Variables and data types

```
# Variable declaration
variable_name = value

# Basic data types
integer = 5
float_number = 5.5
string = "Hello world!"
boolean = True
none = None
```

Operators

```
# Arithmetic operators
addition = 5 + 2
subtraction = 5 - 2

multiplication = 5 * 2
division = 5 / 2
floor_division = 5 // 2

exponentiation = 5 ** 2
modulus = 5 % 2

# Logical operators
and_operator = True and False
or_operator = True or False
not_operator = not True
```

Operators

```
# Comparison operators
greater_than = 5 > 2
less_than = 5 < 2

equal_to = 5 == 2
not_equal_to = 5 != 2

greater_than_or_equal_to  =  5  >=
2 less_than_or_equal_to = 5 <= 2
```

Conditional statements

```
# if statement
if condition:
 # code block

# if-else statement
if condition:
 # code block
else:
 # code block

# if-elif-else statement
if condition:
 # code block
elif condition:
 # code block
else:
 # code block
```

Loops

```
# for loop
for       variable       in
sequence:   # code block

# while loop
while condition:
 # code block
```

Functions

```
# Function declaration
def                 function_name(parameter1,
parameter2):  # code block
 return result

# Function call
function_name(argument1, argument2)
```

Lists

```
# List declaration
list_name = [item1, item2, item3]

# Accessing list items
first_item = list_name[0]
last_item = list_name[-1]

# Modifying list items
list_name[0] = new_item

# Adding items to a list
list_name.append(new_item)
list_name.insert(index, new_item)

# Removing items from a list
list_name.remove(item)
list_name.pop(index)

# List slicing
sliced_list =
list_name[start_index:end_index:step]
```

Dictionaries

```
# Dictionary declaration
dictionary_name = {"k1": v1, "k2": v2}

# Accessing dictionary values
value1 = dictionary_name["k1"]

# Modifying dictionary values
dictionary_name["k1"] = new_value

# Adding items to a dictionary
dictionary_name["new_key"] = new_value

# Removing items from a dictionary
del dictionary_name["key"]

# Getting keys and values from a
dictionary
keys = dictionary_name.keys()
values = dictionary_name.values()
```

Strings

```
# String declaration
string_name = "Hello world!"

# String concatenation
concatenated_string = "Hello" + "world!"

# String interpolation
name = "John"
greeting = f"Hello, {name}!"
```

```
# String methods
string_length = len(string_name)
uppercase_string = string_name.upper()
lowercase_string = string_name.lower()
```

Tuples

```
# Tuple declaration
tuple_name = (item1, item2, item3)

# Accessing tuple items
first_item = tuple_name[0]
last_item = tuple_name[-1]

# Tuple slicing
sliced_tuple =
tuple_name[start_index:end_index:step]
```

Sets

```
# Set declaration
set_name = {item1, item2, item3}

# Adding items to a set
set_name.add(new_item)

# Removing items from a set
set_name.remove(item)

# Set operations
union_set = set1.union(set2)
intersection_set = set1.intersection(set2)
difference_set = set1.difference(set2)
```

List Comprehensions

```
# Creating a new list from an existing
list
new_list = [expression for item in
existing_list]

# Conditionally creating a new list from
an existing list
new_list = [expression for item in
existing_list if condition]
```

Error Handling

```
# try-except block
try:
 # code block
except ErrorType:
 # code block

# try-except-else block
try:
 # code block
except ErrorType:
 # code block
else:
 # code block

# try-except-finally block
try:
 # code block
except ErrorType:
 # code block
finally:
 # code block
```

Modules and Packages

```
# Importing a module
import module_name

# Importing a specific function from a
module
from module_name import function_name

# Importing all functions from a module
from module_name import *

# Importing a package
import package_name

# Importing a specific module from a
package
from package_name import
module_name
# Importing a specific function from a
module in a package
from package_name.module_name import
function_name
```

Classes and Objects

```
# Class declaration
class ClassName:
 def __init__(self, parameter1,
parameter2):
    self.parameter1    =    parameter1
 self.parameter2 = parameter2

   def   method_name(self):
 # code block

# Object creation
object_name                          =
ClassName(argument1, argument2)

# Accessing object properties
property_value = object_name.property_name

# Calling object methods
object_name.method_name()
```

Inheritance

```
# Parent class
class ParentClass:
   def   parent_method(self):
 # code block

# Child class
class
ChildClass(ParentClass):   def
child_method(self):
 # code block
# Object creation
object_name = ChildClass()

# Accessing inherited methods
object_name.parent_method()
```

Polymorphism

```
# Parent class
class ParentClass:
    def   polymorphic_method(self):
  # code block

# Child class 1
class ChildClass1(ParentClass):
    def   polymorphic_method(self):
  # code block

# Child class 2
class ChildClass2(ParentClass):
    def   polymorphic_method(self):
  # code block

# Object creation
object1 = ChildClass1()
object2 = ChildClass2()

# Polymorphic method calls
object1.polymorphic_method()
object2.polymorphic_method()
```

Lambda Functions

```
# Lambda function declaration
lambda_function = lambda p1,p2: expression

# Lambda function call
result = lambda_function(a1, a2)
```

Map, Filter, and Reduce

```
# Map function
new_list = map(function, iterable)

# Filter function
new_list = filter(function, iterable)

# Reduce function
from functools import reduce
result = reduce(function, iterable)
```

Decorators

```
# Decorator function
def decorator_function(original_function):
 def wrapper_function(*args, **kwargs):
 # code before original function
  result  =  original_function(*args,  **kwargs)    #
code after original function
 return result
 return wrapper_function

# Applying a decorator to a function
@decorator_function
def             original_function(*args,
**kwargs):  # code block
```

Generators

```
# Generator function
def generator_function():
  for i in range(10):
 yield i

# Using a generator
for                value                in
generator_function():      #   code
block
```

File Handling

```
# Opening a file
file = open("filename", "mode")

# Reading from a file
file_contents = file.read()

# Writing to a file
file.write("text")

# Closing a file
file.close()
```

Virtual Environments

```
# Creating a virtual environment
python -m venv virtual_environment_name

# Activating a virtual environment
source virtual_environment_name/bin/activate

# Installing packages in a virtual environment
pip install package_name

# Deactivating a virtual environment
deactivate
```

Context Managers

```
# Context manager class
class ContextManagerClass:
 def __enter__(self):
 # code block
 return value

  def  __exit__(self,  exc_type,  exc_value,
traceback):
 # code block

# Using a context manager with 'with' statement
with ContextManagerClass() as value:
 # code block
```

Threading and Multiprocessing

```
import threading
import multiprocessing

# Threading
thread                                                =
threading.Thread(target=function_name,    args=
(argument1, argument2))
thread.start()
# Multiprocessing
process =
multiprocessing.Process(target=function_name,
args=(argument1, argument2))
process.start()
```