Practice1

August 26, 2025

[ ]:
```
# Everthing in Python is an object
import math # math is a module and also an object.
# When we import a math module then it creates a module object named math
type(math)
```

[ ]:
```
module
```

[9]:
```
math?
```

```
Type:        module
String form: <module 'math' from
'/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/lib-dynload/math.cpython-310-x86_64-linux-
gnu.so'>
File:        ~/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/lib-dynload/math.cpython-310-x86_64-linux-gnu.so
Docstring:
This module provides access to the mathematical functions
defined by the C standard.
```

[27]:
```
math.sqrt?
```

```
Signature:
math.sqrt(x,
/)
Docstring: Return the square root of x.
Type:      builtin_function_or_method
```

[28]:
```
math.pi?
```

```
Type:        float
String form: 3.141592653589793
Docstring:   Convert a string or number to a floating point number,
if possible.
```

[29]:
```
math.factorial?
```

```
Signature:
```

```
math.factorial(x,
/)
Docstring:
Find x!.

Raise a ValueError if x is negative or non-integral.
Type:      builtin_function_or_method
```

[30]:
```
math.comb?
```

```
Signature:
math.comb(n,
k, /)
Docstring:
Number of ways to choose k items from n items without repetition and without
order.

Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
to zero when k > n.

Also called the binomial coefficient because it is equivalent
to the coefficient of k-th term in polynomial expansion of the
expression (1 + x)**n.

Raises TypeError if either of the arguments are not integers.
Raises ValueError if either of the arguments are negative.
Type:      builtin_function_or_method
```

[31]:
```
math.gamma?
```

```
Signature:
math.gamma(x,
/)
Docstring: Gamma function at x.
Type:      builtin_function_or_method
```

[32]:
```
math.prod?
```

```
Signature:
math.prod(iterable,
/, *,
start=1)
Docstring:
Calculate the product of all the elements in the input iterable.

The default start value for the product is 1.

When the iterable is empty, return the start value.  This function is
```

```
intended specifically for use with numeric values and may reject
non-numeric types.
Type:      builtin_function_or_method
```

[ ]:
```
math.tau?
#A value of 6.28 represents tau (), a mathematical constant equal to 2 (two␣
 ↪times pi).
# Tau is defined as the ratio of a circle's circumference to its radius,
# where one full circle corresponds to an angle of  radians.
# Using  simplifies many formulas by eliminating the factor of two,
# making concepts like a full turn around a circle equal to 1 .
```

```
Type:        float
String form: 6.283185307179586
Docstring:   Convert a string or number to a floating point number,
if possible.
```

[10]:
```
int?
```

```
Init signature: int(self,
/, *args,
**kwargs)
Docstring:
int([x]) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments
are given.  If x is a number, return x.__int__().  For floating point
numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string,
bytes, or bytearray instance representing an integer literal in the
given base.  The literal can be preceded by '+' or '-' and be surrounded
by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
Base 0 means to interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4
Type:           type
Subclasses:     bool, IntEnum, IntFlag, _NamedIntConstant
```

[14]:
```
int('101', base=2) # base is also a parameter of int class and it is also an␣
 ↪object
```

[14]:
```
5
```

[15]:
```
int('50', base=8) # base is also a parameter of int class and it is also an␣
 ↪object
```

[15]: 40

[19]:
```
float(10) # float() does not take a 'base' parameter
```

[19]: 10.0

[20]:
```
float?
```

```
Init signature:
float(x=0,
/)
Docstring:      Convert a string or number to a floating point
number, if possible.
Type:           type
Subclasses:
```

[21]:
```
str?
```

```
Init signature: str(self,
/, *args,
**kwargs)
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
Type:           type
Subclasses:     DeferredConfigString, FoldedCase, _rstr, LSString,
include, Keys, InputMode, ColorDepth, CompleteStyle, SortKey, …
```

[25]:
```
str(b'ankit', encoding='utf-8', errors='strict') # encoding and errors are also␣
↪parameters of str class and they are also objects
```

[25]: 'ankit'

[8]:
```
x=1.5 # x is a variable of type float but this variable is also an object
type(x)
```

[8]: float

[1]:
```
def type(n): #using keyword "type" as a function name but it is not a good␣
 ↪practice
    return n*n*n
```

[40]:
```
type(5)
```

[40]:
```
125
```

[ ]:
```
type=5 # variable and funcxtion name should not be same as a keyword or␣
 ↪built-in function name
# but it is not a good practice
```

[3]:
```
def 123name(): # function name should not start with a number
    return "ankit"
```

```
  Cell In[3], line 1
    def 123name(): # function name should not start with a number
        ^
SyntaxError: invalid decimal literal
```

[4]:
```
def _name(): # function name can start with an underscore
    return "ankit"
```

[5]:
```
_name()
```

[5]:
```
'ankit'
```

[9]:
```
def __myname(): # function name can start with double underscore
        return "ankit gupta"
```

[10]:
```
__myname()
```

[10]:
```
'ankit gupta'
```

[24]:
```
def name(): # function name should not start with a number
    return "ankit"
print(__builtins__.type(name()))
print(__builtins__.type(name))
```

```
<class 'str'>
<class 'function'>
```

[20]:
```
__builtins__.dir() # it will give all the built-in functions and variables in␣
 ↪python
```

[20]:
```
['In',
 'Out',
 '_',
 '_10',
 '_17',
 '_18',
 '_19',
 '_5',
 '_7',
 '_8',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
 '__loader__',
 '__myname',
 '__name__',
 '__package__',
 '__spec__',
 '__vsc_ipynb_file__',
 '_dh',
 '_i',
 '_i1',
 '_i10',
 '_i11',
 '_i12',
 '_i13',
 '_i14',
 '_i15',
 '_i16',
 '_i17',
 '_i18',
 '_i19',
 '_i2',
 '_i20',
 '_i3',
 '_i4',
 '_i5',
 '_i6',
 '_i7',
 '_i8',
 '_i9',
 '_ih',
 '_ii',
 '_iii',
 '_name',
```

```
 '_oh',
 'exit',
 'get_ipython',
 'name',
 'name_123',
 'open',
 'quit',
 'type']
```

[1]:
```
def my_function():
    pass

print(type(my_function))
```

```
<class 'function'>
```

[2]:
```
import types

def another_function():
    pass

if isinstance(another_function, types.FunctionType):
    print("another_function is a function.")
else:
    print("another_function is not a function.")
```

```
another_function is a function.
```

[15]:
```
def my_name():
    print("My name is Ankit Gupta")
    return "ankit"
```

[16]:
```
def yet_another_function():
    print("Hello, World!")
    my_name()
    x= my_name()
    print(x)
    return "Function executed"
```

[17]:
```
yet_another_function()
```

```
Hello, World!
My name is Ankit Gupta
My name is Ankit Gupta
ankit
```

[17]:
```
'Function executed'
```

[14]:
```
my_name()
```

```
My name is Ankit Gupta
```

[14]:
```
'ankit'
```

[18]:
```
def my_name():
    print("Ankit Gupta")
    x= "My name is Ankit Gupta"
    return x
```

