UNIVERSITY OF ENGINEERING AND TECHNOLOGY PESHAWAR, KOHAT CAMPUS

Lab 3: Different Pythonic Coding Tricks and Construct

Lab Title: EE-271 “OOP & Data Structures Lab”

Date: Tuesday, October 24, 2023 Time: 10 min/ Task

Lab report Work

1. Observe the output of the following if:

```
a = 5
b = 10
f'Five plus ten is {a + b} and not {2 * (a + b)}.
```

2. Make a list of odd number using list comprehension.

Lab practice

Sting formatting:

1. Observe the output of the following if:

```
errno = 50159747054

`name = 'Bob'
```

a. 'Hello, %s' % name

b. 'Hey %s, there is a 0x%x error!' % (name, errno)

c. 'Hey %(name)s, there is a 0x%(errno)x error!' % { "name": name, "errno": errno }

2. Observe the output of: (Mostly used)

a. 'Hello, {}'.format(name)

3. Observe the output of: (Recommended for python 3.6+)

a. f'Hello, {name}!'

4. Observe the output of:

a.

```
from string import Template

t = Template('Hey, $name!')

st.substitute(name=name)
```

b.

```
templ_string = 'Hey $name, there is a $error error!'

Template(templ_string).substitute(name=name, error=hex(errno))
```

For loop:

1. Consider the following code.

```
my_items = ['a', 'b', 'c']

for i in range(len(my_items)):

    print(my_items[i])
```

   a. What will be the output?

1. Consider the following pythonic style for loop. in Python, for-loops are really “for-each” loops that can iterate directly over items from a container or sequence, without having to look them up by index.

```
for item in my_items:

    print(item)
```

2. What if you *need* the item index and item, for example? The enumerate() built-in helps you make those kinds of loops nice and Pythonic. Note that the item is printed directly without using the index.

```
for i, item in enumerate(my_items):

    print(f'{i}: {item}')
```

3. Iterate over the keys and values of a dictionary at the same time. Consider the following code

```
emails = {'Bob': 'bob@example.com','Alice': 'alice@example.com',}

for name, email in emails.items():

    print(f'{name} -> {email}')
```

4. What if you must control the step size for the index? The range() function comes to our rescue again—it accepts optional parameters to control the start value for the loop (a), the stop value (n), and the step size (s).

```
for i in range(1, 15, 2):

    print(i)
```

List comprehensions:

a. Consider the following code and print the square.

```
squares = []

for x in range(10):

    squares.append(x * x)
```

b. Replace the above code with the following. Print the square. This sample one-line code is called list comprehension.

```
squares = [x * x for x in range(10)]
```

c. Chane `x*x` by any function of x and observe the resulting list.

d. List comprehensions can filter values based on some arbitrary condition that decides whether or not the resulting value becomes a part of the output list.

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

1. The generalized syntax:

```
values = [expression for item in collection if condition]
```

e. This new list comprehension can be transformed into an equivalent for-loop:

```
even_squares = []
for x in range(10):
    if x % 2 == 0:
        even_squares.append(x * x)
```

Dictionary comprehension:

a. `{ x: x * x for x in range(5) }`

List Slicing Tricks

a. [start:stop:step]

Consider the following list.

```
lst = [1, 2, 3, 4, 5]
```

What will be the output of lst[1:3:1].

Note: Adding the [1:3:1] index returned a slice of the original list ranging from index 1 to index 2, with a step size of one element. To avoid off-by-one errors, it’s important to remember that the upper bound is always exclusive. This is why we got [2, 3] as the sublist from the [1:3:1] slice.

b. Observe the output of the following.

```
lst[::2]
```

c. Observe the output of the following.

```
lst[::-1]
```

d.

Encouraging

1. Iterator:

```
class Repeater:
    def __init__(self, value):
        self.value = value
    def __iter__(self):
        return self
    def __next__(self):
        return self.value
```

Run the following and observe the output.

```
repeater = Repeater('Hello')
for item in repeater:
    print(item)
```

2. For iterator the following classes are also used.

```
class BoundedRepeater:
    def __init__(self, value, max_repeats):
        self.value = value
        self.max_repeats = max_repeats
        self.count = 0
    def __iter__(self):
        return self
    def __next__(self):
        if self.count >= self.max_repeats:
            raise StopIteration
        self.count += 1
        return self.value
```

Run the following and observe the output.

```
repeater = BoundedRepeater('Hello', 3)
for item in repeater:
    print(item)
```

Recommended reading