```
integer = 123
```

```
print(True and False)
```

PYTHON

CHEAT SHEET

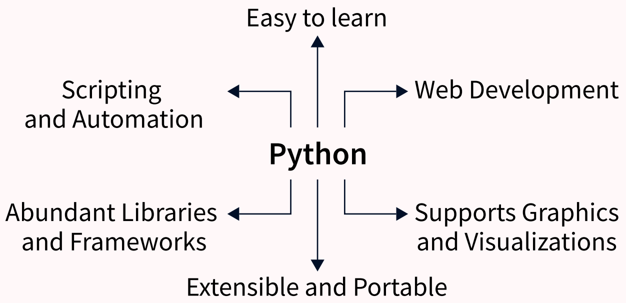
SCALER Topics

Python 3.11

```
Output: True
```

Introduction

Why Python?



Python

Easy to learn

Web Development

Supports Graphics and Visualizations

Extensible and Portable

Abundant Libraries and Frameworks

Scripting and Automation

Python Versions

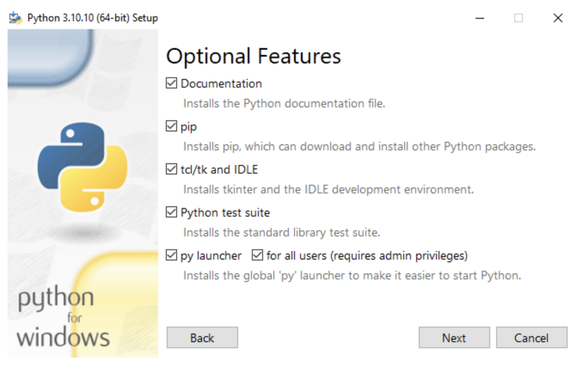
Python 0.9.0 →	Feb 1991
Python 1.0 →	Jan 1994
Python 2.0 →	Oct 2000
Python 2.7.0 →	Jul 2010
Python 3 →	Dec 2008
Python 3.6 →	Dec 2016
Python 3.6.5 →	Mar 2018
Python 3.7.0 →	May 2018
Python 3.8 →	Oct 2019
Python 3.11 →	Oct 2022

Environment Setup

Download Windows Python Installer from https://www.python.org/downloads/ → Run the Executable Installer →

1. Select the Install launcher for all users checkbox
2. Select the Add python.exe to PATH checkbox

Customize installation



Optional Features

☑ Documentation
Installs the Python documentation file.

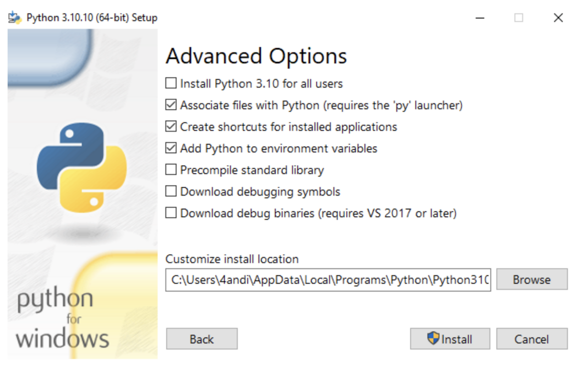
☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires admin privileges)
Installs the global 'py' launcher to make it easier to start Python.

Next



Advanced Options

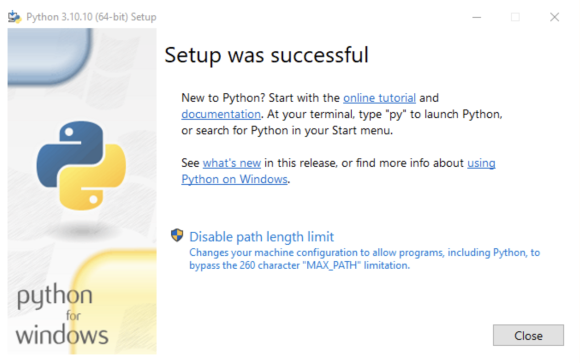
☐ Install Python 3.10 for all users
☑ Associate files with Python (requires the 'py' launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☐ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2017 or later)

Customize install location

C:\Users\4andi\AppData\Local\Programs\Python\Python31(

Install

Install Now
(Happy with default features)



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Verify Installation

```
# Open Command Prompt and run this command.
$ python --version

# Output
3.10
```

Basics

Python Syntax and Comments

Code

Lines prefixed with are ignored by the compiler

```
# This is a comment
print("Hello, world!")
```

Output

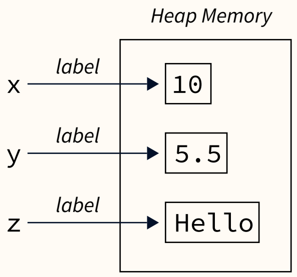
```
Hello, world!
```

Python Variables

Code

```
x = 10
y = 5.5
z = "Hello"
print(x, y, z)
```

Variables are like lables to containers in memory which hold values inside themselves.



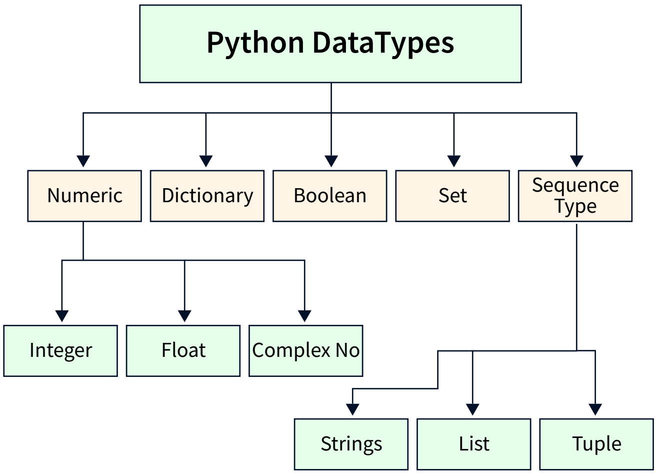
Output

```
10 5.5 Hello
```

Python Data Types

Code

```
integer = 123
float_num = 12.3
complex_num = complex(2, 3) # 2+3j

string = "abc"
list_var = [1, "a", 2.2]
tuple_var = (1, "a", 2.2)

dict_var = {"key": "value"}
set_var = {1, 2, 3}
bool_var = True
```



Operators

Arithmetic Operators

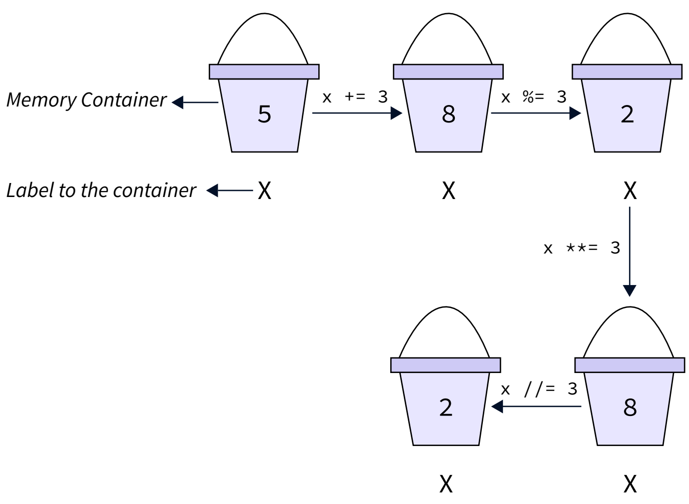
Code

```
print(5 + 3)  # Addition,Output: 8
print(5 - 3)  # Subtraction,Output: 2
print(5 * 3)  # Multiplication,Output: 15
print(5 / 3)  # Division, Output:1.6666666666666667

print(5 % 3)  # Modulo, Output: 2
print(5 ** 3)  # Exponentiation, Output: 125
print(5 // 3)  # Floor division, Output: 1
```

