Cheatsheet

Useful commands for Python

Variables and Types

Variables

- Definition: Containers for storing information.
- Example: `x = 10`

Data Types

- Integers (int): Whole numbers (e.g., count of dates).
- Floats (float): Decimal numbers (e.g., compatibility score).
- Booleans (bool): True/False values (e.g., availability).
- Strings (str): Text values (e.g., names).

```
  name = "Alexander"  # String variable
  flags = 0           # Integer variable
  butterflies = True  # Boolean variable
```

Type Conversion

- Checking: Use `type()` to check the type of a variable.
- Conversion:
  - `int()`: Converts to integer.
  - `float()`: Converts to float.
  - `str()`: Converts to string.
  - `bool()`: Converts to boolean.

String Formatting

- Concatenation: Combine strings using `+`.
- Formatting: Use `f"..."` for formatted strings.

```
name = "Alexander"
print(f"Hello, {name}!")
```

```
Hello, Alexander!
```

Comparisons

Comparison Operators

Symbol	Meaning	Example
==	Equal to	score == 100
!=	Not equal to	degree != “Computer Science”
<	Less than	salary < 80000
>	Greater than	experience > 5
<=	Less than or equal to	age <= 65
>=	Greater than or equal to	test_score >= 80

Logical Operators

Symbol	Meaning	Example
and	Both conditions must be true	score > 80 and experience > 5
or	At least one condition must be true	score > 80 or experience > 5
not	Condition must be false	not (score > 80)

Decision-Making

if Statements

- Structure:

```
if condition:
    # code to execute if condition is True
```

- Example:

```
flat_rating = 8
if flat_rating >= 7:
    print("This is a good apartment!")
```

```
This is a good apartment!
```

if-else Statements

- Structure:

```
if condition:
    # code to execute if condition is True
else:
    # code to execute if condition is False
```

- Example:

```
flat_rating = 4
if flat_rating >= 7:
    print("Apply for this flat!")
else:
    print("Keep searching!")
```

```
Keep searching!
```

if-elif-else Statements

- Structure:

```
if condition:
    # code to execute if condition is True
elif condition:
    # code to execute if condition is False
else:
    # code to execute if condition is False
```

- Example:

```
flat_rating = 8
if flat_rating >= 9:
    print("Amazing flat - apply immediately!")
elif flat_rating >= 7:
    print("Good flat - consider applying")
else:
    print("Keep looking")
```

```
Good flat - consider applying
```

Complex Conditions

- Nested if Statements: Use if statements inside other if statements.
- Logical Operators: Combine conditions using `and`, `or`, `not`.
- Structure:

```
if (condition1) and (condition2):
    # code if both conditions are True
elif (condition1) or (condition2):
    # code if at least one condition is True
```

```
else:
    # code if none of the conditions are True
```

- Example:

```
flat_rating = 9
price = 900
if (flat_rating >= 9) and (price < 1000):
    print("Amazing flat - apply immediately!")
```

```
Amazing flat - apply immediately!
```

Lists and Tuples

Lists

- Definition: Ordered, mutable collections of items.
- Creation: Use square brackets `[]`.

```
ratings = [4.5, 3.8, 4.2]
restaurants = ["Magic Place", "Sushi Bar", "Coffee Shop"]
```

Accessing Elements

- Indexing: Use `[index]` to access elements.

```
print(restaurants[0])  # Access the first element
```

```
Magic Place
```

- Negative Indexing: Use `[-1]` to access the last element.

```
print(restaurants[-1])  # Access the last element
```

```
Coffee Shop
```

- Slicing: Use `[start:end]` to access a range of elements.

```
print(restaurants[0:2])  # Access the first two elements
```

```
['Magic Place', 'Sushi Bar']
```

Adding Elements

- Appending: Use `append()` to add an element to the end of the list.

```
restaurants.append("Pasta Place")
```

- Inserting: Use `insert()` to add an element at a specific index.

```
restaurants.insert(0, "Pasta Magic")
```

Removing Elements

- Removing: Use `remove()` to remove an element by value.

```
restaurants.remove("Pasta Place")
```

- Removing by Index: Use `pop()` to remove an element by index.

```
restaurants.pop(0)
```

```
'Pasta Magic'
```