[20]:
```
my_name()
```

```
Ankit Gupta
```

[20]:
```
'My name is Ankit Gupta'
```

[21]:
```
def my_function():
    """This is a docstring for my_function."""
    x= 10
    return x
x=20
if __name__ == "__main__":
    print("This script is being run directly.")
    print("local x= ",my_function())
    print("global x= ",x)
```

```
This script is being run directly.
local x=  10
global x=  20
```

[22]:
```
import inspect

def func_c(arg_c):
    local_c = arg_c * 2
    print(f"Inside func_c: local_c = {local_c}")
    for frame_info in inspect.stack():
        print(f"  Frame: {frame_info.function}, Line: {frame_info.lineno}")
        if frame_info.function != '<module>':  # Exclude the global scope
            print(f"    Locals: {frame_info.frame.f_locals}")

def func_b(arg_b):
    local_b = arg_b + 5
    print(f"Inside func_b: local_b = {local_b}")
    func_c(local_b)

def func_a(arg_a):
    local_a = arg_a - 1
```

```
    print(f"Inside func_a: local_a = {local_a}")
    func_b(local_a)

func_a(10)
```

```
Inside func_a: local_a = 9
Inside func_b: local_b = 14
Inside func_c: local_c = 28
  Frame: func_c, Line: 6
    Locals: {'arg_c': 14, 'local_c': 28, 'frame_info': FrameInfo(frame=<frame at
0x70f798d5dfc0, file '/tmp/ipykernel_5343/2984794816.py', line 9, code func_c>,
filename='/tmp/ipykernel_5343/2984794816.py', lineno=6, function='func_c',
code_context=['    for frame_info in inspect.stack():\n'], index=0)}
  Frame: func_b, Line: 14
    Locals: {'arg_b': 9, 'local_b': 14}
  Frame: func_a, Line: 19
    Locals: {'arg_a': 10, 'local_a': 9}
  Frame: <module>, Line: 21
  Frame: run_code, Line: 3579
    Locals: {'self': <ipykernel.zmqshell.ZMQInteractiveShell object at
0x70f79a18c970>, 'code_obj': <code object <module> at 0x70f798d4a290, file
"/tmp/ipykernel_5343/2984794816.py", line 1>, 'result': <ExecutionResult object
at 70f798d34760, execution_count=22 error_before_exec=None error_in_exec=None
info=<ExecutionInfo object at 70f798d34fd0, raw_cell="import inspect

def func_c(arg_c):
    local_c = a.." store_history=True silent=False shell_futures=True
cell_id=vscode-notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D> result=None>, 'async_': False,
'__tracebackhide__': '__ipython_bottom__', 'old_excepthook': <bound method
IPKernelApp.excepthook of <ipykernel.kernelapp.IPKernelApp object at
0x70f79e619c00>>, 'outflag': True}
  Frame: run_ast_nodes, Line: 3519
    Locals: {'self': <ipykernel.zmqshell.ZMQInteractiveShell object at
0x70f79a18c970>, 'nodelist': [<ast.Import object at 0x70f798d368c0>,
<ast.FunctionDef object at 0x70f798d34d90>, <ast.FunctionDef object at
0x70f798d35840>, <ast.FunctionDef object at 0x70f798d36620>, <ast.Expr object at
0x70f798d367d0>], 'cell_name': '/tmp/ipykernel_5343/2984794816.py',
'interactivity': 'last', 'compiler': <ipykernel.compiler.XCachingCompiler object
at 0x70f79a18cac0>, 'result': <ExecutionResult object at 70f798d34760,
execution_count=22 error_before_exec=None error_in_exec=None info=<ExecutionInfo
object at 70f798d34fd0, raw_cell="import inspect

def func_c(arg_c):
    local_c = a.." store_history=True silent=False shell_futures=True
cell_id=vscode-notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D> result=None>, '_async': False,
```

```
'to_run_exec': [<ast.Import object at 0x70f798d368c0>, <ast.FunctionDef object
at 0x70f798d34d90>, <ast.FunctionDef object at 0x70f798d35840>, <ast.FunctionDef
object at 0x70f798d36620>], 'to_run_interactive': [<ast.Expr object at
0x70f798d367d0>], 'compare': <function
InteractiveShell.run_ast_nodes.<locals>.compare at 0x70f798d163b0>, 'to_run':
[(<ast.Import object at 0x70f798d368c0>, 'exec'), (<ast.FunctionDef object at
0x70f798d34d90>, 'exec'), (<ast.FunctionDef object at 0x70f798d35840>, 'exec'),
(<ast.FunctionDef object at 0x70f798d36620>, 'exec'), (<ast.Expr object at
0x70f798d367d0>, 'single')], 'node': <ast.Expr object at 0x70f798d367d0>,
'mode': 'single', 'mod': <ast.Interactive object at 0x70f798d34430>, 'code':
<code object <module> at 0x70f798d4a290, file
"/tmp/ipykernel_5343/2984794816.py", line 1>, 'asy': False}
  Frame: run_cell_async, Line: 3336
    Locals: {'raw_cell': 'import inspect\n\ndef func_c(arg_c):\n    local_c =
arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for frame_info
in inspect.stack():\n        print(f"  Frame: {frame_info.function}, Line:
{frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)', 'silent': False,
'shell_futures': True, 'transformed_cell': 'import inspect\n\ndef
func_c(arg_c):\n    local_c = arg_c * 2\n    print(f"Inside func_c: local_c =
{local_c}")\n    for frame_info in inspect.stack():\n        print(f"  Frame:
{frame_info.function}, Line: {frame_info.lineno}")\n        if
frame_info.function != \'<module>\':  # Exclude the global scope\n
print(f"    Locals: {frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n
local_b = arg_b + 5\n    print(f"Inside func_b: local_b = {local_b}")\n
func_c(local_b)\n\ndef func_a(arg_a):\n    local_a = arg_a - 1\n
print(f"Inside func_a: local_a = {local_a}")\n
func_b(local_a)\n\nfunc_a(10)\n', 'preprocessing_exc_tuple': None, 'cell_id':
'vscode-notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D', 'info': <ExecutionInfo object at
70f798d34fd0, raw_cell="import inspect

def func_c(arg_c):
    local_c = a.." store_history=True silent=False shell_futures=True
cell_id=vscode-notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D>, 'error_before_exec': <function
InteractiveShell.run_cell_async.<locals>.error_before_exec at 0x70f798d15b40>,
'cell': 'import inspect\n\ndef func_c(arg_c):\n    local_c = arg_c * 2\n
print(f"Inside func_c: local_c = {local_c}")\n    for frame_info in
inspect.stack():\n        print(f"  Frame: {frame_info.function}, Line:
{frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
```

```
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)\n', 'compiler':
<ipykernel.compiler.XCachingCompiler object at 0x70f79a18cac0>, 'cell_name':
'/tmp/ipykernel_5343/2984794816.py', 'code_ast': <ast.Module object at
0x70f798d35570>, 'interactivity': 'last_expr', 'result': <ExecutionResult object
at 70f798d34760, execution_count=22 error_before_exec=None error_in_exec=None
info=<ExecutionInfo object at 70f798d34fd0, raw_cell="import inspect

def func_c(arg_c):
    local_c = a.." store_history=True silent=False shell_futures=True
cell_id=vscode-notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D> result=None>, 'self':
<ipykernel.zmqshell.ZMQInteractiveShell object at 0x70f79a18c970>,
'store_history': True}
  Frame: _pseudo_sync_runner, Line: 128
    Locals: {'coro': <coroutine object InteractiveShell.run_cell_async at
0x70f798db1e00>}
  Frame: _run_cell, Line: 3132
    Locals: {'self': <ipykernel.zmqshell.ZMQInteractiveShell object at
0x70f79a18c970>, 'raw_cell': 'import inspect\n\ndef func_c(arg_c):\n    local_c
= arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for
frame_info in inspect.stack():\n        print(f"  Frame: {frame_info.function},
Line: {frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)', 'store_history': True,
'silent': False, 'shell_futures': True, 'cell_id': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D', 'preprocessing_exc_tuple': None,
'transformed_cell': 'import inspect\n\ndef func_c(arg_c):\n    local_c = arg_c *
2\n    print(f"Inside func_c: local_c = {local_c}")\n    for frame_info in
inspect.stack():\n        print(f"  Frame: {frame_info.function}, Line:
{frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)\n', 'coro': <coroutine object
InteractiveShell.run_cell_async at 0x70f798db1e00>, 'runner': <function
_pseudo_sync_runner at 0x70f79cc65cf0>}
  Frame: run_cell, Line: 3077
    Locals: {'self': <ipykernel.zmqshell.ZMQInteractiveShell object at
0x70f79a18c970>, 'raw_cell': 'import inspect\n\ndef func_c(arg_c):\n    local_c
= arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for
frame_info in inspect.stack():\n        print(f"  Frame: {frame_info.function},
Line: {frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
```

```
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)', 'store_history': True,
'silent': False, 'shell_futures': True, 'cell_id': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D', 'result': None}
  Frame: run_cell, Line: 577
    Locals: {'self': <ipykernel.zmqshell.ZMQInteractiveShell object at
0x70f79a18c970>, 'args': ('import inspect\n\ndef func_c(arg_c):\n    local_c =
arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for frame_info
in inspect.stack():\n        print(f"  Frame: {frame_info.function}, Line:
{frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)',), 'kwargs': {'store_history':
True, 'silent': False, 'cell_id': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, '__class__': <class
'ipykernel.zmqshell.ZMQInteractiveShell'>}
  Frame: do_execute, Line: 455
    Locals: {'self': <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>, 'code': 'import inspect\n\ndef func_c(arg_c):\n    local_c =
arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for frame_info
in inspect.stack():\n        print(f"  Frame: {frame_info.function}, Line:
{frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)', 'silent': False,
'store_history': True, 'user_expressions': {}, 'allow_stdin': True, 'cell_meta':
{'cellId': 'vscode-notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-
ML-DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'cell_id': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D', 'reply_content': {}, 'run_cell':
<bound method InteractiveShell.run_cell_async of
<ipykernel.zmqshell.ZMQInteractiveShell object at 0x70f79a18c970>>,
'should_run_async': <bound method InteractiveShell.should_run_async of
<ipykernel.zmqshell.ZMQInteractiveShell object at 0x70f79a18c970>>,
'accepts_params': {'cell_id': True}, 'preprocessing_exc_tuple': None,
'transformed_cell': 'import inspect\n\ndef func_c(arg_c):\n    local_c = arg_c *
2\n    print(f"Inside func_c: local_c = {local_c}")\n    for frame_info in
inspect.stack():\n        print(f"  Frame: {frame_info.function}, Line:
{frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
```

```
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)\n', 'shell':
<ipykernel.zmqshell.ZMQInteractiveShell object at 0x70f79a18c970>}
  Frame: execute_request, Line: 767
    Locals: {'self': <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>, 'stream': <zmq.eventloop.zmqstream.ZMQStream object at
0x70f79bfd6e90>, 'ident': [b'3b9d94f1-e1da-45bb-ad41-fea0f24e6e77'], 'parent':
{'header': {'date': datetime.datetime(2025, 8, 25, 12, 11, 35, 829000,
tzinfo=tzutc()), 'msg_id': 'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type':
'execute_request', 'session': 'fab9517e-1ebd-4779-8b74-980244f24816',
'username': 'fd9c9126-3e96-452b-a5f7-4cf8de179441', 'version': '5.2'}, 'msg_id':
'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type': 'execute_request',
'parent_header': {}, 'metadata': {'cellId': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'content': {'silent': False,
'store_history': True, 'user_expressions': {}, 'allow_stdin': True,
'stop_on_error': False, 'code': 'import inspect\n\ndef func_c(arg_c):\n
local_c = arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for
frame_info in inspect.stack():\n        print(f"  Frame: {frame_info.function},
Line: {frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)'}, 'buffers': []}, 'content':
{'silent': False, 'store_history': True, 'user_expressions': {}, 'allow_stdin':
True, 'stop_on_error': False, 'code': 'import inspect\n\ndef func_c(arg_c):\n
local_c = arg_c * 2\n    print(f"Inside func_c: local_c = {local_c}")\n    for
frame_info in inspect.stack():\n        print(f"  Frame: {frame_info.function},
Line: {frame_info.lineno}")\n        if frame_info.function != \'<module>\':  #
Exclude the global scope\n            print(f"    Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n    local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n    func_c(local_b)\n\ndef
func_a(arg_a):\n    local_a = arg_a - 1\n    print(f"Inside func_a: local_a =
{local_a}")\n    func_b(local_a)\n\nfunc_a(10)'}, 'code': 'import inspect\n\ndef
func_c(arg_c):\n    local_c = arg_c * 2\n    print(f"Inside func_c: local_c =
{local_c}")\n    for frame_info in inspect.stack():\n        print(f"  Frame:
{frame_info.function}, Line: {frame_info.lineno}")\n        if
frame_info.function != \'<module>\':  # Exclude the global scope\n
print(f"    Locals: {frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n
local_b = arg_b + 5\n    print(f"Inside func_b: local_b = {local_b}")\n
func_c(local_b)\n\ndef func_a(arg_a):\n    local_a = arg_a - 1\n
print(f"Inside func_a: local_a = {local_a}")\n
func_b(local_a)\n\nfunc_a(10)', 'silent': False, 'store_history': True,
'user_expressions': {}, 'allow_stdin': True, 'cell_meta': {'cellId': 'vscode-
```

notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'cell_id': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D', 'stop_on_error': False,
'metadata': {'started': datetime.datetime(2025, 8, 25, 12, 11, 35, 832314,
tzinfo=datetime.timezone.utc), 'dependencies_met': True, 'engine':
'6b4b3ccb-8bad-4aca-8a9b-a2404dd62be1'}, 'do_execute_args': {'code': 'import
inspect\n\ndef func_c(arg_c):\n   local_c = arg_c * 2\n   print(f"Inside
func_c: local_c = {local_c}")\n   for frame_info in inspect.stack():\n
print(f" Frame: {frame_info.function}, Line: {frame_info.lineno}")\n   if
frame_info.function != \'<module>\': # Exclude the global scope\n
print(f"   Locals: {frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n
local_b = arg_b + 5\n   print(f"Inside func_b: local_b = {local_b}")\n
func_c(local_b)\n\ndef func_a(arg_a):\n   local_a = arg_a - 1\n
print(f"Inside func_a: local_a = {local_a}")\n
func_b(local_a)\n\nfunc_a(10)', 'silent': False, 'store_history': True,
'user_expressions': {}, 'allow_stdin': True, 'cell_meta': {'cellId': 'vscode-
notebook-cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'cell_id': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'reply_content': <coroutine
object IPythonKernel.do_execute at 0x70f798db25e0>}
  Frame: execute_request, Line: 368
    Locals: {'self': <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>, 'stream': <zmq.eventloop.zmqstream.ZMQStream object at
0x70f79bfd6e90>, 'ident': [b'3b9d94f1-e1da-45bb-ad41-fea0f24e6e77'], 'parent':
{'header': {'date': datetime.datetime(2025, 8, 25, 12, 11, 35, 829000,
tzinfo=tzutc()), 'msg_id': 'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type':
'execute_request', 'session': 'fab9517e-1ebd-4779-8b74-980244f24816',
'username': 'fd9c9126-3e96-452b-a5f7-4cf8de179441', 'version': '5.2'}, 'msg_id':
'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type': 'execute_request',
'parent_header': {}, 'metadata': {'cellId': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'content': {'silent': False,
'store_history': True, 'user_expressions': {}, 'allow_stdin': True,
'stop_on_error': False, 'code': 'import inspect\n\ndef func_c(arg_c):\n
local_c = arg_c * 2\n   print(f"Inside func_c: local_c = {local_c}")\n   for
frame_info in inspect.stack():\n   print(f" Frame: {frame_info.function},
Line: {frame_info.lineno}")\n   if frame_info.function != \'<module>\': #
Exclude the global scope\n   print(f"   Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n   local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n   func_c(local_b)\n\ndef
func_a(arg_a):\n   local_a = arg_a - 1\n   print(f"Inside func_a: local_a =
{local_a}")\n   func_b(local_a)\n\nfunc_a(10)'}, 'buffers': []},
'parent_header': {'date': datetime.datetime(2025, 8, 25, 12, 11, 35, 829000,
tzinfo=tzutc()), 'msg_id': 'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type':
'execute_request', 'session': 'fab9517e-1ebd-4779-8b74-980244f24816',
'username': 'fd9c9126-3e96-452b-a5f7-4cf8de179441', 'version': '5.2'},