Assignment Operators

Code

```
x = 5

x += 3  # Equivalent to x = x + 3
x %= 3  # Equivalent to x = x % 3
x **= 3 # Equivalent to x = x ** 3
x //= 3 # Equivalent to x = x // 3

print(x)  # Output: 2
```



Comparison Operators

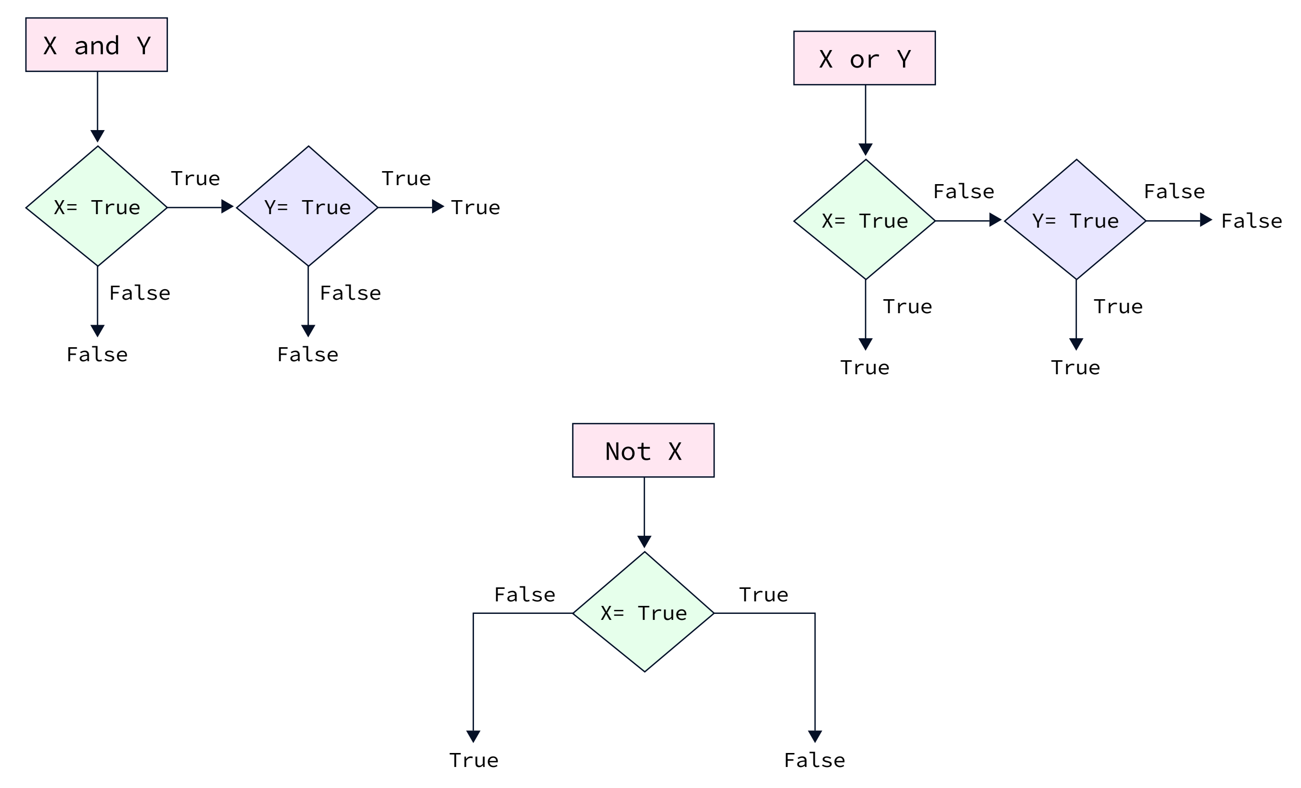
Code

```
print(5   3)  # Equal, Output: False
print(5   3)  # Not equal, Output: True

print(5 > 3)  # Greater than, Output: True
print(5 < 3)  # Less than, Output: False

print(5    3)  # Greater than or equal to, Output: True
print(5    3)  # Less than or equal to, Output: False
```

Logical Operators (and/or/not)

Code

```
print(True and False)  # Logical AND, Output: False
print(True or False)  # Logical OR, Output: True
print(not True)  # Logical NOT, Output: False
```



Identity Operators

Code

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)  # Output: True
print(x is y)  # Output: False

print(x==y)  # Output: True
```

Identity operator compares the objects rather than their values. Both objects should have same memory location.

The == operator compares the values of the variables.

Membership Operators

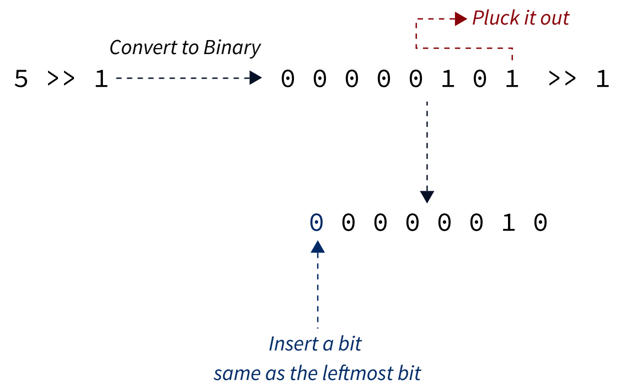
Code

```
x = 'Hello world'

print('H' in x)           # Output: True
print('hello' not in x)  # Output: True

x = ['grape', 'mango', 'banana']

print('grape' in x)      # Output: True
print('man' in x)        # Output: False
```

Bitwise Operators

Code

```
print(5 & 3)  # Bitwise AND,Output: 1
print(5 | 3)  # Bitwise OR,Output: 7
print(5 ^ 3)  # Bitwise XOR,Output: 6

print(~5)  # Bitwise NOT, Output: -6
print(5 >> 1)  # Bitwise Right Shift,
Output: 2
print(5 << 1)  # Bitwise Left Shift,
Output: 10
```



Control Flow

If...Else

```
age = 22
if (age < 18): [Condition False]
  print("Teenager!")
else:
  print("Adult!")
```

If..Elif..Else