Nested Lists

- Definition: Lists containing other lists or tuples.
- Accessing: Use nested indexing.

```
restaurant_data = [
    ["Pasta Place", 4.5, 3],
    ["Sushi Bar", 4.2, 1]
]
print(restaurants[0][1])  # Access the second element of the first list
```

```
a
```

Tuples

- Definition: Ordered, immutable collections of items.
- Creation: Use parentheses `()`.
- Immutability: Once created, cannot be changed.
- Memory Efficiency: Use less memory than lists.
- Use Cases: Ideal for fixed data (e.g., restaurant location).

```
ratings = (4.5, 3.8, 4.2)
restaurant_info = ("Pasta Place", "Italian", 2020)
```

Loops

for Loops

- Definition: Iterate over a sequence of items.
- Structure:

```python
for item in sequence:
    # code to execute for each item
```

- Example:

```python
treatments = ["Standard Drug", "New Drug A", "New Drug B"]
for treatment in treatments:
    print(f"Evaluating efficacy of {treatment}")
```

```
Evaluating efficacy of Standard Drug
Evaluating efficacy of New Drug A
Evaluating efficacy of New Drug B
```

Range in for Loops

- Definition: Generate a sequence of numbers.
- Structure:

```python
range(start, stop, step)
```

- Example:

```python
for phase in range(5):  # 0 to 4
    print(f"Starting Phase {phase + 1}")
```

```
Starting Phase 1
Starting Phase 2
Starting Phase 3
Starting Phase 4
Starting Phase 5
```

```python
for phase in range(1, 5):  # 1 to 4
    print(f"Starting Phase {phase}")
```

```
Starting Phase 1
Starting Phase 2
Starting Phase 3
Starting Phase 4
```

```python
for phase in range(1, 5, 2):  # 1 to 4, step 2
    print(f"Starting Phase {phase}")
```

```
Starting Phase 1
Starting Phase 3
```

break and continue

- break: Exit the loop.
- continue: Skip the current iteration and continue with the next.

```python
efficacy_scores = [45, 60, 75, 85, 90]
for score in efficacy_scores:
    if score < 50:
        continue
        print(f"Treatment efficacy: {score}%")
    if score >= 85:
        break
```

Tuple unpacking

- Definition: Assign elements of a tuple to variables.
- Structure:
- Example:

```python
restaurant_info = ("Pasta Place", "Italian", 2020)
name, cuisine, year = restaurant_info
print(name)
print(cuisine)
print(year)
```

```
Pasta Place
Italian
2020
```

while Loops

- Definition: Execute code repeatedly as long as a condition is true.
- Structure:

```python
while condition:
    # code to execute while condition is True
```

- Example:

```python
phase = 1
while phase <= 5:
    print(f"Starting Phase {phase}")
    phase += 1
```

```
Starting Phase 1
Starting Phase 2
Starting Phase 3
Starting Phase 4
Starting Phase 5
```

Functions

Basic Function

- Definition: Use the `def` keyword.
- Structure:

```
def function_name(parameters):
    # code to execute (function body)
    return value  # Optional
```

- Example:

```
def greet_visitor(name):
    return f"Welcome to the library, {name}!"

greet_visitor("Student")
```

```
'Welcome to the library, Student!'
```

Return Value

- Definition: The value returned by a function.
- Example:

```
def multiply_by_two(number):
    return number * 2

result = multiply_by_two(5)
print(result)
```

```
10
```

- Note: If a function does not return a value, it implicitly returns `None`.

Default Parameters

- Definition: Provide default values for function parameters.
- Structure:

```python
def greet_visitor(name="People"):
    return f"Welcome to the library, {name}!"

print(greet_visitor()) # Calls the function with the default parameter
print(greet_visitor("Tobias")) # Calls the function with a custom parameter
```

Multiple Parameters

- Definition: Functions can have multiple parameters.
- Structure:

```python
def greet_visitor(name, age):
    return f"Welcome to the library, {name}! You are {age} years old."

print(greet_visitor("Tobias", 30))
```

String Methods

- Definition: Methods are functions that are called on strings.
- Structure:

```python
string.method()
```

- Common String Methods:
  - `.strip()` - Removes whitespace from start and end
  - `.title()` - Capitalizes first letter of each word
  - `.lower()` - Converts to lowercase
  - `.upper()` - Converts to uppercase
- Example:

```python
title = "the hitchhikers guide"
print(title.title())
```

```
The Hitchhikers Guide
```

```python
title = "   the hitchhikers guide    "
print(title.strip())
```

```
the hitchhikers guide
```