'__class__': <class 'ipykernel.ipkernel.IPythonKernel'>}
  Frame: dispatch_shell, Line: 400
    Locals: {'self': <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>, 'msg': {'header': {'date': datetime.datetime(2025, 8, 25, 12,
11, 35, 829000, tzinfo=tzutc()), 'msg_id':
'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type': 'execute_request',
'session': 'fab9517e-1ebd-4779-8b74-980244f24816', 'username':
'fd9c9126-3e96-452b-a5f7-4cf8de179441', 'version': '5.2'}, 'msg_id':
'd365bd21-682b-4441-a888-5d9a67dad924', 'msg_type': 'execute_request',
'parent_header': {}, 'metadata': {'cellId': 'vscode-notebook-
cell:/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/1.Python/Practice1.ipynb#X51sZmlsZQ%3D%3D'}, 'content': {'silent': False,
'store_history': True, 'user_expressions': {}, 'allow_stdin': True,
'stop_on_error': False, 'code': 'import inspect\n\ndef func_c(arg_c):\n
local_c = arg_c * 2\n   print(f"Inside func_c: local_c = {local_c}")\n   for
frame_info in inspect.stack():\n   print(f" Frame: {frame_info.function},
Line: {frame_info.lineno}")\n   if frame_info.function != \'<module>\': #
Exclude the global scope\n   print(f"   Locals:
{frame_info.frame.f_locals}")\n\ndef func_b(arg_b):\n   local_b = arg_b + 5\n
print(f"Inside func_b: local_b = {local_b}")\n   func_c(local_b)\n\ndef
func_a(arg_a):\n   local_a = arg_a - 1\n   print(f"Inside func_a: local_a =
{local_a}")\n   func_b(local_a)\n\nfunc_a(10)'}, 'buffers': []}, 'idents':
[b'3b9d94f1-e1da-45bb-ad41-fea0f24e6e77'], 'msg_type': 'execute_request',
'handler': <bound method IPythonKernel.execute_request of
<ipykernel.ipkernel.IPythonKernel object at 0x70f79bfd7730>>, 'result':
<coroutine object IPythonKernel.execute_request at 0x70f798db2340>}
  Frame: process_one, Line: 508
    Locals: {'self': <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>, 'wait': True, 't': 75, 'dispatch': <bound method
Kernel.dispatch_shell of <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>>, 'args': ([<zmq.Frame(b'3b9d94f1-e1d'…36B)>,
<zmq.Frame(b'<IDS|MSG>')>, <zmq.Frame(b'b0c7adfcdaab'…64B)>,
<zmq.Frame(b'{"date":"202'…227B)>, <zmq.Frame(b'{}')>,
<zmq.Frame(b'{"cellId":"v'…128B)>, <zmq.Frame(b'{"silent":fa'…736B)>],)}
  Frame: dispatch_queue, Line: 519
    Locals: {'self': <ipykernel.ipkernel.IPythonKernel object at
0x70f79bfd7730>}
  Frame: _run, Line: 80
    Locals: {'self': <Handle Task.task_wakeup(<Future finis…8B)>, …],))>)>}
  Frame: _run_once, Line: 1909
    Locals: {'self': <_UnixSelectorEventLoop running=True closed=False
debug=False>, 'sched_count': 0, 'handle': <Handle Task.task_wakeup(<Future
finis…8B)>, …],))>)>, 'timeout': 0, 'event_list': None, 'end_time':
1162.126594676, 'ntodo': 2, 'i': 0}
  Frame: run_forever, Line: 603
    Locals: {'self': <_UnixSelectorEventLoop running=True closed=False
debug=False>, 'old_agen_hooks': asyncgen_hooks(firstiter=None, finalizer=None)}
  Frame: start, Line: 211

```
    Locals: {'self': <tornado.platform.asyncio.AsyncIOMainLoop object at
0x70f79bfd7160>}
  Frame: start, Line: 739
    Locals: {'self': <ipykernel.kernelapp.IPKernelApp object at 0x70f79e619c00>}
  Frame: launch_instance, Line: 1075
    Locals: {'cls': <class 'ipykernel.kernelapp.IPKernelApp'>, 'argv': None,
'kwargs': {}, 'app': <ipykernel.kernelapp.IPKernelApp object at 0x70f79e619c00>}
  Frame: <module>, Line: 18
  Frame: _run_code, Line: 86
    Locals: {'code': <code object <module> at 0x70f79e67b890, file
"/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py", line 1>,
'run_globals': {'__name__': '__main__', '__doc__': 'Entry point for launching an
IPython kernel.\n\nThis is separate from the ipykernel package so we can avoid
doing imports until\nafter removing the cwd from sys.path.\n', '__package__':
'', '__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x70f79e657d60>, '__spec__': ModuleSpec(name='ipykernel_launcher',
loader=<_frozen_importlib_external.SourceFileLoader object at 0x70f79e657d60>,
origin='/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py'),
'__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>,
'__file__': '/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py',
'__cached__': '/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-
packages/__pycache__/ipykernel_launcher.cpython-310.pyc', 'sys': <module 'sys'
(built-in)>, 'Path': <class 'pathlib.Path'>, 'app': <module
'ipykernel.kernelapp' from '/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel/kernelapp.py'>},
'init_globals': None, 'mod_name': '__main__', 'mod_spec':
ModuleSpec(name='ipykernel_launcher',
loader=<_frozen_importlib_external.SourceFileLoader object at 0x70f79e657d60>,
origin='/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py'),
'pkg_name': '', 'script_name': None, 'loader':
<_frozen_importlib_external.SourceFileLoader object at 0x70f79e657d60>, 'fname':
'/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py', 'cached':
'/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-
packages/__pycache__/ipykernel_launcher.cpython-310.pyc'}
  Frame: _run_module_as_main, Line: 196
    Locals: {'mod_name': 'ipykernel_launcher', 'alter_argv': True, 'mod_spec':
ModuleSpec(name='ipykernel_launcher',
loader=<_frozen_importlib_external.SourceFileLoader object at 0x70f79e657d60>,
origin='/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py'), 'code':
<code object <module> at 0x70f79e67b890, file
```

```
"/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py", line 1>,
'main_globals': {'__name__': '__main__', '__doc__': 'Entry point for launching
an IPython kernel.\n\nThis is separate from the ipykernel package so we can
avoid doing imports until\nafter removing the cwd from sys.path.\n',
'__package__': '', '__loader__': <_frozen_importlib_external.SourceFileLoader
object at 0x70f79e657d60>, '__spec__': ModuleSpec(name='ipykernel_launcher',
loader=<_frozen_importlib_external.SourceFileLoader object at 0x70f79e657d60>,
origin='/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py'),
'__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>,
'__file__': '/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel_launcher.py',
'__cached__': '/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-
packages/__pycache__/ipykernel_launcher.cpython-310.pyc', 'sys': <module 'sys'
(built-in)>, 'Path': <class 'pathlib.Path'>, 'app': <module
'ipykernel.kernelapp' from '/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-
DS/practice_env/lib/python3.10/site-packages/ipykernel/kernelapp.py'>}}
```

