

Loops and Basic Data Structures in Python

1. Loops in Python

Loops allow us to repeat instructions multiple times.

For Loops

A for loop iterates over items in a sequence:

```
for i in range(5):    # range(5) = 0,1,2,3,4
    print("Number:", i)
```

```
for i in range(5, 11):    # range(5, 11) = 5,6,7,8,9,10
    print("Number:", i)
```

While Loops

A while loop repeats while a condition is true:

```
count = 0
while count < 5:
    print("Count is:", count)
    count += 1
```

Why doesn't this count to 5?

Control Keywords

- `break` exits a loop early.
- `continue` skips the rest of the current iteration.

```
for i in range(10):
    if i == 5:
        break
    if i % 2 == 0:
        continue
    print(i)
```

Exercises (Loops)

1. Print all the odd numbers between 1 and 9.

```
for i in range(1,10):
    if i % 2 == 1:
        print(i)
```

2. Use a while loop to print numbers from 10 down to 1.

```
n = 10
while n > 0:
    print(n)
    n -= 1
```

—

2. Data Structures

Lists

Ordered, mutable (changeable) sequences:

```
numbers = [1, 2, 3]
numbers.append(4)
print(numbers[0])    # 1
```

Indexing in Python begins with 0. In the list `letters = ['a', 'b', 'c']`, 'a' is index 0, 'b' is index 1, 'c' is index 2. Use square brackets to reference them. `print(letters[1])` will print 'b'.

List Methods

Lists have built-in methods that let you modify or inspect them:

- `append(item)`: Adds `item` to the end.
- `remove(item)`: Removes the first occurrence of `item`.
- `insert(index, item)`: Inserts `item` at `index`.
- `pop()`: Removes and returns the last item.
- `sort()`: Sorts the list in place.
- `reverse()`: Reverses the list in place.

```

numbers = [2, 1, 2]
numbers.append(3)      # [2, 1, 2, 3]
numbers.remove(2)      # [1, 2, 3] (removes the first number 2 found)
numbers.insert(1, 99)  # [1, 99, 2, 3]
last = numbers.pop()   # last = 3, numbers = [1, 99, 2]
numbers.sort()         # [1, 2, 99]
numbers.reverse()      # [99, 2, 1]

```

Tuples

Ordered, immutable (unchangeable) sequences:

```

point = (2, 5)
print(point[1])    # 5

```

Tuple Methods

Tuples support:

- `count(item)`: Number of times `item` appears.
- `index(item)`: Position of first occurrence of `item`.

```

t = (1, 4, 1, 5)
print(t.count(1))    # 2
print(t.index(5))     # 3

```

Tuples cannot be changed after creation.

Dictionaries

Key-value pairs:

```

person = {"name": "Sam", "age": 16}
print(person["name"])

```

Dictionary Methods

Useful methods include:

- `keys()`: Gives all the keys.
- `values()`: Gives all the values.
- `items()`: Gives (key, value) pairs.

- `get(key, default)`: Gets value or default.
- `pop(key)`: Removes key, returns its value.
- `update(other)`: Adds pairs from another dictionary.

```
scores = {"Sam": 90, "Alex": 84}
print(scores.keys())      # dict_keys(['Sam', 'Alex'])
print(scores.get("Bob", 0)) # 0
scores.update({"Bob": 75})
```

Sets

Unordered collections of unique elements, each item can only be in the set once:

```
unique_nums = {1, 2, 2, 3}
print(unique_nums)  # {1, 2, 3}
```

Set Methods

Important methods:

- `add(item)`: Adds an item.
- `remove(item)`: Removes item (error if absent).
- `discard(item)`: Removes if present, no error if not.
- `pop()`: Removes and returns a random element.
- `union(other)`: Combines sets.
- `intersection(other)`: Common elements.
- `difference(other)`: Elements not in other.

```
primes = {2, 3}
primes.add(5)
primes.remove(2)
primes2 = {3, 5, 7}
print(primes.intersection(primes2)) # {3, 5}
print(primes.union(primes2))        # {3, 5, 7}
```

Exercises (Data Structures)

1. Create a list of your three favorite foods and print the second item.

```
foods = ["pizza", "sushi", "tacos"]
print(foods[1])
```

2. Make a dictionary for a book with keys "title" and "author". Print the author's name.

```
book = {"title": "1984", "author": "Orwell"}
print(book["author"])
```

—

3. Indexing and Slicing

Indexing lets us access individual elements inside lists, tuples, and strings.

Lists and Tuples

Indexing starts at 0 in Python:

```
numbers = [10, 20, 30, 40, 50]
print(numbers[0])    # First item: 10
print(numbers[-1])   # Last item: 50
```

Tuples are indexed the same way:

```
point = (3, 7)
print(point[0])      # 3
```

Slicing

You can take parts of a list (or string) using slicing:

```
nums = [0, 1, 2, 3, 4, 5]
print(nums[1:4])     # Items at indexes 1,2,3 => [1,2,3]
print(nums[:3])      # First 3 items: [0,1,2]
print(nums[3:])      # From index 3 to end: [3,4,5]
```

Dictionaries

Dictionaries are not accessed by position, but by key:

```
person = {"name": "Sam", "age": 16}
print(person["name"]) # Access by key
```

Exercises (Indexing)

1. Given `letters = ["a","b","c","d","e"]`, print the third element.

```
letters = ["a","b","c","d","e"]
print(letters[2])    # c
```

2. From the list `nums = [5,10,15,20,25]`, slice out `[10,15,20]`.

```
nums = [5,10,15,20,25]
print(nums[1:4])
```

—

4. Combining Loops and Data Structures

Looping Through Lists

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print("I like", fruit)
```

Looping Through Dictionaries

```
scores = {"Alice": 85, "Bob": 72, "Cara": 90}
for name, score in scores.items():
    print(name, "scored", score)
```

Looping With Sets and Tuples

```
numbers = {1, 2, 3, 4, 5}
for n in numbers:
    print(n*n)
```

```
point = (3, 4)
for coord in point:
    print(coord)
```

Exercises (Combining)

1. Print each food in your `foods` list with a message like "I want to eat pizza".

```
foods = ["pizza", "sushi", "tacos"]
for food in foods:
    print("I want to eat", food)
```

2. Given a dictionary of library books and counts, print the title of every book with more than 2 copies.

```
library = {"1984": 4, "Dune": 1, "Hobbit": 3}
for title, count in library.items():
    if count > 2:
        print(title)
```

—

5. Challenge Problems (No Solutions)

1. Write a program that goes through a list of numbers and creates a new list of only the even numbers. (Modulus)
 2. Count how many times each letter appears in a word. (Hint: every string is a list of letters)
 3. Given a list of student scores, print the average score. (Dictionary)
 4. Write a program that prints out all unique words from a sentence. (Set)
-

Expanded Study: Topics for Learning More About Data Structures

If you'd like to increase your knowledge, explore these concepts:

- **List comprehensions** for building lists quickly
- **Nested data structures** (lists of lists, etc.)
- **Advanced set operations:** `issubset`, `issuperset`, `symmetric_difference`
- The `collections` module (`defaultdict`, `Counter`, `OrderedDict`)
- Differences between mutable and immutable types
- Designing your own data structure for a project

Check the Python documentation: [Python's Data Structures Tutorial](#)

Next Lesson Preview: Functions, Libraries, and Practical Applications

Next time, we'll expand your Python toolbox:

- Defining and calling functions
- Passing parameters and using return values
- Importing and using external libraries (`math`, `time`, and more)
- Combining all concepts to build basic robotic program structures

Keep practicing—the basics make the next step easier and more creative!