CSC110 Lecture 2: Representing Data

Print this handout

Exercise 1: Data types in Python

1. In the table below, write down the Python type for each Python literal. The first row is done fo you.

Data Type	Python Literal
`int`	`12345`
float	`12345.0`
bool	`True`
str	`'You are doing great :)'`
set	`{'David', 'Tom'}`
list	`[1, 2, 3, 4, 5]`
dict	`{'cool': ['David', 'Tom', 'You']}`

2. What are *two* differences between the `set` and `list` data type in Python?

   1. Order of elements does not matter in sets, but it does matter in lists.
   2. Lists can contain duplicates while sets can not.

3. Without using the Python console, complete the table below.

Python Expression	Value	Type of Value
`9 // 3`	3	int
`9 / 3`	3.0	float
`2 + 2.0`	4.0	float
`3 in {1, 2}`	False	bool
`'x' + 'y'`	'xy'	str
`not False`	True	bool
`[1, 2] + [2, 3]`	[1, 2, 2, 3]	list
`['hi', 'bye'][0]`	'hi'	str
`{'hi': 'bye'}` `['hi']`	'bye'	str

When you are finished, check your work using the Python console.

4. Imagine that you are the owner of a neighbourhood grocery store that sells many kinds of food Think about different kinds of data you would have to keep track of. For each data type below, write a description of a piece of data that would have that type. We have completed the first ro for you.

Data Type	Example data of that type
Boolean	"Is my store wheelchair accessible?"
Real Number	How much cash is present in the cash register?
String	Which item is out of stock?
Set	Collection of employee IDs
List	Collection of transactions made
Mapping	Amount of sale generated by each employee matched by employee ID

Exercise 2: Variables

1. Suppose we execute this code in the Python console:

```
>>> x = 4
>>> y = x + 2
>>> z = {y: x}
```

Complete the value-based memory model table below to show what `x`, `y`, and `z` refer to.

Variable	Value
x	4
y	6
z	{6:4}

2. Repeat the above exercise, now with the following code:

```
>>> a = 'cat'
>>> b = [a[0], a[1], a[2]]
>>> c = ('c' in a) and ('c' in b)
>>> d = ('cat' in a) and ('cat' in b)
```

Variable	Value
a	'cat'
b	['c', 'a', 't']
c	True
d	False

3. *When things go wrong*. As part of reviewing this lecture, your friend Mikasa opens the Python console and tries to assign a variable `x` to the integer value `5`. This is what she types and then sees:

```
>>> 5 = x
Traceback (most recent call last):
... [some output omitted] ...
File "<input>", line 1
    5 = x
    ^
SyntaxError: cannot assign to literal here. Maybe you meant '=='
        instead of '='?
```

Read what your friend typed and the error message, and then write down: (1) a brief explanation of the problem, and (2) how to correct the problem.

Additional exercises

1. *Order of operations*. When combining multiple operations in the same expression (e.g., `3 + * 2`), we recommend using parentheses to make it clear what order the operations are evaluated in. However, you might see code where the programmer didn't include parentheses, and you'll need to figure out what the code is doing yourself.

   Python follows the same order of arithmetic operations that you're used to from mathematics (sometimes referred to as "BEDMAS" or "PEMDAS"), with the modulo `%` operator at the same precedence as multiplication/division.

   In the table below, add parentheses to indicate the order that operations are evaluated, and then write the value of the expression. Make sure to check your work using the Python console

Python Expression	Expression with parentheses	Value
`3 + 5 * 2`		
`3 / 5 * 2`		
`3 * 5 / 2`		
`3 + 5 % 2`		
`4 + 8 / 2 ** 2 / -2`		

2. *Terminology recap*. For each piece of code, circle all terms that can be used to describe it.

   1. `4 + 5`

      a. literal
      b. expression
      c. statement

   2. `'Hi ' + 'there'`

      a. literal
      b. expression

   c. statement

3. `'Hi'`

   a. literal
   b. expression
   c. statement

4. `'fries' in  {'fries': 5.99, 'steak': 25.99, 'soup': 8.99}`

   a. literal
   b. expression
   c. statement

5. `distance1 = ((1 - 2) ** 2 + (3 - 5) ** 2) ** 0.5`

   a. literal
   b. expression
   c. statement

6. `{1, 2, 3} == {3, 1, 2}`

   a. literal
   b. expression
   c. statement

3. *Error message follow-up (1)*. Recall Exercise 2, Question 3 from above. Your friend Mikasa entered an invalid piece of code into the Python console, and saw the following:

```
>>> 5 = x
Traceback (most recent call last):
  ... [some output omitted] ...
  File "<input>", line 1
    5 = x
    ^
SyntaxError: cannot assign to literal here. Maybe you meant '==' instead of '='?
```

   Rather than listen to your great explanation, she then read the error message and tried typing in something different, but then saw a different error message. 😔

```
>>> 5 == x
Traceback (most recent call last):
  ... [some output omitted] ...
  File "<input>", line 1, in <module>
NameError: name 'x' is not defined
```

Explain this new error.

4. *Error message follow-up (2).* Below, we have written several assignment statements. For each one, determine whether it is valid Python code. If it is *invalid*, explain what the problem is and how to fix it (if it makes sense to do so).

   Try doing this without the Python console first, and then check your work in the Python console.

   a. `5 = x`

   b. `x = 5.0`

   c. `x = hello!`

   d. `x = true`

   e. `x = {}`

   f. `x = [1, 'hi', 3]`

   g. `x = {1: 'two', 'three'}`