[24]:
```python
def print_twice(x):
    print(x)
    print(x)
x=print_twice("Hello")
```

```
Hello
Hello
```

[27]:
```python
print(x)
```

```
None
```

[25]:
```python
type(x)
```

[25]: NoneType

[26]:
```python
type(None)
```

[26]: NoneType

[28]:
```python
import turtle

# Create a Turtle object
my_turtle = turtle.Turtle()

# Set the drawing speed (optional)
my_turtle.speed(1) # 1 is slowest, 10 is fastest, 0 is no animation
```

```
# Draw a square
for _ in range(4):
    my_turtle.forward(100)  # Move forward 100 units
    my_turtle.right(90)     # Turn right 90 degrees

# Keep the window open until it's closed manually
turtle.done()
```

Encapsulation means hiding internal details of a class and only exposing what's necessary. It helps to protect important data from being changed directly and keeps the code secure and organized.

Python uses naming conventions rather than strict access modifiers:

Public: No underscore prefix (e.g., variable_name)

Protected: Single underscore prefix (e.g., _variable_name) - convention only

Private: Double underscore prefix (e.g., ___variable_name) - name mangling

Example: Employee Management System

[32]:
```
class Employee:
    def __init__(self, name, position, salary):
        self.name = name  # Public
        self.position = position  # Public
        self.__salary = salary  # Private
        self.__performance_rating = 0  # Private

    def get_salary(self):
        """Getter for salary"""
        return self.__salary

    def set_salary(self, new_salary, manager_approval=False):
        """Setter for salary with validation"""
        if manager_approval and new_salary > self.__salary:
            self.__salary = new_salary
            return True
        return False

    def update_performance(self, rating):
        """Update performance rating with validation"""
        if 1 <= rating <= 5:
            self.__performance_rating = rating
            return True
        return False

    def calculate_bonus(self):
        """Calculate bonus based on performance"""
        if self.__performance_rating >= 4:
            return self.__salary * 0.15
```

```
        elif self.__performance_rating >= 3:
            return self.__salary * 0.10
        else:
            return self.__salary * 0.05

    def get_employee_details(self):
        """Public method to get employee details"""
        return {
            "name": self.name,
            "position": self.position,
            "salary": self.__salary,
            "performance": self.__performance_rating,
            "bonus": self.calculate_bonus()
        }

# Using the Employee class
emp = Employee("Alice Smith", "Developer", 75000)

# Public access to name and position
print(f"Employee: {emp.name} - {emp.position}")

# Controlled access to salary
print(f"Current Salary: ${emp.get_salary()}")

# Trying to change salary without approval
if emp.set_salary(80000):
    print("Salary updated successfully")
else:
    print("Salary update failed - requires manager approval")

# Update performance and calculate bonus
emp.update_performance(4)
details = emp.get_employee_details()
print(f"Performance Bonus: ${details['bonus']:.2f}")
```

```
Employee: Alice Smith - Developer
Current Salary: $75000
Salary update failed - requires manager approval
Performance Bonus: $11250.00
```

Generalization in Python:

Generalization is a fundamental concept in Object-Oriented Programming (OOP) that involves creating a general class (superclass) that defines common attributes and behaviors, which can then be inherited by more specific classes (subclasses). It's about identifying common characteristics among classes and moving them up the inheritance hierarchy.

Types of Generalization:

1. Inheritance (IS-A relationship)
2. Interface Implementation
3. Abstract Classes

[277]:
```python
# General class (Superclass)
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def make_sound(self):
        """General method - to be overridden by subclasses"""
        return "Some generic animal sound"

    def eat(self, food):
        """Common behavior for all animals"""
        return f"{self.name} is eating {food}"

    def sleep(self):
        """Common behavior for all animals"""
        return f"{self.name} is sleeping"

# Specific classes (Subclasses)
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name, "Canine")
        self.breed = breed

    def make_sound(self):
        """Specialized behavior"""
        return "Woof! Woof!"

    def fetch(self, item):
        """Dog-specific behavior"""
        return f"{self.name} is fetching the {item}"

class Cat(Animal):
    def __init__(self, name, color):
        super().__init__(name, "Feline")
        self.color = color

    def make_sound(self):
        """Specialized behavior"""
        return "Meow! Meow!"

    def climb(self):
        """Cat-specific behavior"""
```

```
        return f"{self.name} is climbing a tree"

class Bird(Animal):
    def __init__(self, name, wing_span):
        super().__init__(name, "Avian")
        self.wing_span = wing_span

    def make_sound(self):
        return "Chirp! Chirp!"

    def fly(self):
        return f"{self.name} is flying with {self.wing_span}cm wingspan"

# Using the classes
animals = [
    Dog("Buddy", "Golden Retriever"),
    Cat("Whiskers", "Orange"),
    Bird("Tweety", 30)
]

print("=== Animal Sounds ===")
for animal in animals:
    print(f"{animal.name} ({animal.species}): {animal.make_sound()}")

print("\n=== Common Behaviors ===")
for animal in animals:
    print(animal.eat("food"))
    print(animal.sleep())
    print()

# Access specialized methods
dog = Dog("Rex", "German Shepherd")
print(dog.fetch("ball"))

cat = Cat("Mittens", "Black")
print(cat.climb())
```

```
=== Animal Sounds ===
Buddy (Canine): Woof! Woof!
Whiskers (Feline): Meow! Meow!
Tweety (Avian): Chirp! Chirp!

=== Common Behaviors ===
Buddy is eating food
Buddy is sleeping

Whiskers is eating food
```

```
Whiskers is sleeping

Tweety is eating food
Tweety is sleeping

Rex is fetching the ball
Mittens is climbing a tree
```

