1. List and dictionary comprehensions

List comprehensions are a shorter way to create lists.

A list comprehension has the form:

```
variable = [expression for item in list]
```

You can also add a condition:

```
variable = [expression for item in list if condition]
```

For example:

```
nums = [x * 2 for x in range(5)]
```

creates the list:

```
nums = [0, 2, 4, 6, 8]
```

Fill in the blanks.

a. Create a list containing the numbers from 1 to 10.

```
nums = [_____________ for _____________ in range(1,11)]
```

Solution:

```
nums = [x for x in range(1, 11)]
```

b. Write a list comprehension that creates a list containing only the even numbers from 0 to 9.

```

_____
```

Solution:

```
evens = [x for x in range(10) if x % 2 == 0]
```

c. Write a dictionary comprehension that maps each number from 1 to 5 to its square.

Example:

```
{1:1, 2:4, 3:9, 4:16, 5:25}
```

```

_____
```

Solution:

```
squares = {x: x * x for x in range(1, 6)}
```

2. String functions

Python has useful string functions:

```
sentence.split(" ")      # string -> list

"-".join(words)          # list -> string

text.upper()             # uppercase

text.lower()             # lowercase
```

What does each expression produce?

Expression	Output
`"a,b,c".split(",")`	
`"-".join(["a","b","c"])`	
`"Hello".lower()`	
`"python".upper()`	

Solution:

Expression	**Output**
`"a,b,c".split(",")`	`["a", "b", "c"]`
`"-".join(["a","b","c"])`	`"a-b-c"`
`"Hello".lower()`	`"hello"`
`"python".upper()`	`"PYTHON"`

3. Finding the error

Match each buggy code with the correct error type.

Possible error types:

`IndexError` `KeyError` `TypeError` `ValueError`

Code	**Error type**
`nums = [1, 2, 3]` `print(nums[5])`	
`d = {"a": 1}` `print(d["b"])`	
`5 + "hello"`	
`int("abc")`	

Solution:

Code	Error type
`nums = [1, 2, 3]` `print(nums[5])`	`IndexError`
`d = {"a": 1}` `print(d["b"])`	`KeyError`
`5 + "hello"`	`TypeError`
`int("abc")`	`ValueError`

Now debug the code below:

```
numbers = [1, 2, 3]

for i in range(4):
    print(numbers[i])
```

What error will this code produce? Explain why.

Solution:

The code produces an `IndexError`.

The list has only three elements, so the valid indices are `0`, `1`, and `2`. When `i` becomes `3`, the statement

```
print(numbers[3])
```

tries to access an element that does not exist.

A correct version is:

```
numbers = [1, 2, 3]

for i in range(len(numbers)):
    print(numbers[i])
```

Another correct solution is:

```
numbers = [1, 2, 3]

for number in numbers:
    print(number)
```