Quick Reference

Type	Example	Type	Example
int	42	float	3.14
str	"hello"	bool	True
list	[1, 2, 3]	tuple	(1, 2, 3)
dict	{"a": 1}	set	{1, 2, 3}

1 Basic Syntax

Variable Assignment

```
name = value
x, y = 1, 2
a = b = c = 0
```

Function Definition

```
def func(args):
return value
def func(a, b=0, *args):
```

2 Control Structures

2.1 If Statement

```
if condition:
    statement
elif other_condition:
    statement
else:
    statement
```

2.2 For Loop

```
for item in iterable:
    statement

for i in range(10):
    print(i)
```

2.3 While Loop

```
while condition:
    statement
    if break_condition:
        break
```

2.4 List Comprehension

```
 Basic
[x for x in range(10)]

\# With condition
[x for x in range(10) if x % 2 == 0]

\# Nested
[(x, y) for x in [1,2] for y in [3,4]]
```

3 Data Structures

3.1 Lists

- `list.append(x)` - Add item to end
- `list.insert(i, x)` - Insert at position
- `list.remove(x)` - Remove first occurrence
- `list.pop([i])` - Remove and return item
- `list.sort()` - Sort in place
- `list.reverse()` - Reverse in place

3.2 Dictionaries

- `dict.get(key, default)` - Get with default
- `dict.update(other)` - Update with other dict
- `dict.pop(key)` - Remove and return value
- `dict.keys()` - Get all keys
- `dict.values()` - Get all values
- `dict.items()` - Get key-value pairs

4 Common Methods

4.1 String Methods

`str.split(sep)` *Split string by separator* `str.join(iterable)` *Join iterable with string* `str.strip()` *Remove whitespace* `str.replace(old, new)` *Replace substring* `str.startswith(prefix)` *Check prefix* `str.endswith(suffix)` *Check suffix* `str.upper()` *Convert to uppercase* `str.lower()` *Convert to lowercase*

4.2 List Methods

`list.append(x)` *Add item to end* `list.extend(iterable)` *Extend with iterable* `list.insert(i, x)` *Insert at index* `list.remove(x)` *Remove first occurrence* `list.pop([i])` *Remove and return item* `list.index(x)` *Find index of item* `list.count(x)` *Count occurrences* `list.sort()` *Sort in place*

5 Key Concepts

List vs Tuple
Lists: Mutable, use `[]`
Tuples: Immutable, use `()`
Tuples are faster and use less memory

Shallow vs Deep Copy
Shallow: `list.copy()`
Deep: `copy.deepcopy()`
Deep copy creates new nested objects

Generator Expressions
`(x for x in range(10))`
Memory efficient, lazy evaluation
Use parentheses, not brackets

Context Managers
`with open('file.txt') as f:`
Automatic resource management
Ensures cleanup even with exceptions

6 Common Patterns

6.1 Error Handling

```python
try:
    risky_operation()
except ValueError as e:
    handle_error(e)
except Exception as e:
    handle_generic(e)
finally:
    cleanup()
```

6.2 File Operations

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

\# Write file
with open('file.txt', 'w') as f:
    f.write('content')
```

6.3 Lambda Functions

```python
 Simple lambda
lambda x: x * 2

\# With filter
filter(lambda x: x > 0, numbers)

\# With map
map(lambda x: x**2, numbers)
```

6.4 Decorators

```python
def decorator(func):
    def wrapper(*args, **kwargs):
        \# Do something before
        result = func(*args, **kwargs)
        \# Do something after
        return result
    return wrapper

@decorator
def function():
    pass
```