```
age = 22
if (age < 12): [Condition False]
   print("Child")
elif (age < 18): [Condition False]
   print("Teenager")
elif (age < 40): [Condition True]
   print("Adult")
else: [Ignored]
   print("Old age")
print("End");
```

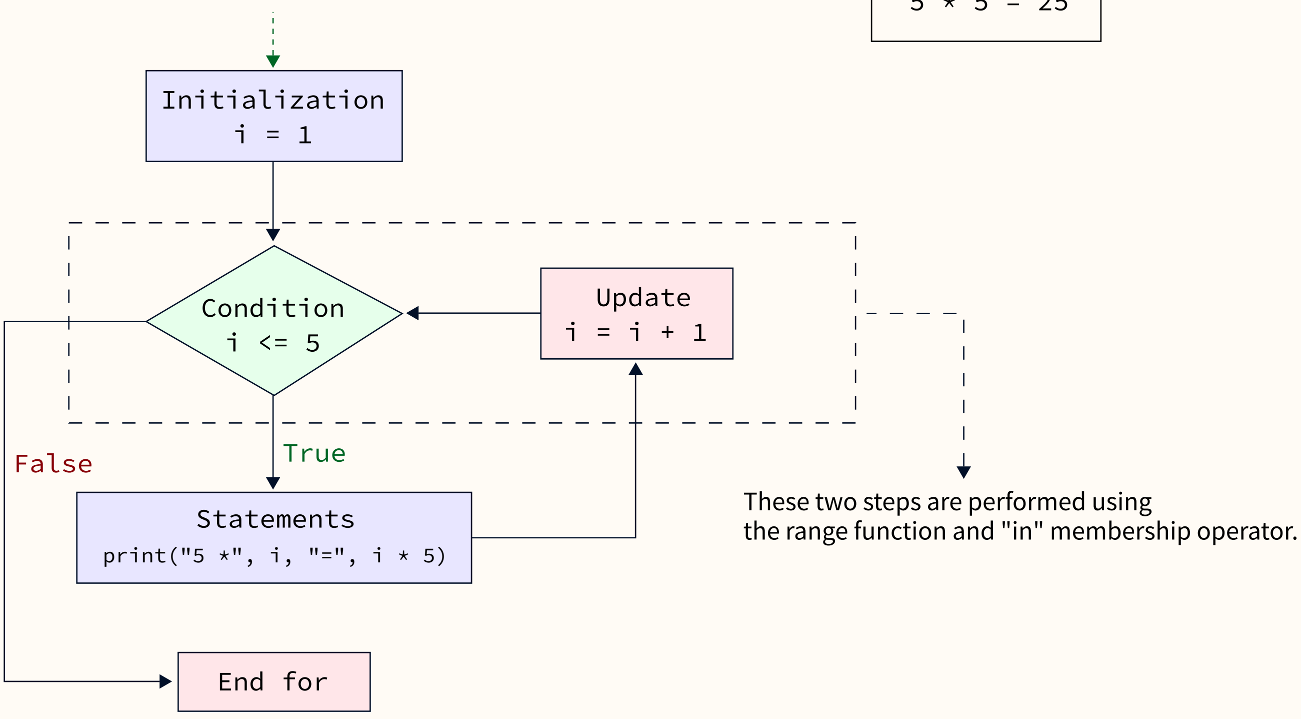
for loop

Code

```
for i in range(1, 6):
    print("5 *", i, "=", i * 5)
```

Output

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```



These two steps are performed using the range function and "in" membership operator.

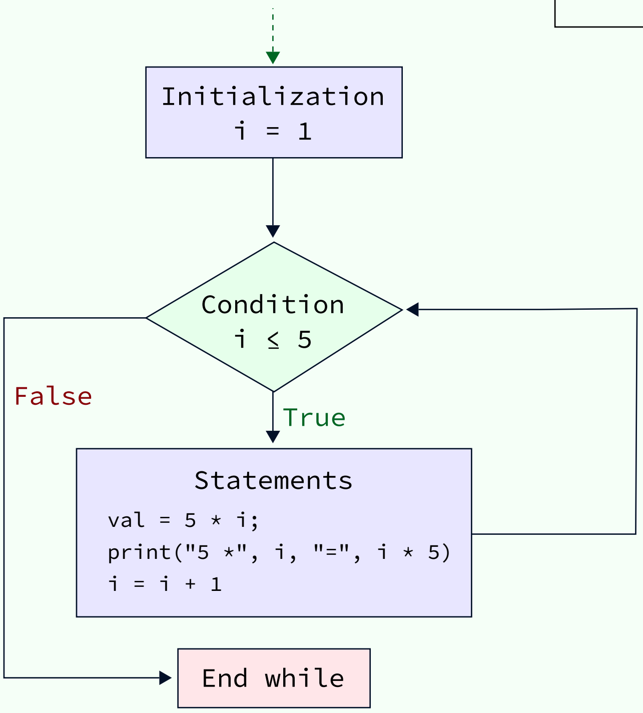
While loop

Code

```
i = 1
while i <= 5:
    val = 5 * i
    print("5 *", i, "=", i * 5)

    i = i + 1
```

Output

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```



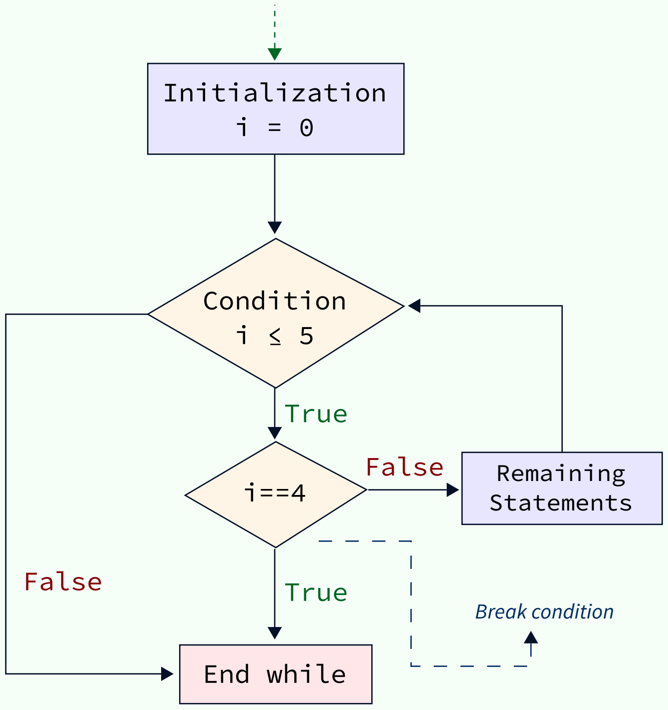
Break and Continue

Code

```
i = 0
while i <= 5:

    if (i == 4):
        break

    print(i)
    i = i + 1
```

Output

```
0
1
2
3
```

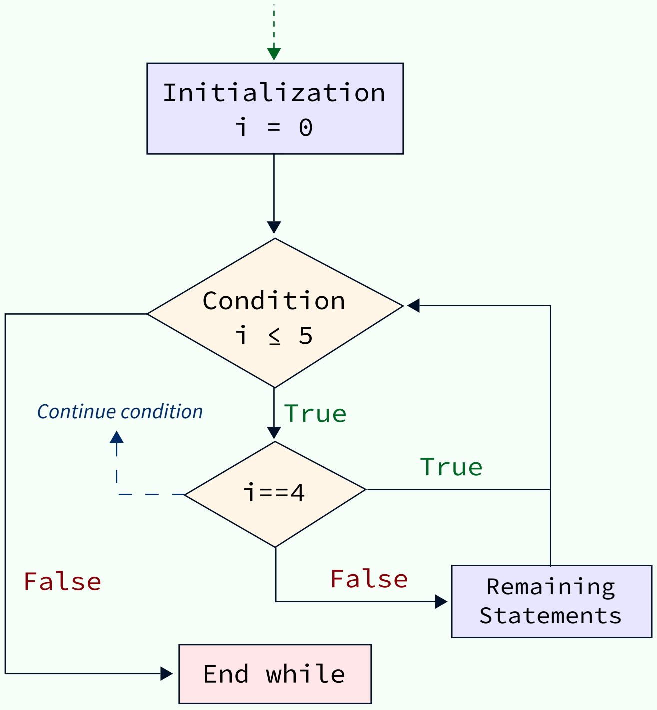


Code

```
i = 0
while i <= 5:

    if (i == 4):
        continue

    print(i)
    i = i + 1
```

Output

```
0
1
2
3
5
```



pass statement

Code

```
for i in range(5):
    pass
```

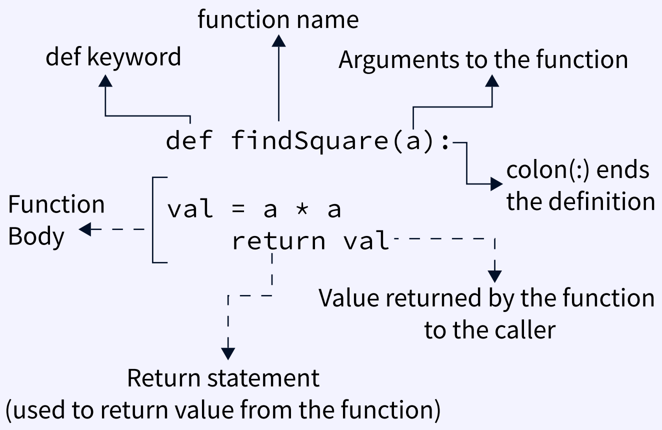
Output

Nothing will happen, it's a null operation

Functions

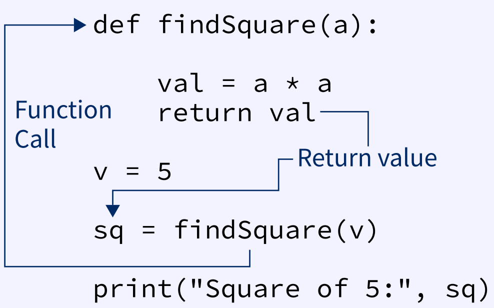
Defining a Function

Code



```
def findSquare(a):
    val = a * a
    return val
```

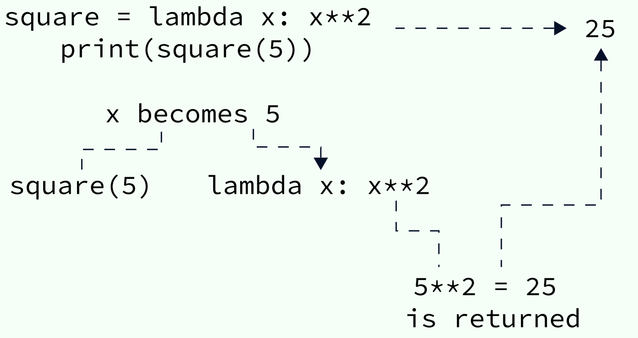
function name
def keyword
Arguments to the function
colon(:) ends the definition
Function Body
Value returned by the function to the caller
Return statement
(used to return value from the function)



```
def findSquare(a):

    val = a * a
    return val

v = 5

sq = findSquare(v)

print("Square of 5:", sq)
```

Lambda Functions

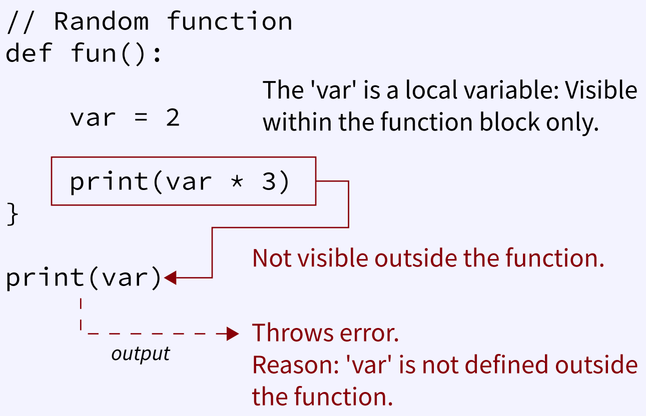
Code



```
square = lambda x: x**2
    print(square(5))
```

Scope of Variables

Code: Local Variables



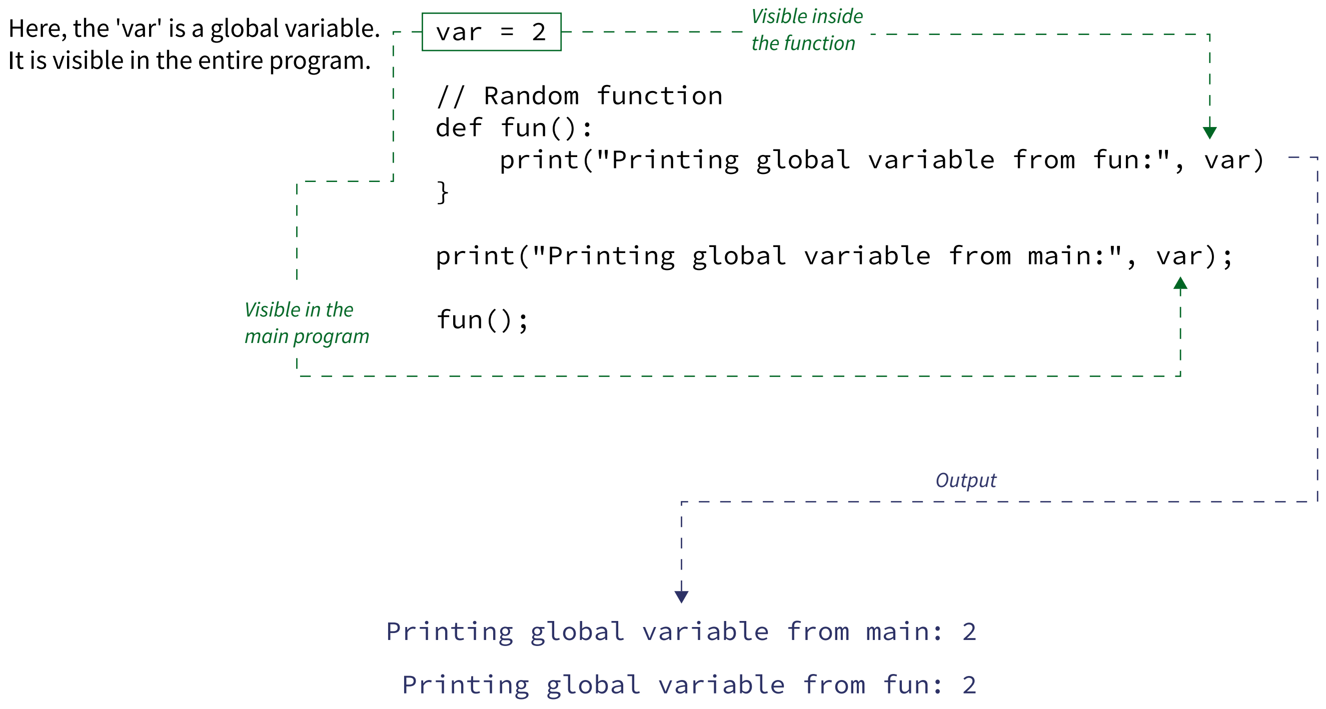
```
// Random function
def fun():

    var = 2

    print(var * 3)
}

print(var)
```

The 'var' is a local variable: Visible within the function block only.

Not visible outside the function.

Throws error.
Reason: 'var' is not defined outside the function.

Code: Global Variables



Here, the 'var' is a global variable. It is visible in the entire program.