[ ]:

[33]:
```
"ankit"
```

[33]: 'ankit'

[34]:
```
'ankit'
```

[34]: 'ankit'

[35]:
```
""" Ankit
    is a
    good boy

"""
```

[35]: ' Ankit \n   is a \n   good boy\n\n'

[37]:
```
5/2
```

[37]: 2.5

[38]:
```
5//2
```

[38]: 2

[44]:
```
import math
math.floor(27/10)
```

[44]: 2

[49]:
```
x=2
y=3
if y==3:
    print("hi")
elif x==2:
    print("bye")
else:
    print("hi-bye")
```

```
hi
```

[53]:
```
# # infinite recursion
# def infinite_recursion(x):
#     if x==2:
#         return 1
#     return infinite_recursion(x//2)+x
# infinite_recursion(7)
```

[55]:
```
# float equality
0.33==1/3
```

[55]: False

[56]:
```
# we can't change the string object once defined in memory
s="ankit"
s[0]='A'
print(s)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[56], line 3
      1 # we can't change the string object once defined in memory
      2 s="ankit"
----> 3 s[0]='A'
      4 print(s)

TypeError: 'str' object does not support item assignment
```

[57]:
```
"ankit".upper()
```

[57]: 'ANKIT'

[68]:
```
"Ankit\T".casefold()
```

[68]: 'ankit\\t'

[69]:
```
'Kit' in 'ankit'
```

[69]: False

[70]:
```
'kit' in 'ankit'
```

[70]: True

[71]:
```
f=open('sample.txt','w')
f.write("This is a sample file.\nIt contains multiple lines of text.\nThis is␣
 ↪the third line.")
f.close()
```

[81]:
```
f=open('sample.txt','r')
```

[73]:
```
f.read()
```

[73]:
```
'This is a sample file.\nIt contains multiple lines of text.\nThis is the third
line.'
```

[76]:
```
f.readlines()
```

[76]:
```
['This is a sample file.\n',
 'It contains multiple lines of text.\n',
 'This is the third line.']
```

[82]:
```
f.readline()
```

[82]:
```
'This is a sample file.\n'
```

[83]:
```
f.readline()
```

[83]:
```
'It contains multiple lines of text.\n'
```

[84]:
```
f.readline()
```

[84]:
```
'This is the third line.'
```

[85]:
```
f.readline()
```

[85]:
```
''
```

[86]:
```
for x in []:
    print('This never happens.')
```

[87]:
```
# reduce
t=[1,2,3]
sum(t)
```

[87]:
```
6
```

[89]:
```
y=(1,2,3)
sum(y)
```

[89]:
```
6
```

[90]:
```
u={1,2,3}
sum(u)
```

[90]: 6

[93]:
```
d={1:9,2:8,3:7}
sum(d) # it will sum the keys of the dictionary
```

[93]: 6

[94]:
```
# map

def capitalize(words):
    res=[]
    for x in words:
        res.append(x.capitalize())
    return res
capitalize(['anKit','kiioo','guPTA'])
```

[94]: ['Ankit', 'Kiioo', 'Gupta']

[95]:
```
# reduce

def filter_words(words):
    res=[]
    for x in words:
        if x.isupper():
            res.append(x)
    return res
filter_words(['ankit','ANKIT','Kiioo','KIIOO'])
```

[95]: ['ANKIT', 'KIIOO']

[96]:
```
lst=[1,2,3]
lst.pop()
```

[96]: 3

[97]:
```
lst
```

[97]: [1, 2]

[98]:
```
lst=[1,2,3]
lst.pop(1)
```

[98]: 2

[99]:
```
lst
```

[99]:
```
[1, 3]
```

[100]:
```
lst=['ankit','kiioo','summi']
lst.remove('kiioo')
```

[101]:
```
lst
```

[101]:
```
['ankit', 'summi']
```

[103]:
```
sentence='I like kiioo'
x=sentence.split()
print(x)
y='@'.join(x)
print(y)
```

```
['I', 'like', 'kiioo']
I@like@kiioo
```

[ ]:
```
a='banana'
b='banana'
a is b # a and b are identical and equivalent
```

[ ]:
```
True
```

[107]:
```
a=[1,2,3]
b=[1,2,3]
a is b # a and  b are equivalent but not identical
```

[107]:
```
False
```

[ ]:
```
a=[1,2,3]
b=a
a is b # a and b are identical and equivalent. a and b refer to the same object.
```

[ ]:
```
True
```

An object with more than one reference has more than one name, so we say that the object is aliased. If the aliased object is mutable, changes made with one alias affect the other.

[113]:
```
a=[1,2,3]
b=a
x=a.extend([3,4,5])
```

[114]:
```
b
```

[114]:
```
[1, 2, 3, 3, 4, 5]
```

[115]:
```
a
```

[115]:
```
[1, 2, 3, 3, 4, 5]
```

[117]:
```
type(x)
```

[117]:
```
NoneType
```

[ ]:
```
a=[1,2,3]
b=a
x=a+[3,4,5] # + creates a new list
```

[119]:
```
a
```

[119]:
```
[1, 2, 3]
```

[120]:
```
b
```

[120]:
```
[1, 2, 3]
```

[121]:
```
x
```

[121]:
```
[1, 2, 3, 3, 4, 5]
```

A dictionary contains a collection of indices, which are called keys, and a collection of values. Each key is associated with a single value. The association of a key and a value is called a key-value pair or sometimes an item.

[122]:
```
d={}
d['ankit']='kiioo'
d
```

[122]:
```
{'ankit': 'kiioo'}
```

[124]:
```
d.update({
    'gupta':'t'
})
```

[125]:
```
d
```

[125]:
```
{'ankit': 'kiioo', 'gupta': 't'}
```

[126]:
```
d.update({'ankit':'summi'})
```

[127]:
```
d
```

[127]:
```
{'ankit': 'summi', 'gupta': 't'}
```

[128]:
```
d.keys(),d.values(),d.items()
```

[128]:
```
(dict_keys(['ankit', 'gupta']),
 dict_values(['summi', 't']),
 dict_items([('ankit', 'summi'), ('gupta', 't')]))
```

The order of the key-value pairs might not be the same. If you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable.

Python dictionaries use a data structure called a hashtable that has a remarkable property: the in operator takes about the same amount of time no matter how many items are in the dictionary.

[ ]:
```
sorted(d) # sort the keys
```

[ ]:
```
['ankit', 'gupta']
```

[145]:
```
for x in d:
    try:
        print(x)
    except:
        LookupError('value is not in dictionary')
```

```
ankit
gupta
```

Lists can be values in a dictionary, as this example shows, but they cannot be keys.

The system works fine if the keys are immutable. But if the keys are mutable, like lists, bad things happen because a=[1,2,3] and b=[1,2,3] are different objects and stored in different locations.

