110 Review Session Quiz I

Basic Data Types

Numerical Types (int, float)

- Represented as integers and floats
- Can be used with standard mathematical operations (e.g. +, -, *, /,**)
- Can be combined into expressions using order of operations
- Note: // returns an integer but / returns a float

```
>>> 4 + 4
8

>>> 4 - 5
-1

>>> 4**2
16

>>> 10/5
2.0

>>> 10//5
2

>>> ((4 + 3) * (12 + 6))**2
15876
```

String (str)

- Main representation of text in code
- Represented with single or double quotes
- Can be concatenated together using the + operator
- Within a print statement, a comma can be used for joining strings together

```
>>> x = "Hello world"

>>> y = " Goodbye"

>>> x + y
'Hello world Goodbye'

>>> print(x + y)
Hello world Goodbye

>>> print(x, "Goodbye")
Hello world Goodbye
```

Boolean (bool)

- Either True or False
- Result of comparison operators (<, >, <=, >=, ==, !=)
- Can only compare things of the same type (even strings!)
- Can join multiple comparisons together with the ‘and’ operator and ‘or’ operator
- The ‘not’ operator inverts a truth value

```
>>> 4 > 5
False

>>> (4 < 5) and (4 > 3)
True

>>> not True
False

>>> not False
True

>>> (4.0 > 4) or (4.0 >= 4)
True

>>> x = 5 == 6

>>> x
False

>>> x = 5 != 6

>>> x
True
```

Converting Types

- You can convert between different types
- str(x): converts x to string
- int(x): converts x to integer
- float(x): converts x to float

```
>>> x = 4.0

>>> str(x)
'4.0'

>>> int(x)
4

>>> float(x)
4.0

>>> y = "I am " + str(x) + " years old"

>>> y
'I am 4.0 years old'
```

Basic Data Types Practice Problems

```
>>> (5**2*4)/2+8//2
```

- How would you print a string that declares the average of x,y, and z, which are strings? Examples are shown below.

```
x='8'
y='2'
z='6'
```

```
'The average is 5.'
```

```
x='8'
y='2'
z='7'
```

```
'The average is 5.'
```

```
x='8'
y='2'
z='8'
```

```
'The average is 6.'
```

Error Types

Syntax Errors

- Occurs when you write code that is incorrect Python
- Examples include missing quotes and incorrect indentation
- White space (indentations) is considered syntax too
- Always read the error message for more information!

```
Kernel process terminated for restart. (0)

Python 3.8.11 (default, Aug  6 2021, 09:57:55) on Windows (64 bits).
This is the Pyzo interpreter with integrated event loop for PYQT5.
Type 'help' for help, type '?' for a list of *magic* commands.
Running script: "C:\Users\fhu14\Desktop\rec2.py"
  File "C:\Users\fhu14\Desktop\rec2.py", line 1
    x = "What is the weather?
                             ^
SyntaxError: EOL while scanning string literal
```

```
>>>     4 > 4
  File "<console>", line 1
    4 > 4
    ^
IndentationError: unexpected indent
```

Runtime Errors

- Errors which occur when the code is executed, but syntax is fine
- Common examples include type errors and divide by zero
- Remember that comparisons (other than == and !=) can only be done between elements of the same type!

```
>>> 4.0 > "Hello"
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: '>' not supported between instances of 'float' and 'str'

>>> 4.0 == "Hello"
False

>>> 4.0 != "Hello"
True

>>> "Hello" >= 5
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: '>=' not supported between instances of 'str' and 'int'

>>> 3.0/0
Traceback (most recent call last):
  File "<console>", line 1, in <module>
ZeroDivisionError: float division by zero
```