```
var = 2

// Random function
def fun():
    print("Printing global variable from fun:", var)
}

print("Printing global variable from main:", var);

fun();
```

```
Printing global variable from main: 2
 Printing global variable from fun: 2
```

Modules

Importing a Module

Code

```
import math
print(math.sqrt(16))
```

Outputs all the defined names in the math module

from...import Statement

Code

```
from math import sqrt
print(sqrt(16))
```

Name of the library from where you want to import

Function name which you want to import

from keyword mentions the library from which you want to import something

dir() Function

Code

```
import math
print(dir(math))
```

Outputs all the defined names in the math module

Output

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

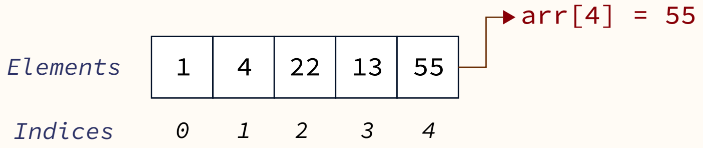
Data Structures

Lists

Declaration

```
Method 1: Create an empty list
arr = []

Method 2: Create a list with 5 elements.
arr = [3, 5, 1, 2, 3]
```

```
arr = [1, 4, 21, 13, 55]
```



Create a list:

```
arr = []
```

Add elements:

```
arr.append(1)
arr.append(4)
arr.append(22)
arr.append(13)
arr.append(55)

print(arr) # [1, 4, 22, 13, 55]
```

Remove elements:

```
arr.pop()    # Removes the last
element
arr.pop(2)   # Removes the element as index 2.

print(arr) # [1, 4, 13]
```

Slice a list:

```
print(arr[1:])  # Returns all elements from index 1 to the last.
print(arr[:2])
# Returns all elements from the start till index 2 (exclusive)
print(arr[1:3])
# Returns all elements from index 1 to index 3 (exclusive)

# Output:
[4, 13]
[1, 4]
[4, 13]
```

Miscellaneous:

```
c = arr.count(2)  # Counts the elements with value = 2.

arr.insert(3, 5)  # Inserts the element 5 at index = 3
print(arr)

arr.clear()  # Removes all elements from the list.
print(arr)
```

Tuples

In Python, a tuple is like a list but can't be changed once created.
It can hold a bunch of items in a specific order. Unlike a list, a tuple can be used to calculate hash values.

Declaration

```
Method 1: Create an empty tuple
tup = ()

Method 2: Create a tuple with 3 elements.
tup = [3, 5, 1]
```

Create a tuple:

```
tup = (3, 5, 1, 8)
```

Add elements:

Not possible as tuples are immutable.

Remove elements:

Again not possible as tuples are immutable.

Slice a tuple:

```
print(tup[1:])   # Returns all elements from index 1 to the last.
print(tup[:2])   # Returns all elements from the start till index 2 (exclusive)
print(tup[1:3])  # Returns all elements from index 1 to index 3 (exclusive)

# Output:
(5, 1, 8)
(3, 5)
(5, 1)
```

Miscellaneous:

```
c = tup.count(2)  # Counts the elements with value = 2.
print(c)

ind = tup.index(3)  # Tell the index of 3 in the tuple
print(ind)

print(min(tup))
print(max(tup))
print(sum(tup))  # Self explanatory

# Output
0
0
1
8
17
```

Set

Sets are unordered. Set elements are unique. Duplicate elements are not allowed.
A set itself may be modified, but the elements contained in the set must be of an immutable type.

Declaration

```
Method 1: Create an empty set
st = set()

Method 2: Create a set with 5 elements.
st = {4, 5, 1, 6, 7}
```

Create a set:

```
st = {4, 5, 1, 6, 7}
```

Add elements:

```
st.add(5)
st.add(9)
st.add(10)
st.add(16)

print(st)

# Output
{1, 4, 5, 6, 7, 9, 10, 16}
```

Remove elements:

```
st.remove(5)
# Remove 5,throws KeyError if not present
print(st)

st.discard(10)
# Removes 5 if present, no exception thrown
print(st)

st.pop()
# Removes a random element from the set
print(st)

# Output
{1, 4, 6, 7, 9, 10, 16}
{1, 4, 6, 7, 9, 16}
{4, 6, 7, 9, 16}
```

Miscellaneous:

```
print(len(st))  # Output: 5

st_copy = st.copy()
print(st_copy) # {16, 4, 6, 7, 9}

st.clear()
print(st)  # Output: set()
```

Dictionary

In Python, a dictionary is an unordered collection of key-value pairs, where each key is unique within the dictionary. It is also known as an associative array or a hash map in other programming languages.

Declaration

Method 1: Create an empty dictionary

```
st = {}
```

Method 2: Create a dictionary with 2 key-value pairs.

```
dt = {"apple": "green", "banana": "yellow"}
```

Create a dictionary:

```
dt = {
    "apple": "green",
    "banana": "yellow",
    "pear": "pink",
    "Lemon": "lime"
}
```

Add elements:

```
dt["orange"] = "orange"
dt.update({"pomegranate": "red"})


print(dt)

# Output
{
    "apple": "green",
    "banana": "yellow",
    "pear": "pink",
    "Lemon": "lime",
    "orange": "orange",
    "pomegranate": "red",
}
```

Remove elements:

```
del dt["apple"]
# Deletes the dict item with key = "apple"
removed_value = dt.pop("pomegranate")
# Removes and returns the value with key = "pomegranate"
removed_pair = dt.popitem()
# Pops a random item and returns the key-value pair as a tuple.

print(removed_value)
print(removed_pair)
print(dt)

# Output
red
('orange', 'orange')
{"banana": "yellow", "pear": "pink",
"Lemon": "lime"}
```

Miscellaneous:

```
count = len(dt)
print(count)

value = dt['banana']
print(value)

keys = list(dt.keys())
print(keys)

values = list(dt.values())
print(values)

key_exists = 'apple' in dt
print(key_exists)

items = list(dt.items())
print(items)

dt.clear()
print(dt)
```

Output:

```
3
yellow
['banana', 'pear',
'Lemon']
['yellow', 'pink',
'lime']
False
[('banana', 'yellow'),
('pear', 'pink'),
('Lemon', 'lime')]
{}
```

File Handling

Reading from a File

Code

```
with open('filename.txt', 'r') as file:
    print(file.read())  # Outputs the content of the file
```

```
File Handling using Python
Python can open a file in read mode.
        It preserves the special characters like newline, spaces, etc.
```

File Content as it is.

File Methods

Code

```
with open('filename.txt', 'r') as file:
    print(file.readline())  # Outputs the first line of the file
```