[150]:
```
x=True
def local_function():
    global x
    x=False # local function is changing the value of global variable
local_function()
print(x)
```

```
False
```

[153]:
```
lst=[1,2]
def local_function():
    lst.append(3)
local_function()
print(lst)
```

```
[1, 2, 3]
```

[154]:
```
lst=[1,2]
def local_function():
    lst=[1]
local_function()
print(lst)
```

```
[1, 2]
```

[155]:
```
lst=[1,2]
def local_function():
    global lst
    lst=[1]
local_function()
print(lst)
```

```
[1]
```

If a global variable refers to a mutable value, you can modify the value without declaring the variable. So you can add, remove and replace elements of a global list or dictionary, but if you want to reassign the variable, you have to declare it

[156]:
```
# tuples are immutable
st=('a') # string
t=('a',) # too-ple
print(type(st),type(t))
```

```
<class 'str'> <class 'tuple'>
```

[158]:
```
divmod(27,5) # function returing moe than one value in the form of tuple
```

[158]:
```
(5, 2)
```

Functions can take a variable number of arguments. A parameter name that begins with "*" gathers arguments into a tuple.

[ ]:
```
def func(*args): # *args can also work as gatherer.
    print(args)
func(1,3,5)
```

```
(1, 3, 5)
```

[163]:
```
func(1,3)
```

```
(1, 3)
```

[ ]:
```
def func(*args): # *args is working as gatherer.
    print(args)
func((1,3,5),(6,8))
```

```
((1, 3, 5), (6, 8))
```

[164]:
```
t=(27,7)
divmod(t)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[164], line 2
      1 t=(27,7)
----> 2 divmod(t)

TypeError: divmod expected 2 arguments, got 1
```

[166]:
```
t=(27,7)
divmod(*t) # *t is scattering the tuple
```

[166]:
```
(3, 6)
```

zip is a built-in function that takes two or more sequences and interleaves them. The name of the function refers to a zipper, which interleaves two rows of teeth.

[175]:
```
for x,y in zip(sorted([1,2,7]),reversed((7,8,9))):
    print(x,y)
```

```
1 9
2 8
7 7
```

[177]:
```
for pair in zip(sorted([1,2,7]),reversed((7,8,9))):
    print(pair)
```

```
(1, 9)
(2, 8)
(7, 7)
```

[176]:
```
s = 'abc'
t = [0, 1, 2]
for pair in zip(s, t):
    print(pair)
```

```
('a', 0)
('b', 1)
('c', 2)
```

A zip object is a kind of iterator, which is any object that iterates through a sequence. Iterators are similar to lists in some ways, but unlike lists, you can't use an index to select an element from an iterator.

[185]:
```
iter('abc').__next__()
```

[185]:
```
'a'
```

[186]:
```
for index,value in enumerate('abc'):
    print(index,value)
```

```
0 a
1 b
2 c
```

[187]:
```
dict(zip('abc',range(3)))
```

[187]:
```
{'a': 0, 'b': 1, 'c': 2}
```

Pseudorandom numbers are not truly random because they are generated by a deterministic computation, but just by looking at the numbers it is all but impossible to distinguish them from random.

The random module provides functions that generate pseudorandom numbers (which I will simply call "random" from here on). The function random returns a random float between 0.0 and 1.0 (including 0.0 but not 1.0).

[188]:
```
import random
random.random() # it will give random float number between 0.0 to 1.
```

[188]:
```
0.07038231112825843
```

The function randint takes parameters low and high and returns an integer between low and high (including both).

[190]:
```
random.randint(1,10) # it will give random integer number between 1 to 10 (both␣
↪inclusive)
```

[190]:
```
8
```

[196]:
```
lst=[1,2,3]
x=random.shuffle(lst) # it will shuffle the list randomly
print(lst)
```

```
[3, 1, 2]
```

[197]:
```
lst=[1,2,3]
random.choice(lst)
```

[197]:
```
3
```

[201]:
```
random.gauss(0,1) # it will give random float number based on normal␣
↪distribution with mean 0 and standard deviation 1
```

[201]:
```
2.291758854236265
```

[202]:
```
random.normalvariate(0,1) # it will give random float number based on normal␣
↪distribution with mean 0 and standard deviation 1
```

[202]:
```
-0.35333772038946576
```

[203]:
```
random.uniform(0,1) # it will give random float number between 0.0 to 1.0 (both␣
↪inclusive)
```

[203]:
```
0.6500581088913472
```

[208]:
```
f=open('sample.txt','w')
f.write(str(random.uniform(0,1)))
```

[208]:
```
19
```

[211]:
```
f=open('sample.txt','a') # 'a' mode will append the content at the end of the␣
↪file
f.write(str(111.11))
```

[211]:
```
6
```

[ ]:
```
f=open('sample.txt','w+') # 'w+' mode will overwrite the existing content
f.write(str(111.11))
```

[ ]:
```
6
```

[212]:
```
'my name is %s and my age is %d with height %g' % ('ankit',33,5.4)
```

[212]:
```
'my name is ankit and my age is 33 with height 5.4'
```

[213]:
```
import os
os.path.abspath('sample.txt')
```

[213]:
```
'/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-DS/1.Python/sample.txt'
```

[214]:
```
os.path.relpath('sample.txt')
```

[214]:
```
'sample.txt'
```

[215]:
```
os.path.realpath('sample.txt')
```

[215]:
```
'/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-DS/1.Python/sample.txt'
```

[218]:
```
os.path.exists('./sample1.txt')
```

[218]:
```
False
```

[220]:
```
os.path.exists('./sample1/ws')
```

[220]:
```
False
```

[221]:
```
os.path.isdir('sample.txt')
```

[221]:
```
False
```

[223]:
```
os.path.isfile('./sample.txt')
```

[223]:
```
True
```

[222]:
```
os.path.exists('.')
```

[222]:
```
True
```

[224]:
```
os.listdir('.')
```

[224]:
```
['sample.txt', 'Practice1.ipynb']
```

[225]:
```
os.listdir(os.getcwd())
```

[225]:
```
['sample.txt', 'Practice1.ipynb']
```

[229]:
```
os.path.join(os.getcwd(),'sample2.txt')
```

[229]:
```
'/home/ankit/Desktop/ml/InterviewPreparation/AI-ML-DS/1.Python/sample2.txt'
```

The pickle module can help. It translates almost any type of object into a string suitable for storage in a database, and then translates strings back into objects.

[230]:
```
import pickle
t1=pickle.dumps([1,2,3])
```

[ ]:
```
t1
```

[ ]:
```
b'\x80\x04\x95\x0b\x00\x00\x00\x00\x00\x00\x00]\x94(K\x01K\x02K\x03e.'
```

[232]:
```
type(t1)
```

[232]:
```
bytes
```

[233]:
```
t2=pickle.loads(t1)
```

[234]:
```
t2
```

[234]:
```
[1, 2, 3]
```

[ ]:
```
t1 is t2 # t1 and t2 are equivalent but not identical.
# In other words, pickling and then unpickling has the same effect as copying␣
 ↪the object.
```

[ ]:
```
False
```

Any program that you can launch from the shell can also be launched from Python using a pipe object, which represents a running program.

