midterm-notes.py[1]

Created by Amin Hashemi

1. Expressions

- An expression is a combination of variables, operators, and values that produces a result.
- Example:

```
x = 5 + 3
y = x * 2
z = (x + y) / 2
```

2. Binary Operations

- Operations on numbers that work at the binary level.
- Example:

```
x = 5  # 101 in binary
y = 3  # 011 in binary
and_result = x & y  # 001 in binary, so result = 1
or_result = x | y   # 111 in binary, so result = 7
xor_result = x ^ y  # 110 in binary, so result = 6
left_shift = x << 1  # 1010 in binary, result = 10
right_shift = x >> 1  # 010 in binary, result = 2
```

3. Functions

- A function is a block of reusable code that performs a specific task.
- Syntax:

```
def add(a, b):
    return a + b
```

- Example:

```
def multiply(a, b):
    return a * b

result = multiply(5, 3)  # result is 15
```

4. Recursion

- A function that calls itself.
- Key components:
  - **Base Case**: Prevents infinite recursion.
  - **Recursive Case**: Reduces the problem.
- Example: Factorial of a number.

```
def factorial(n):
    if n == 0:
        return 1  # Base case
    return n * factorial(n-1)  # Recursive case

print(factorial(5))  # 120
```

5. Conditionals: if, else, elif

- `if`: Runs code if the condition is true.
- `else`: Runs code if the condition is false.
- `elif`: Checks additional conditions if the previous ones are false.
- Example:

```
x = 10
if x < 5:
    print("x is less than 5")
elif x == 10:
    print("x is 10")
else:
    print("x is greater than 5")
```

6. Loops: for, while

- `for`: Iterates over a sequence (e.g., list, range).
- `while`: Repeats code as long as a condition is true.
- Example (`for` loop):

```
for i in range(5):
    print(i)
```

- Example (`while` loop):

```
count = 0
while count < 5:
    print(count)
    count += 1
```

7. Lists

- Ordered, mutable collection of elements.
- Syntax: `list = [element1, element2, element3]`
- Example:

```
my_list = [1, 2, 3]
my_list.append(4)  # Adds 4 to the end
my_list[0] = 0  # Updates the first element
print(my_list)  # [0, 2, 3, 4]

# Loop through the list
for item in my_list:
    print(item)  # Prints each element

# List comprehension to square each number
squared = [x**2 for x in my_list]
print(squared)  # [0, 4, 9, 16]
```

- Methods:
  - `.append()`: Add an element to the end.

```
lst = [1, 2]
lst.append(3)
lst  # Output: [1, 2, 3]
```

  - `.remove()`: Remove the first occurrence of an element.

```
lst = [1, 2, 3]
lst.remove(2)
lst  # Output: [1, 3]
```

  - `.pop()`: Remove and return an element by index.

```
lst = [1, 2, 3]
lst.pop(1)  # Output: 2
lst  # Output: [1, 3]
```

  - `.sort()`: Sort the list in place.

```
lst = [3, 1, 2]
lst.sort()
lst  # Output: [1, 2, 3]
```

8. Sets

- Unordered collection of unique elements.
- Syntax: `set = element1, element2, element3`
- Example:

```
my_set = {1, 2, 3, 3}
print(my_set)  # {1, 2, 3}  # Duplicate 3 is removed

# Adding and removing elements
my_set.add(4)
my_set.remove(2)
print(my_set)  # {1, 3, 4}
```

- Methods:
  - `.add()`: Add an element.

```
s = {1, 2}
s.add(3)
s  # Output: {1, 2, 3}
```

  - `.remove()`: Remove an element.