```
File Handling using Python
```

First line only.

Writing to a File

Code

```
with open('filename.txt', 'w') as file:
    file.write('Hello, world!')
```

```
Hello, world!
```

Updated file content.

Working with JSON Data

Code

```
import json
data = {'Name': 'Zophie', 'Species': 'cat', 'age': '7'}
json_data = json.dumps(data)  # Converts into JSON string

with open('filename.txt', 'w') as file:
    file.write(json_data)
```

```
{"Name": "Zophie", "Species": "cat", "age": "7"}
```

Updated file content.

Object-Oriented Programming

Classes and Objects

Code

```
class MyClass:
    x = 5

obj = MyClass()
print(obj.x)     # Output: 5
```

Class blueprint

object of MyClass

The object has all properties of the class.

The self Parameter

1. The first argument of any method of a class is always self.
2. The other arguments follow up after the self argument.
3. Using self keyword, you can access the data members and call member functions of the object.
4. self is not a keyword in python.
5. By convention, the first argument is always kept as self.
6. In the init method, self refers to the newly created object and in other class methods it refers to the object whose method was called.
7. self is nothing but a placeholder for the current object.

Code

```
class MyClass:
    base = 2

    def func(self, pw):
        print(self.base ** pw)

obj = MyClass()
obj.func(5)   # Output: 32
```

Using self, we can access the members of the current object.

Constructors: init() function

1. Constructors are special or specific methods used by a class to perform task such as initiating variables, performing startup task and that needs to be done when an instance of a class is generated.
2. When you don't provide any constructor, then automatically a default constructor is created for you.
3. Whenever you create an object of a class, a constructor is called.

Default Constructor

```
def __init__(self):
  pass
```

Code

```
class MyClass:

    def __init__(self, name):
        self.name = name

obj = MyClass("Alice")
print(obj.name)  # Output: Alice
```

A constructor to create an object with name attribute equal to the passed value.

Object Methods

1. Constructors are special or specific methods used by a class to perform task such as initiating variables, performing startup task and that needs to be done when an instance of a class is generated.
2. When you don't provide any constructor, then automatically a default constructor is created for you.
3. Whenever you create an object of a class, a constructor is called.

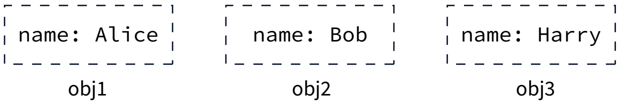
Default Constructor

```
def __init__(self):
  pass
```

Code

```
class MyClass:

    def __init__(self, name):
        self.name = name

obj1 = MyClass("Alice")
obj2 = MyClass("Bob")
obj3 = MyClass("Harry")
```

A constructor to create an object with name attribute equal to the passed value.



name: Alice

obj1

name: Bob

obj2

name: Harry

obj3

Inheritance

1. Inheritance is the process by which an object of one class acquires the properties of another class.
2. Reusable code
3. It resembles real life models.
4. Base class: The class which is inherited is called the base class.
5. Derived class: The class which inherits is called derived class.

Code

```
class Parent:
    def func(self):
        print("This is a function of the parent class.")

class Child(Parent):
    pass

obj = Child()
obj.func()  # Output: This is a function of the parent class.
```

The func() is not defined in the Child class. It is defined in the Parent class.

Child class object has access to all methods and variables of Parent class.

Encapsulation

1. Data and the methods which operate on that data are defined inside a single unit. This concept is called encapsulation.
2. No manipulation or access is allowed directly from outside the capsule or class.

Code

```
class MyClass:
    def __init__(self):
        self.__private_var = "I'm private!"

    def access_private_var(self):
        return self.__private_var

obj = MyClass()
print(obj.access_private_var())  # Output: I'm private!
```

The __private_var cannot be accessed directly from the object

Erroneous Code

```
class MyClass:
    def __init__(self):
        self.__private_var = "I'm private!"

    def access_private_var(self):
        return self.__private_var

obj = MyClass()
print(obj.__private_var)  # Output: I'm private!
```

Throws Error!

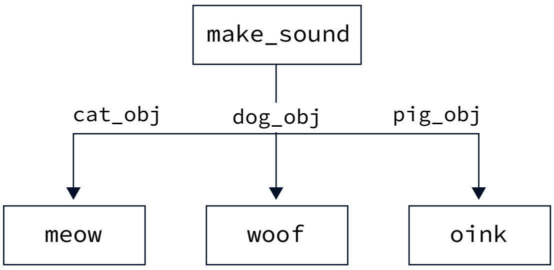
```
Traceback (most recent call last):
  File "/home/captain/python_programs/f1.py", line 9, in <module>
    print(obj.__private_var)  # Output: I'm private!
AttributeError: 'MyClass' object has no attribute '__private_var'
```

Polymorphism

1. Data and the methods which operate on that data are defined inside a single unit. This concept is called encapsulation.
2. No manipulation or access is allowed directly from outside the capsule or class.

Code

```
class Cat:
    def sound(self):
        return "meow"

class Dog:
    def sound(self):
        return "woof"

class Pig:
    def sound(self):
        return "oink"

def make_sound(animal):
    print(animal.sound())

cat_obj = Cat()
dog_obj = Dog()
pig_obj = Pig()

make_sound(cat_obj)  # Output: meow
make_sound(dog_obj)  # Output: woof
make_sound(pig_obj)  # Output: woof
```



Errors and Exception Handling

Syntax Errors

Code

```
while True print('Hello world')
# Syntax error: invalid syntax
```

SyntaxError: invalid syntax

Exceptions

Code

```
print(10 * (1/0))
```

ZeroDivisionError: division by zero

Try...Except

Code

```
try:
    print(10 * (1/0))

except ZeroDivisionError:
    print("Division by zero occurred!")
```

Division by zero occurred!

The Else Clause

When there is no exception, execute the code under else block.

Code

```
try:
    print("Hello")
except:
    print("Something went wrong")
else:
    print("Nothing went wrong")
```

```
Hello
Nothing went wrong
```

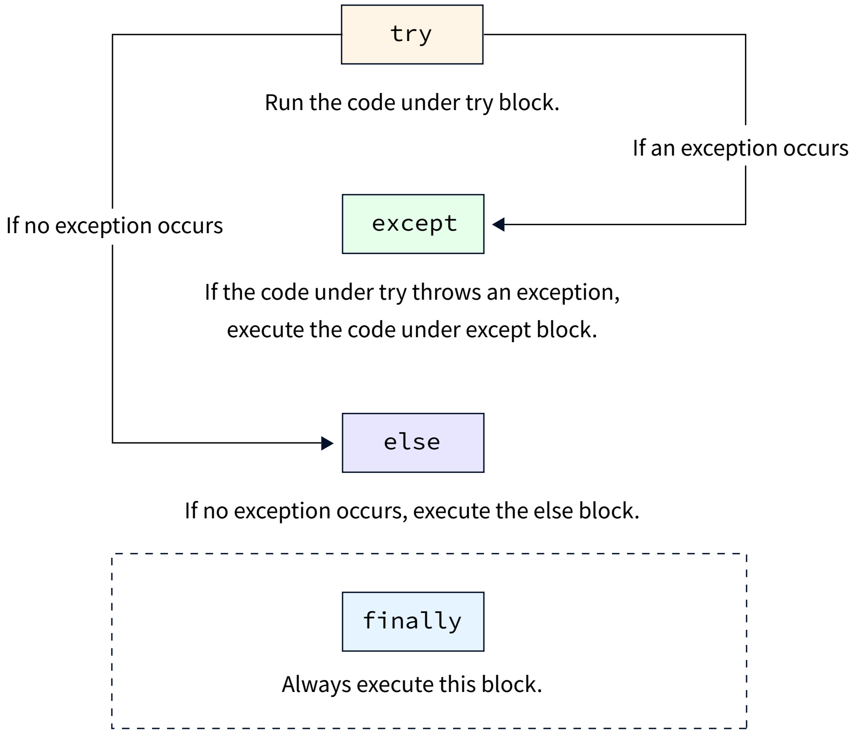
The Finally Clause

The code block under finally is always executed.

Code