For example, the Unix command ls -l normally displays the contents of the current directory in long format. You can launch ls with os.popen:

[236]:
```
import os
os.popen('ls -l').read()
```

[236]:
```
'total 100\n-rw-rw-r-- 1 ankit ankit 97739 Aug 25 23:52 Practice1.ipynb\n-rw-
rw-r-- 1 ankit ankit     6 Aug 25 23:36 sample.txt\n'
```

The argument is a string that contains a shell command.The return value is an object that behaves like an open file. You can read the output from the ls process one line at a time with readline or get the whole thing at once with read

[238]:
```
os.popen('ls -l').readlines()
```

[238]:
```
['total 104\n',
 '-rw-rw-r-- 1 ankit ankit 99052 Aug 25 23:55 Practice1.ipynb\n',
 '-rw-rw-r-- 1 ankit ankit     0 Aug 25 23:53 sample2.txt\n',
 '-rw-rw-r-- 1 ankit ankit     6 Aug 25 23:36 sample.txt\n']
```

[237]:
```
os.popen('touch sample2.txt').read()
```

[237]:
```
''
```

[239]:
```
os.popen('rm sample2.txt').read()
```

[239]:
```
''
```

if ___name___ == '___main___':

___name___ is a built-in variable that is set when the program starts.

If the program is running as a script, ___name___ has the value ___main___

If you import a module that has already been imported, Python does nothing. It does not re-read the file, even if it has changed.

If you want to reload a module, you can use the built-in function reload.

[242]:
```
s = '1 2\t 3\n 4'
print(s)
print(repr(s))
```

```
1 2	 3
 4
'1 2\t 3\n 4'
```

A programmer-defined type is also called a class. Defining a class named Point creates a class object. Because Point is defined at the top level, its "full name" is **main**.Point.

[243]:
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

[245]:
```
point=Point(2,3)
```

[246]:
```
type(point)
```

[246]:
```
__main__.Point
```

point=Point()

The return value is a reference to a Point object.

Creating a new object is called instantiation, and the object is an instance of the class. When you print an instance, Python tells you what class it belongs to and where it is stored in memory.

Objects are mutable. You can change the state of an object by making an assignment to one of its attributes.

The copy module contains a function called copy that can duplicate any object

[248]:
```
p1=Point(2,3)
import copy
p2=copy.copy(p1) # shallow copy
p3=copy.deepcopy(p1) # deep copy
```

[252]:
```
p1
```

[252]:
```
<__main__.Point at 0x70f792bd6bc0>
```

[251]:
```
p2
```

[251]:
```
<__main__.Point at 0x70f7917abbb0>
```

[253]:
```
p3
```

[253]:
```
<__main__.Point at 0x70f7917a8370>
```

p1 and p2 contain the same data, but they are not the same Point.

If you use copy.copy to duplicate a Rectangle, you will find that it copies the Rectangle object but not the embedded Point.

rect1 —> Point <— rect2

This operation is called a shallow copy because it copies the object and any references it contains, but not the embedded objects.

Fortunately, the copy module provides a method named deepcopy that copies not only the object but also the objects it refers to, and the objects they refer to, and so on. You will not be surprised to learn that this operation is called a deep copy.

[255]:
```
class Time:
    def print_time(time):
        print('%.2d:%.2d:%.2d' % (time.hour, time.minute, time.second))
```

Now there are two ways to call print_time. The first (and less common) way is to use function syntax:

start = Time()
Time.print_time(start)

B. The second (and more concise) way is to use method syntax:

start.print_time()

[256]:
```
start=Time()
start.hour=9
start.minute=45
start.second=0
start.print_time()
```

```
09:45:00
```

By convention, the first parameter of a method is called self, so it would be more common to write print_time like this:

[257]:
```
class Time:
    def print_time(self):
        print('%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second))
```

[258]:
```
start=Time()
start.hour=9
start.minute=45
start.second=0
start.print_time()
```

```
09:45:00
```

a positional argument is an argument that doesn't have a parameter name; that is, it is not a keyword argument.

sketch(parrot, cage, dead=True)

parrot and cage are positional, and dead is a keyword argument.

The init method (short for "initialization") is a special method that gets invoked when an object is instantiated. Its full name is ___init___.

[259]:
```
class Time:
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
```

[260]:
```
start=Time()
start.hour
```

[260]: 0

___str___ is a special method, like ___init___, that is supposed to return a string representation of an object.

[261]:
```
class Time:
    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second
    def __str__(self):
        print('%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second))
    def print_time(self):
        print('%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second))
```

[262]:
```
start=Time()
print(start.hour, start.minute, start.second)
```

```
0 0 0
```

[263]:
```
start.print_time()
```

```
00:00:00
```

[264]:
```
start.__str__()
```

```
00:00:00
```

[266]:
```
start.__init__()
```

Operator Overloading

By defining other special methods, you can specify the behavior of operators on programmer-defined types. For example, if you define a method named ___add___ for the Time class, you can use the

+ operator on Time objects.

add it in time class

```
def __add__(self, other):
    seconds = self.time_to_int() + other.time_to_int()
    return int_to_time(seconds)
```

When you apply the + operator to Time objects, Python invokes ___add___.

Changing the behavior of an operator so that it works with programmer-defined types is called operator overloading. For every operator in Python there is a corresponding special method, like ___add___.

other example: ___lt___ (less than)

Polymorphism

Functions that work with several types are called polymorphic. Polymorphism can facilitate code reuse. For example, the built-in function sum, which adds the elements of a sequence, works as long as the elements of the sequence support addition.

[267]:
```
[1,2,4]+[2,3,5]
```

[267]:
```
[1, 2, 4, 2, 3, 5]
```

[268]:
```
sum([1,2,4]+[2,3,5])
```

[268]:
```
17
```

[269]:
```
sum((1,2))
```

[269]:
```
3
```

Inheritance is the ability to define a new class that is a modified version of an existing class

Variables which are defined inside a class but outside of any method, are called class attributes because they are associated with the class object.
Class_Name.class_attribute_name

A class diagram is a more abstract representation of the structure of a program. Instead of showing individual objects, it shows classes and the relationships between them.

Objects in one class might contain references to objects in another class. For example, each Rectangle contains a reference to a Point, and each Deck contains references to many Cards. This kind of relationship is called HAS-A, as in, “a Rectangle has a Point.”

One class might inherit from another. This relationship is called IS-A, as in, “a Hand is a kind of a Deck.”

The arrow with a hollow triangle head represents an IS-A relationship

The standard arrow head represents a HAS-A relationship

Generator expressions are similar to list comprehensions, but with parentheses instead of square brackets:

[270]:
```
g = (x**2 for x in range(5))
g.__next__()
```

[270]:
```
0
```

The result is a generator object that knows how to iterate through a sequence of values. But unlike a list comprehension, it does not compute the values all at once; it waits to be asked. The built-in function next gets the next value from the generator:

[271]:
```
g.__next__()
```

[271]:
```
1
```

[272]:
```
g.__next__()
```

[272]:
```
4
```

[273]:
```
g.__next__()
```

[273]:
```
9
```

[274]:
```
next(g)
```

[274]:
```
16
```

[275]:
```
next(g)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Cell In[275], line 1
----> 1 next(g)

StopIteration:
```

The * operator doesn't gather keyword arguments. To gather keyword arguments, you can use the ** operator.

[276]:
```
def printall(*args, **kwargs):
    print(args, kwargs)
printall(1,2,3,a=4,b=5)
```

```
(1, 2, 3) {'a': 4, 'b': 5}
```

[ ]:
```

```