[1] Prepared for *Fundamentals of Python Programming* at Sharif University of Technology, taught by Marzieh Sadri.

```
s = {1, 2, 3}
s.remove(2)
s  # Output: {1, 3}
```

- `.union()`: Combine two sets.

```
s1 = {1, 2}
s2 = {2, 3}
s1.union(s2)  # Output: {1, 2, 3}
```

- `.intersection()`: Find common elements.

```
s1 = {1, 2}
s2 = {2, 3}
s1.intersection(s2)  # Output: {2}
```

9. Dictionaries

- Unordered collection of key-value pairs.
- Syntax: `dict = key1:  value1, key2:  value2`
- Example:

```
my_dict = {'a': 1, 'b': 2}
print(my_dict['a'])  # 1
my_dict['c'] = 3  # Adds new key-value pair
print(my_dict)  # {'a': 1, 'b': 2, 'c': 3}

# Loop through dictionary keys and values
for key, value in my_dict.items():
    print(key, value)  # a 1, b 2, c 3
```

- Methods:
  - `.get()`: Retrieve a value, with an optional default.

```
d = {'a': 1, 'b': 2}
d.get('a')  # Output: 1
d.get('c', 0)  # Output: 0
```

  - `.keys()`: Get all keys.

```
d = {'a': 1, 'b': 2}
list(d.keys())  # Output: ['a', 'b']
```

  - `.values()`: Get all values.

```
d = {'a': 1, 'b': 2}
list(d.values())  # Output: [1, 2]
```

  - `.items()`: Get all key-value pairs as tuples.

```
d = {'a': 1, 'b': 2}
list(d.items())  # Output: [('a', 1), ('b', 2)]
```

  - `.pop()`: Remove a key and return its value.

```
d = {'a': 1, 'b': 2}
value = d.pop('a')  # Removes 'a' and returns 1
d  # Output: {'b': 2}
```

  - `.popitem()`: Remove and return the last inserted key-value pair as a tuple.

```
d = {'a': 1, 'b': 2}
pair = d.popitem()  # Removes and returns ('b', 2)
d  # Output: {'a': 1}
```

  - `.clear()`: Remove all items from the dictionary.

```
d = {'a': 1, 'b': 2}
d.clear()
d  # Output: {}
```

10. Tuples

- Immutable ordered collection of elements.
- Syntax: `tuple = (element1, element2, element3)`
- Example:

```
my_tuple = (1, 2, 3)
print(my_tuple[1])  # 2  # Accessing by index

# Unpacking a tuple
x, y, z = my_tuple
print(x, y, z)  # 1 2 3
```

- Methods:
  - `.count()`: Count occurrences of an element.

```
t = (1, 2, 2, 3)
t.count(2)  # Output: 2
```

  - `.index()`: Find the index of an element.

```
t = (1, 2, 3)
t.index(3)  # Output: 2
```

11. Strings

- A sequence of characters.
- Common operations:
  - `len(string)`: Length of string.
  - `string.upper()` or `string.lower()`: Convert to uppercase or lowercase.
  - `string.split()`: Split string into list of words.
  - `string.replace(old, new)`: Replace substrings.
- Example:

```
my_string = "Hello, world!"
```

- Methods
  - `.count()`: Count occurrences of a substring.

```
my_string = "hello world"
my_string.count("l")  # Output: 3
```

  - `.find()`: Find the index of the first occurrence of a substring.

```
my_string = "hello world"
my_string.find("o")  # Output: 4
```

  - `.index()`: Find the index of the first occurrence of a substring (raises error if not found).

```
my_string = "hello world"
my_string.index("o")  # Output: 4
```

  - `.lower()`: Convert all characters to lowercase.

```
my_string = "HELLO"
my_string.lower()  # Output: "hello"
```

  - `.upper()`: Convert all characters to uppercase.

```
my_string = "hello"
my_string.upper()  # Output: "HELLO"
```

  - `.replace()`: Replace a substring with another substring.

```
my_string = "hello world"
my_string.replace("world", "Python")
        # Output: "hello Python"
```

  - `.split()`: Split a string into a list of substrings.

```
my_string = "hello world"
my_string.split()  # Output: ['hello', 'world']
```

  - `.strip()`: Remove leading and trailing spaces.

```
my_string = "   hello world   "
my_string.strip()  # Output: "hello world"
```

  - `.join()`: Join elements of an iterable (e.g., list) into a string.

```
my_list = ['hello', 'world']
" ".join(my_list)  # Output: "hello world"
```

  - `.startswith()`: Check if the string starts with a specified substring.

```
my_string = "hello world"
my_string.startswith("hello")  # Output: True
```

  - `.endswith()`: Check if the string ends with a specified substring.

```
my_string = "hello world"
my_string.endswith("world")  # Output: True
```