```
try:
    print(10 * (1/0))

except ZeroDivisionError:
    print("Division by zero occurred!")

finally:
    print("This line will always be executed")
```

```
Division by zero occurred!
This line will always be executed
```


try
Run the code under try block.
If an exception occurs
If no exception occurs
except
If the code under try throws an exception,
execute the code under except block.
else
If no exception occurs, execute the else block.
finally
Always execute this block.

Python Standard Library

math Module

Code

```python
import math

# Square Root
print("Square root of 16 is:", math.sqrt(16))

# Power
print("2 raised to the power 3 is:", math.pow(2, 3))

# Absolute
print("Absolute value of -10 is:", math.fabs(-10))

# Ceiling
print("Ceiling value of 2.3 is:", math.ceil(2.3))

# Floor
print("Floor value of 2.3 is:", math.floor(2.3))

# PI
print("Value of PI is:", math.pi)

# Euler's number (e)
print("Value of Euler's number is:", math.e)

# Trigonometric functions
print("Cosine of PI is:", math.cos(math.pi))
print("Sine of PI/2 is:", math.sin(math.pi/2))
print("Tangent of 0 is:", math.tan(0))

# Logarithm (base e)
print("Natural logarithm of 1 is:", math.log(1))

# Logarithm (base 10)
print("Common logarithm (base 10) of 100 is:", math.log10(100))
```

Output

```
Square root of 16 is: 4.0
2 raised to the power 3 is: 8.0
Absolute value of -10 is: 10.0
Ceiling value of 2.3 is: 3
Floor value of 2.3 is: 2
Value of PI is: 3.141592653589793
Value of Euler's number is: 2.718281828459045
Cosine of PI is: -1.0
Sine of PI/2 is: 1.0
Tangent of 0 is: 0.0
Natural logarithm of 1 is: 0.0
Common logarithm (base 10) of 100 is: 2.0
```

datetime Module

Code

```
import datetime

# Get the current date and time
now = datetime.datetime.now()
print("Current date and time is:", now)

# Get just the current date
today = datetime.date.today()
print("Current date is:", today)

# Create a specific date
specific_date = datetime.date(2023, 5, 19)  # format is (year, month, day)
print("Specific date is:", specific_date)

# Create a specific time
specific_time = datetime.time(13, 24, 45)  # format is (hour, minute, second)
print("Specific time is:", specific_time)

# Create a specific date and time
specific_datetime = datetime.datetime(2023, 5, 19, 13, 24, 45)  # format is (year, month, day,
hour, minute, second)
print("Specific datetime is:", specific_datetime)

# Get the day of the week (Monday is 0, Sunday is 6)
print("Day of the week:", today.weekday())

# Time delta (difference between two dates or times)
date1 = datetime.date(2023, 5, 19)
date2 = datetime.date(2023, 6, 19)
delta = date2 - date1
print("Days between date1 and date2:", delta.days)

# Adding or subtracting a timedelta to a date or datetime
one_week = datetime.timedelta(weeks=1)
future_date = date1 + one_week
print("One week after date1:", future_date)
```

Output

```
Current date and time is: 2023-05-19 22:26:27.975170
Current date is: 2023-05-19
Specific date is: 2023-05-19
Specific time is: 13:24:45
Specific datetime is: 2023-05-19 13:24:45
Day of the week: 4
Days between date1 and date2: 31
One week after date1: 2023-05-26
```

os Module

Code

```
import os

# Get the current working directory
cwd = os.getcwd()
print("Current working directory is:", cwd)

# Change the current working directory
os.chdir('/path/to/directory')  # replace '/path/to/directory' with an actual directory path
print("Current working directory is:", os.getcwd())

# List files and directories in the current working directory
print("Files and directories in '", cwd, "' :")
print(os.listdir(cwd))

# Create a new directory
os.mkdir('test_dir')  # creates a directory named 'test_dir' in the current working directory
print("Files and directories in '", cwd, "' after creating new directory:")
print(os.listdir(cwd))

# Rename a file or directory
os.rename('test_dir', 'new_dir')  # renames 'test_dir' to 'new_dir'
print("Files and directories in '", cwd, "' after renaming directory:")
print(os.listdir(cwd))

# Remove a directory
os.rmdir('new_dir')  # removes the directory named 'new_dir'
print("Files and directories in '", cwd, "' after removing directory:")
print(os.listdir(cwd))

# Get environment variables
print("Environment variables:")
print(os.environ)

# Get specific environment variable
print("HOME environment variable:")
print(os.environ.get('HOME'))
```

random Module

Code

```
import random

# Generate a random float between 0.0 and 1.0
rand_float = random.random()
print("Random float between 0.0 and 1.0:", rand_float)

# Generate a random integer between a and b (both inclusive)
rand_int = random.randint(1, 10)
print("Random integer between 1 and 10:", rand_int)

# Generate a random float between a and b
rand_uniform = random.uniform(1, 10)
print("Random float between 1 and 10:", rand_uniform)

# Choose a random element from a list
my_list = [1, 2, 3, 4, 5]
rand_choice = random.choice(my_list)
print("Random choice from the list [1, 2, 3, 4, 5]:", rand_choice)

# Shuffle a list
random.shuffle(my_list)
print("List [1, 2, 3, 4, 5] after shuffling:", my_list)

# Generate a random sample from a list
rand_sample = random.sample(my_list, 3)
print("Random sample of 3 elements from the list:", rand_sample)

# Generate a random float with a normal distribution
rand_normal = random.gauss(mu=0, sigma=1)
print("Random float with a normal distribution (mu=0, sigma=1):", rand_normal)
```

Output

```
Random float between 0.0 and 1.0: 0.8061507911052314
Random integer between 1 and 10: 9
Random float between 1 and 10: 7.189451780281582
Random choice from the list [1, 2, 3, 4, 5]: 5
List [1, 2, 3, 4, 5] after shuffling: [2, 1, 3, 4, 5]
Random sample of 3 elements from the list: [1, 3, 2]
Random float with a normal distribution (mu=0, sigma=1): -0.3137487021982147
```

Advanced Topics

Generators

1. Python's yield keeps local variables intact upon function return.
2. Yield halts function execution, resumes from last yield on re-invocation.
3. Yielding a value transforms a function into a generator, returning a generator object.

Function -> Generator

```
def square(numbers):
    sqs = []
    for n in numbers:
        sqs.append(n ** 2)

    return sqs

numbers = [1, 2, 3, 4, 5]
sq_nums = square(numbers)

print(sq_nums)
```

Replace return with yield

```
def square(numbers):
    for n in numbers:
        yield n ** 2

numbers = [1, 2, 3, 4, 5]
sq_nums = square(numbers)

print(sq_nums)
```

Extract values from generator object

Method 1: next() function

