual characters in the string
→ Access to the individual components of the string type

- Characters: A character is a symbol
  - E.g., the English alphabet has 26 symbols, other alphabets have different sets of symbols, plus we need characters for punctuation, characters for mathematics, characters for …
  - Computers do not deal with *characters*, they deal with numbers (binary). Every character is internally stored and manipulated as a combination of 0's and 1's
  - **Encoding:** Character → Integer number → Binary representation → Python uses **Unicode encoding**

Numeric conversions between different bases

- Let's consider an **integer number $\boldsymbol{x}$** with $n = 5$ digits, e.g., $x = 64523$
- This is a base 10 $(b = 10)$ representation of the number, using digits from 0 to 9
  - $x = 6 \cdot 10^4 + 4 \cdot 10^3 + 5 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 = 64{,}523$

Position	4	3	2	1	0
Exponent	10^4	10^3	10^2	10^1	10^0
Value	10,000	1,000	100	10	1
Digits	x_4	x_3	x_2	x_1	x_0

Position	4	3	2	1	0
Exponent	2^4	2^3	2^2	2^1	2^0
Value	16	8	4	2	1
Digits	x_4	x_3	x_2	x_1	x_0

- Let's consider now a **binary number $\boldsymbol{x}$** with $n = 5$ digits, e.g., $x = 11001$
- This is a base 2 $(b = 2)$ representation of the number, using digits 0 and 1
- What is the integer value of the number x?
  - $x = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 25$
  - How many *unsigned* integer numbers can be represented with 8 bits? → 256
  - How many *signed* integer numbers can be represented with 8 bits? → 128
  - Internal *non-scalar* representation of numbers

Binary	Octal	Decimal	Hexadecimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
Base-2	*Base-8*	*Base-10*	*Base-16*

Bits

- **One bit** (that can take on two values, 0 or 1)
  - We can represent 2 integer numbers: 0 1
  - The max value of an integer that we can represent with 1 bit: 1

- **Two bits**
  - We can represent 4 integer numbers: 00 01 10 11, from 0 to 3
  - The max value of an integer that we can represent with 2 bits: 3 (obtained from $2^2 - 1$)

- **Three bits**
  - We can represent 8 integer numbers: 000 010 100 110 011 101 001 111, from 0 to 7
  - The max value of an integer that we can represent with 2 bits: 7 (obtained from $2^3 - 1$)

…….

- **8 bits = 1 byte**
  - We can represent 256 (unsigned) integer numbers: from 0 to 255
  - The max value of an integer that we can represent with 8 bits: 255 (obtained from $2^8 - 1$)

Numeric conversions between different bases

Position	7	6	5	4	3	2	1	0
Exponent	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1
Digits	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0

MSB LSB

- From base 10 to base 2?
- Keep dividing by 2 and storing the remainder
- → Modulo operation!!!

Number	294		
divide by 2			
result	147	remainder	0 (LSB)
divide by 2			
result	73	remainder	1
divide by 2			
result	36	remainder	1
divide by 2			
result	18	remainder	0
divide by 2			
result	9	remainder	0

Dividing each decimal number by "2" as shown will give a result plus a remainder.

If the decimal number being divided is even then the result will be whole and the remainder will be equal to "0". If the decimal number is odd then the result will not divide completely and the remainder will be a "1".

divide by 2			
result	4	remainder	1
divide by 2			
result	2	remainder	0
divide by 2			
result	1	remainder	0
divide by 2			
result	0	remainder	1 (MSB)

The binary result is obtained by placing all the remainders in order with the least significant bit (LSB) being at the top and the most significant bit (MSB) being at the bottom.

$$(294)_{10} = (100100110)_2$$

ASCII encoding

- **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	

Unicode encoding

- **Encoding:** Character → Integer number → Binary representation
- Developed in recent times to address the widespread use of computers in different countries using different symbols in their alphabet
- Different Unicode codes are around, using encoding larger (and more complex) than the 8 bits of ASCII, allowing to index code points (characters) large enough, to represent virtually any language around
- The most commonly used Unicode encoding is the UTF-8, that is fairly compact and includes ASCII codes
- Your Spider makes use of UTF-8!

String indexing

- Indexing:

"Hello Joe"

H	e	l	l	o		J	o	e
0	1	2	3	4	5	6	7	8

Indexing of the positions of the individual characters in the string
→ Access to the individual components of the string type

- Index starts from 0 and must be an integer
- Notation to **access the n-th component** in a string variable my_string: my_string[n]

```
greet="Hello Joe"
print(greet[0], greet[4], greet[6])
```

- We can use *variables* as index:

```
x = 3
print(greet[x])
```

H	e	l	l	o		J	o	e
-9	-8	-7	-6	-5	-4	-3	-2	-1

- We can also *index from the right end of the string* (useful to get the last character!)

```
print(greet[x-4])
```

String operators

- **String concatenation**, + operator, *overloaded*: It returns a string consisting of the string operands joined together

```
greet_joe = “Hello Joe”
comma = “,”
greet_mary = “hello Mary”
greet = greet_joe + comma +  greet_mary
print(greet)
```

```
Hello Joe,hello Mary
```

Can I do `greet + 1`? NO!

- **String duplication**, * operator, *overloaded:* It creates multiple copies of a string. If s is a string and n is an integer:

```
s = “Hello”
n = 4
print(s * n)
s4 = n * s
print(s4)
```

```
HelloHelloHelloHello
```

```
HelloHelloHelloHello
```

```
s = “Hello”
n = -4
print(s * n)
```

```

```

Can I do `s*s`? NO!

```
s = (3 * “spam”) + “ – “ + (“StopSpam” * 5)
```

String operators

- **Part of,** `in` operator, *overloaded:* Membership operator that returns `True` if the first operand is contained within the second, and `False` otherwise

```
s = “Joe”
in_hello = s in “Hello Joe”
in_food = s in “Yummy meal”
print(in_hello, in_food, type(in_hello))
```

```
True False <class 'bool'>
```

- **Not Part of,** `not in` operator, *overloaded:* Membership operator that returns `True` if the first operand is not contained within the second, and `False` otherwise

```
s = “Joe”
in_hello = s not in “Hello Joe”
in_food = s not in “Yummy meal”
print(in_hello, in_food, type(in_hello))
```

```
False True <class 'bool'>
```