Python Control Structures

Conditional Statements

if Statement

The basic conditional statement:

```
age = 18
if age >= 18:
    print("You are an adult")
```

if-else Statement

Adding an alternative condition:

```
age = 16
if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

if-elif-else Statement

Multiple conditions:

```
score = 85
if score >= 90:
    print("A grade")
elif score >= 80:
    print("B grade")
elif score >= 70:
    print("C grade")
else:
    print("Below C grade")
```

Nested if Statements

Conditions within conditions:

```
age = 25
has_license = True

if age >= 18:
    if has_license:
        print("You can drive")
    else:
        print("You need a license")
else:
    print("Too young to drive")
```

Loops

for Loop

Iterating over sequences:

```
# Iterating over a list
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)

# Iterating over a range
for i in range(5):
    print(i)  # Prints 0 to 4

# Iterating over a string
for char in "Python":
    print(char)
```

while Loop

Repeating while a condition is true:

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

Loop Control Statements

break Statement Exits the loop:

```
for i in range(10):
    if i == 5:
        break
    print(i)  # Prints 0 to 4
```

continue Statement Skips the rest of the current iteration:

```
for i in range(5):
    if i == 2:
        continue
    print(i)  # Prints 0, 1, 3, 4
```

pass Statement Does nothing but acts as a placeholder:

```
for i in range(5):
    if i == 2:
```

```
        pass  # Do nothing
    print(i)  # Prints all numbers
```

Exception Handling

try-except

Basic error handling:

```
try:
    number = int(input("Enter a number: "))
    result = 10 / number
    print(result)
except ValueError:
    print("Please enter a valid number")
except ZeroDivisionError:
    print("Cannot divide by zero")
```

try-except-else

Adding code that runs if no exception occurs:

```
try:
    number = int(input("Enter a number: "))
except ValueError:
    print("Invalid input")
else:
    print(f"You entered {number}")
```

try-except-finally

Adding cleanup code:

```
try:
    file = open("data.txt")
    # Process file
except FileNotFoundError:
    print("File not found")
finally:
    file.close()  # Always runs
```

Match Statement (Python 3.10+)

Pattern matching:

```
command = "help"
match command:
    case "quit":
        print("Exiting...")
```

```
    case "help":
        print("Showing help...")
    case _:
        print("Unknown command")
```

Comprehensions

List Comprehension

Creating lists concisely:

```
# Traditional way
squares = []
for i in range(5):
    squares.append(i ** 2)

# Using comprehension
squares = [i ** 2 for i in range(5)]
```

Dictionary Comprehension

Creating dictionaries concisely:

```
# Traditional way
squares_dict = {}
for i in range(5):
    squares_dict[i] = i ** 2

# Using comprehension
squares_dict = {i: i ** 2 for i in range(5)}
```

Best Practices

Clean Code Guidelines

- Keep conditions simple and readable
- Avoid deep nesting
- Use meaningful variable names
- Add comments for complex logic

Optimization Tips

- Use appropriate loop types
- Break loops when possible
- Handle exceptions properly
- Use comprehensions for simple operations

Common Patterns

Enumerate

Getting index while iterating:

```
fruits = ["apple", "banana", "orange"]
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
```

Zip

Iterating over multiple sequences:

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

Context Managers (with)

Resource management:

```
with open("file.txt") as file:
    content = file.read()
    # File automatically closes
```