Logical Errors

- What we tend to think of when we talk about “bugs”
- The code runs, but it doesn’t do what you want!
- Requires more in-depth debugging techniques to figure out the problem

```
1  def average(x, y, z):
2      #logical error: order of operations
3      #   not obeyed
4      return x + y + z/3
5
6  def isEven(x):
7      #logical error: boolean values are
8      #   switched
9      if x % 2 == 0:
10         return False
11     else:
12         return True
13
14 import math
15 def squareInput(x):
16     #logical error: incorrect argument order,
17     #   computing 2^x instead of x^2
18     return math.pow(2, x)
19
```

Error Types Practice Problems

Find the errors in these functions.

```
def f(x):
    x+4=tmp
    y=5
    z==y
    print     ("Hello!")
        return tmp+z

def g()
    c='Fred'
    d='Fido'
    cat_age=2
    dog_age=4
    print("My dog, "+c+", is "+cat_age+" years old and my cat,
"+d+" is "+dog_age+' years old.")
```

Variables

Variables Store Values

- We can store anything we want in variables
- Literal values, results of function calls, even other variables
- Variables are convenient when you want to re-use certain values multiple times
- Variable values can change throughout the code

```
1  x = 5.0
2  y = "The temperature is"
3  z = x ** 2
4  g = y + " " + str(z) + " degrees."
5
6  x = x + 10
7  z = x ** 2
8  g = y + " " + str(z) + " degrees."
9
10 #Final values
11 #x = 15.0
12 #z = 225.0
13 #g = "The temperature is 225.0 degrees."
14 #y = "The temperature is"
15
```

Variable Naming Conventions

- Variables cannot start with punctuation or numbers
- Good variable names are descriptive!
- Use camelCase or snake_case for variable names, but be consistent

```
1 x = 5 #Bad variable name, but valid assignment
2
3 numPages = 5 #Good variable name
4
5 12Variable = 4 #Invalid variable name, SyntaxError
6
7 $Variable = 3 #Invalid variable name, SyntaxError
8
```

Code Evaluates Sequentially

- Variables must be defined before they are used
- When tracing code, go line by line
- Keep track of variable values as you go
- Be sure to note when a variable's value is updated!

```
1  import math
2  #Error because x is called before it is
3  #   assigned a value
4  result = math.pow(2, x)
5  x = 5
6
7  import math
8  #This is fine
9  x = 5
10 result = math.pow(2, x)
11
```

```
1  x = 5
2
3  y = x + 5 #y evaluates to 10
4
5  x = x + 3 #x updates to 8
6
7  z = x * y #z evaluates to 80
```

Number Systems and Abstraction

Everything is Binary

- Computers depend on systems of 1’s and 0’s to represent all data
- Binary is base 2, and is analogous to base 10
- Read from right to left, each bit’s value is double the previous bit

10^3 **1000**	10^2 **100**	10^1 **10**	10^0 **1**
1	9	8	0

2^3 **8**	2^2 **4**	2^1 **2**	2^0 **1**
1	1	0	1

Converting Between Binary and Decimal

- To go from binary to decimal, multiply all the place values by their respective bit value and add together

Convert 101001 to decimal

32	16	8	4	2	1
1	0	1	0	0	1

(1 * 32) + (0 * 16) + (1 * 8) + (0 * 4) + (0 * 2) + 1 = 41

Convert 110111 to decimal

32	16	8	4	2	1
1	1	0	1	1	1

(1 * 32) + (1 * 16) + (0 * 8) + (1 * 4) + (1 * 2) + 1 = 55

Converting Between Binary and Decimal

- To go from decimal to binary, successively remove bit place values from largest to smallest until you reach 0
- Only remove bit place value if the remaining decimal value is greater
- You should use as many bits as you need, and it's fine to have extra zeros out front

Convert 98 to binary = 1100010

Bit place value	**Remaining value**	**Bit value**
64	98 - 64 = 34	1
32	34 - 32 = 2	1
16	2 < 16	0
8	2 < 8	0
4	2 < 4	0
2	2 - 2 = 0	1
1	0 < 1	0