```
print(next(sq_nums))
print(next(sq_nums))
print(next(sq_nums))
print(next(sq_nums))
print(next(sq_nums))
```

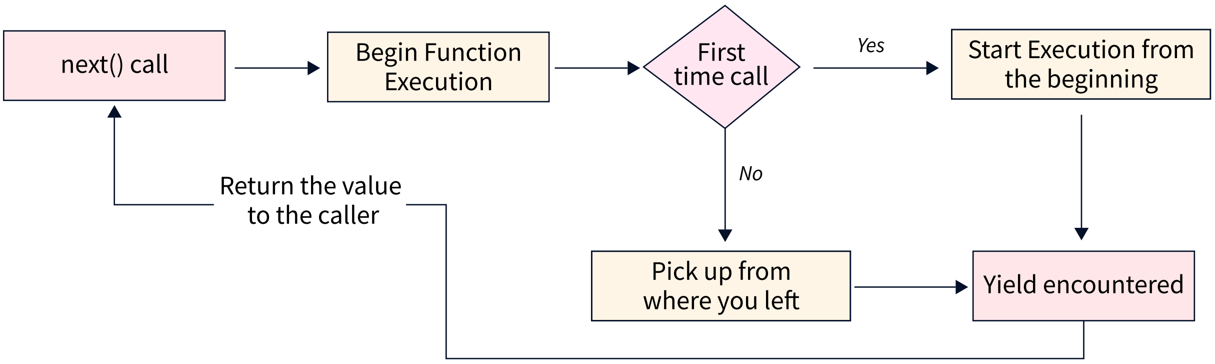
Method 2: next() function

```
for n in sq_nums:
    print(n)
```

Output

```
1
4
9
16
25
```

How it works!


next() call
Begin Function Execution
First time call
Yes
Start Execution from the beginning
No
Pick up from where you left
Yield encountered
Return the value to the caller

Decorators

1. A decorator allows to modify the functionality of a function by wrapping in another function.
2. The outer function is called the decorator and the inner one is modified if required and returned by the decorator.

Example

Consider a function which divides two numbers.

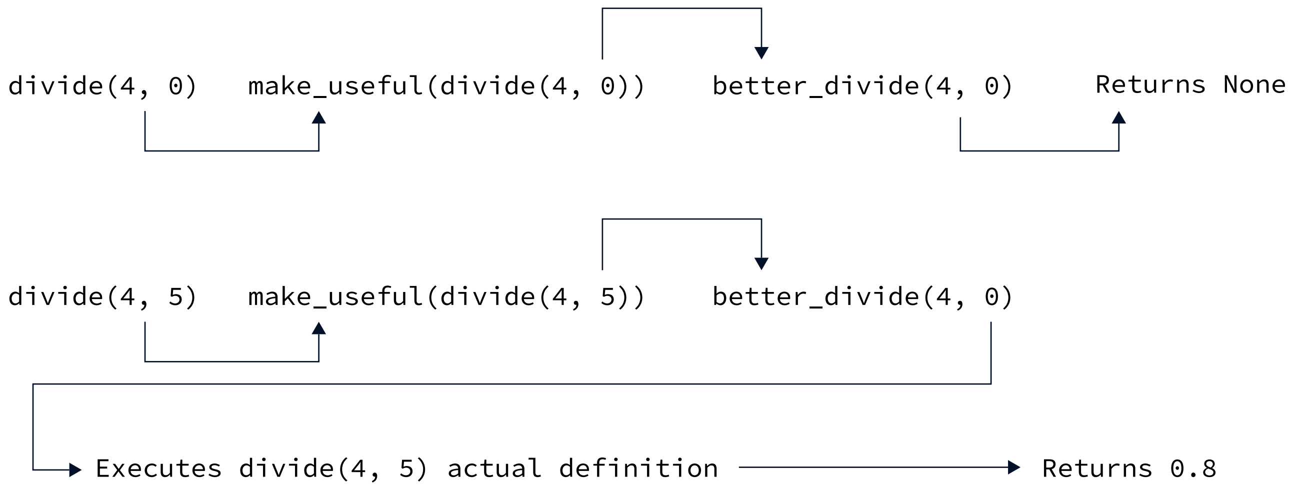
```
def divide(a, b):
    return a / b
```

Now, you want to add one functionality which checks if b is non-zero without changing divide function.

Here, decorators are of use.

```
def make_useful(divide_func):

    def better_divide(a, b):

        if b == 0:
            print("Denominator must be non-zero")
            return None

        return divide_func(a, b)

    return better_divide

@make_useful
def divide(a, b):
    return a / b
```

Flow of a divide call



divide(4, 0)

make_useful(divide(4, 0))

better_divide(4, 0)

Returns None

divide(4, 5)

make_useful(divide(4, 5))

better_divide(4, 0)

Executes divide(4, 5) actual definition

Returns 0.8

Context Managers

1. Context Managers are used to manage resources efficiently.
2. It uses the with statement to define the scope of the resource.
3. It ensures proper resource handling and exception handling by invoking the __enter__() and __exit__() methods.

Example

```
with open("filename.txt") as f:
    data = f.read()
```

Creates a file descriptor f used to access a file resource.

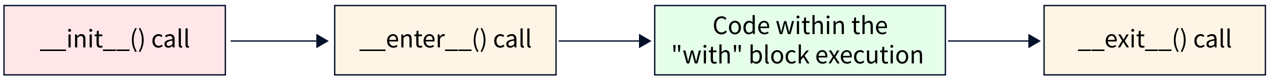
```
file_descriptors = []
for fl in range(10000):
    file_descriptors.append(open('filename.txt', 'w'))
```

```
Traceback (most recent call last):
  File "contextManager.py", line 3, in
OSError: [Errno 24] Too many open files: 'filename.txt'
```

With Context Managers, a resource is handled properly by calling the __enter__() and __exit__() methods by default.

```
class FileManager():
    def __init__(self, filename, mode):
        self.file = None
        self.filename = filename
        self.mode = mode

    def __enter__(self):
        self.file = open(self.filename, self.mode) # Open the file
        return self.file # return the file descriptor

    def __exit__(self, exc_type, exc_value, exc_traceback):
        self.file.close() # Close the file while exiting

# loading a file
with FileManager('test.txt', 'w') as f:
    f.write('Test') # Execute this code after __enter__() method finishes.
```

Sequence of Function calls



SCALER TOPICS

Unlock your potential in software development with **FREE COURSES** from **SCALER TOPICS!**

Register now and take the first step towards your future Success!

C++ for Beginners

5.9k enrolled ₹ Free

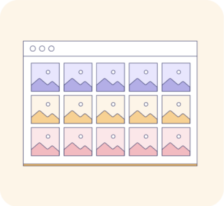
Java for Beginners

6.8k enrolled ₹ Free

That’s not it. Explore 20+ Courses by clicking below

Explore Other Courses

Practice **CHALLENGES**
and become 1% better everyday

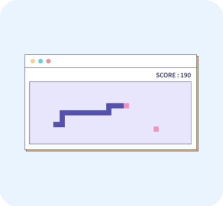


CIFAR-10 Image Classification Using PyTorch

Article

No. Of Questions : 3

Go to Challenge >



How to Build a Snake Game in JavaScript?

Article

No. Of Questions : 3

Go to Challenge >

Explore Other Challenges