Packages

Standard Libraries

- Definition: Libraries that are part of the Python standard library.
- Access: Import them using `import`.

```python
import math
import random
```

- For long package names, you can use the `as` keyword to create an alias.

```python
import random as rd
```

- To call a function from an imported package, use the package name as a prefix.

```python
random_number = rd.random()
print(random_number)
```

```
0.09946579870799288
```

Installing Packages

- Definition: Install packages using `uv`. Note, don't do this inside of a notebook but in the terminal in your project folder!

```
{bash}
uv add package_name
```

Probability Distributions

Normal Distribution

- When to Use: Most common in business and nature; symmetric outcomes around a mean
- Characteristics:
  - Bell-shaped, symmetric curve
  - Most values cluster around the mean
  - Rare extreme values in tails
- Examples:
  - Investment returns
  - Manufacturing variations
  - Employee performance scores
  - Measurement errors

Python Syntax:

```python
import numpy as np

# Generate normal distribution
returns = np.random.normal(loc=mean, scale=std_dev, size=n_samples)
```

```python
# Example: Stock returns with 10% mean, 15% volatility
stock_returns = np.random.normal(loc=0.10, scale=0.15, size=10000)
```

Parameters:

- `loc`: The mean (center) of the distribution
- `scale`: The standard deviation (spread)
- `size`: Number of samples to generate

Uniform Distribution

- When to Use: Complete uncertainty within a range; all outcomes equally likely
- Characteristics:
  - Flat distribution
  - All values equally likely
  - Hard boundaries (min/max)
  - No clustering around any value
- Examples:
  - Random wait times
  - Initial demand estimates with only min/max known
  - Random sampling from a range

Python Syntax:

```python
# Generate uniform distribution
values = np.random.uniform(low=minimum, high=maximum, size=n_samples)

# Example: Demand between 1000 and 5000 units
demand = np.random.uniform(low=1000, high=5000, size=10000)
```

Parameters:

- `low`: Minimum value (inclusive)
- `high`: Maximum value (exclusive)
- `size`: Number of samples to generate

Exponential Distribution

- When to Use: Time between events; waiting times
- Characteristics:
  - Many small values, few large ones
  - Always positive
  - Memoryless property
  - Right-skewed (long tail)
- Examples:
  - Time between customer arrivals
  - Equipment failure times
  - Time until next sale

    - Duration of phone calls

Python Syntax:

```python
# Generate exponential distribution
wait_times = np.random.exponential(scale=average_time, size=n_samples)

# Example: Time between customers (avg 5 minutes)
arrivals = np.random.exponential(scale=5, size=10000)
```

Parameters:

- `scale`: The average (mean) time between events
- `size`: Number of samples to generate

Binomial Distribution

- When to Use: Fixed number of independent yes/no trials
- Characteristics:
    - Discrete outcomes (counts)
    - Fixed number of trials
    - Each trial has same probability
    - Trials are independent
- Examples:
    - Number of defective items in a batch
    - Number of successful sales calls
    - Number of customers who convert
    - Number of loans that default

Python Syntax:

```python
# Generate binomial distribution
successes = np.random.binomial(n=n_trials, p=prob_success, size=n_samples)

# Example: 100 sales calls with 20% conversion rate
conversions = np.random.binomial(n=100, p=0.20, size=10000)
```

Parameters:

- `n`: Number of trials
- `p`: Probability of success on each trial
- `size`: Number of experiments to simulate

Common Risk Metrics

Calculate from simulated results:

```python
# Basic statistics
mean_return = results.mean()
std_dev = results.std()
```

```python
min_value = results.min()
max_value = results.max()

# Percentiles (Value at Risk)
var_5 = np.percentile(results, 5)  # 5th percentile (worst 5%)
var_95 = np.percentile(results, 95)  # 95th percentile (best 5%)

# Probability of loss
prob_loss = (results < 0).mean()

# Expected shortfall (average of worst 5%)
worst_5_percent = results[results <= var_5]
expected_shortfall = worst_5_percent.mean()

# Correlation between two variables
correlation = np.corrcoef(returns1, returns2)[0, 1]
```

Bibliography