Binary for Representing Characters

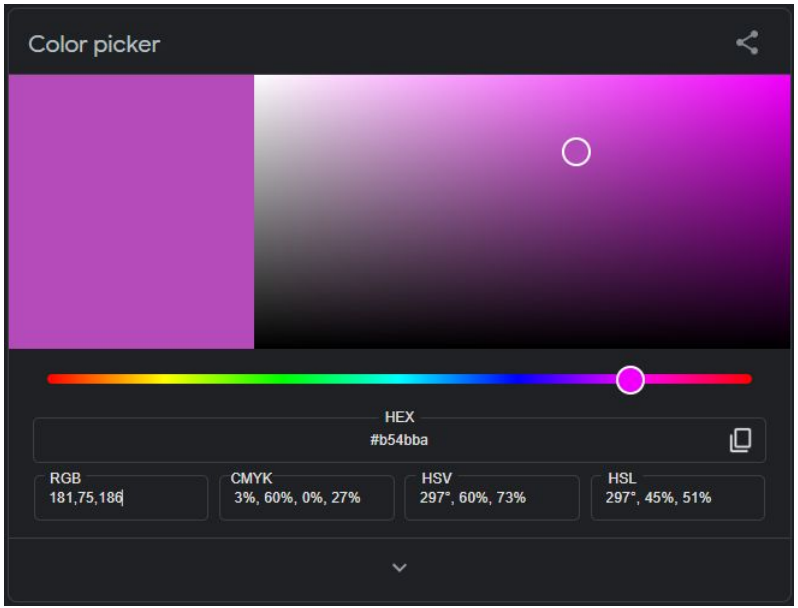
- Convert from binary to decimal and find the corresponding value on the ASCII table
- Can also go the other way (decimal to binary) to find the binary representation of an ASCII character
- Ex. 1100010 →98→'b'

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Binary for Representing Colors

- Binary can be used to represent colors in RGB
- One binary number for each component's intensity, three in total
- Each component contains 8 bits (0 → 255 max value)

R = 10110101 = 181
G = 01001011 = 75
B = 10111010 = 186



Number Systems and Abstraction Practice Problems

- Convert 16 to 4 bit binary.
- Convert 100 to 8 bit binary.
- Convert 10101111 to decimal.
- What is the highest value that k bits can produce?
- How many bytes and bits for an RGB color?
- Convert ‘Z’ to binary.

Abstraction & Algorithms

Algorithms

Algorithms are procedures that specify how to do a needed task or solve a problem.

- Standardized
- Human-made
- inputs, steps, outputs
- Robustness, correctness, efficiency, clarity

Used all over the place (recipes, ikea instructions, directions, etc)!

Abstraction

Abstraction is a technique used to make complex systems manageable by changing the amount of detail used to represent or interact with the system.

- Low level abstraction - needs specific details, not much is known problem at hand
- Medium level abstraction - some specifics, know something about the problem at hand
- High level abstraction - No specifics, knows a good amount of problem at hand

Practice Problem

Create an algorithm to explain to your friend how to make a cup of tea.

Low level - They know nothing about what tea is/a kettle

Medium level - Know what tea is/know what a kettle is

High Level - has brewed tea before

Practice Problem

Low –

Define a kettle has a metal container that stores water

Define tea as leafs in a bag

Define a faucet as an output of water

Define a mug as a ceramic cup that holds liquid

Define a stove as something that produces flame/heats up

Go to the faucet and open the top of a kettle, turn on the faucet by turning the hot water knob.

Fill the kettle with water until the fill line and then put it on the stove and turn the stove on with the knob

Wait until you hear a whistle, then turn the knob on the stove off

Retrieve the tea and put it in the cup, then fill the cup with water

Wait 3 minutes and then enjoy

Practice Problem

Medium –

Fill the kettle with water from the faucet up until the fill line

Turn on the stove; boil the water

Put the tea bag in the mug

Pour the water into the mug

Wait 3 minutes and enjoy

Practice Problem

High –

Boil water in a kettle

Put a tea bag in a mug

Pour the boiling water into the mug

Wait 3 minutes and enjoy