Python Bytes: 30 Quick Tips for Everyday Coding

List Comprehensions

Use list comprehensions for concise and readable code.

Example:

```
squares = [x**2 for x in range(10)]
```

Enumerate

Use enumerate to get both index and value in a loop.

Example:

```
for index, value in enumerate(['a', 'b', 'c']):
   print(index, value)
```

Zip

Use zip to iterate over multiple lists in parallel.

Example:

```
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]
for name, age in zip(names, ages):
   print(f'{name} is {age} years old')
```

Defaultdict

Use defaultdict from collections to handle missing keys in a dictionary.

Example:

```
from collections import defaultdict
d = defaultdict(int)
d['key'] += 1
```

Counter

Use Counter from collections to count occurrences of elements in a list.

Example:

```
from collections import Counter
```

```
count = Counter(['a', 'b', 'a', 'c', 'b', 'a'])
print(count)
```

Namedtuple

Use namedtuple from collections for readable and self-documenting tuples.

Example:

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
print(p.x, p.y)
```

F-strings

Use f-strings for formatted string literals.

Example:

```
name = 'Alice'
age = 25
print(f'{name} is {age} years old')
```

Unpacking

Use unpacking to assign multiple variables at once.

Example:

```
x, y, z = [1, 2, 3]
```

Lambda Functions

Use lambda functions for small anonymous functions.

Example:

```
add = lambda x, y: x + y
print(add(2, 3))
```

Map

Use map to apply a function to all items in an iterable.

Example:

```
squares = list(map(lambda x: x**2, range(10)))
```

Filter

Use filter to filter items in an iterable based on a function.

Example:

```
even_numbers = list(filter(lambda x: x % 2 == 0, range(10)))
```

Reduce

Use reduce to apply a function cumulatively to the items of an iterable.

Example:

```
from functools import reduce
product = reduce(lambda x, y: x * y, [1, 2, 3, 4])
```

Any and All

Use any and all to check conditions on iterables.

Example:

```
numbers = [1, 2, 3, 4]
print(any(n > 3 for n in numbers))
print(all(n > 0 for n in numbers))
```

Set Operations

Use set operations for mathematical set operations.

Example:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 & set2)
print(set1 | set2)
```

Dictionary Comprehensions

Use dictionary comprehensions for concise dictionary creation.

Example:

```
squares = {x: x**2 for x in range(10)}
```

Generators

Use generators for memory-efficient iteration.

Example:

```
def gen():
  for i in range(10):
    yield i
for value in gen():
  print(value)
```

Decorators

Use decorators to modify the behavior of functions or methods.

Example:

```
def my_decorator(func):
  def wrapper():
    print('Something is happening before the function is called.')
    func()
    print('Something is happening after the function is called.')
  return wrapper
@my_decorator
def say_hello():
  print('Hello!')
say_hello()
```

Context Managers

Use context managers to manage resources.

Example:

```
with open('file.txt', 'r') as file:
  content = file.read()
```

Type Hinting

Use type hinting for better code readability and debugging.

Example:

```
def greeting(name: str) -> str:
  return 'Hello ' + name
```

List Slicing

Use list slicing for extracting parts of lists.

Example:

```
numbers = [1, 2, 3, 4, 5]
print(numbers[1:3])
```

List Methods

Use list methods like append, extend, and insert.

Example:

```
numbers = [1, 2, 3]
numbers.append(4)
numbers.extend([5, 6])
numbers.insert(0, 0)
```

Dictionary Methods

Use dictionary methods like get, keys, and values.

Example:

```
d = {'a': 1, 'b': 2}
print(d.get('a'))
print(d.keys())
print(d.values())
```

String Methods

Use string methods like split, join, and replace.

Example:

```
text = 'Hello World'
print(text.split())
print('-'.join(['Hello', 'World']))
print(text.replace('World', 'Python'))
```

List Sorting

Use sorted and sort for sorting lists.

Example:

```
numbers = [3, 1, 4, 1, 5, 9]
print(sorted(numbers))
numbers.sort()
print(numbers)
```

Dictionary Sorting

Use sorted for sorting dictionaries by keys or values.

Example:

```
d = {'a': 3, 'b': 1, 'c': 2}
print(sorted(d.items(), key=lambda item: item[1]))
```

Exception Handling

Use try, except, and finally for exception handling.

Example:

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print('Cannot divide by zero')
finally:
    print('This will always execute')
```

List Flattening

Use list comprehensions to flatten a list of lists.

Example:

```
nested_list = [[1, 2, 3], [4, 5, 6]]
flat_list = [item for sublist in nested_list for item in sublist]
```

Reading Files

Use open to read files.

Example:

```
with open('file.txt', 'r') as file:
    content = file.read()
```

Writing Files

Use open to write files.

Example:

```
with open('file.txt', 'w') as file:
    file.write('Hello World')
```

JSON Handling

Use json module to handle JSON data.

Example:

```
import json
data = {'name': 'Alice', 'age': 25}
json_string = json.dumps(data)
parsed_data = json.loads(json_string)
```

Bonus Quiz

1. What is the output of the following code?

```
numbers = [1, 2, 3, 4, 5]
print(numbers[1:3])
```

2. How do you handle missing keys in a dictionary using collections?

3. Write a lambda function to add two numbers.

4. How do you use a context manager to read a file?

5. What is the difference between append and extend methods in a list?