```python
def greeting():
    print("Hello world")
    print('this is the body of the function')
    print('hello function!')

def greet():
    return "hello world"

def _greet(name):       # "name" is a variable or a parameter
    print(f"Hello {name}")

def greet_someone(name):
    return f"hello {name}"

def add(a, b):
    return a + b

# function with default values to the arguments
def _greet(name="world"):       # "name" is optional argument
    print(f"Hello {name}")

def greeting_(name, age, pay):
    # name, age and pay are called positional arguments
    print(f"Hello {name} you are {age} years of age and you get ${pay} as pay

def greeting_(name, age=26, pay=1000):
    # name, age and pay are called positional arguments
```

```
28      print(f"Hello {name} you are {age} years of age and you get ${pay} as pay
29
30  def greet(name, debug=False):
31      if debug:       # if debug == True
32          print("You called greet function")
33      print(f"hello {name}")
34
35  def greet(name, reverse=False, debug=False):
36      if debug:
37          print("You called greet function")
38      if reverse:
39          return f"hello {name[::-1]}"        # exits the function.
40      return f"hello {name}"
41
42  def parse_string(line, delimiter=","):
43      parts = line.split(delimiter)
44      return parts
45
46  def greeting_(name, *, age, pay):
47      # the parameters that are after * are to be called using keyword only
48      # age and pay are KEYWORD ONLY Arguments, i.e. the value for age and pay
49      print(f"Hello {name} you are {age} years of age and you get ${pay} as pay
50
51  def greet(name, *, reverse=False, debug=False):
52      if debug:
53          print("You called greet function")
54      if reverse:
55          return f"hello {name[::-1]}"        # exits the function.
56      return f"hello {name}"
57
58  def greet(*, name, reverse=False, debug=False):
59      if debug:
60          print("You called greet function")
61      if reverse:
62          return f"hello {name[::-1]}"        # exits the function.
63      return f"hello {name}"
64
65  def greet(name, /, *, reverse=False, debug=False):
66      # the parameters that appears before "/" is positional only arguments
67      if debug:
68          print("You called greet function")
69      if reverse:
70          return f"hello {name[::-1]}"        # exits the function.
71      return f"hello {name}"
72
```

```
73  # ------------------------------------------------------------------------
74  # Variable number of positional (*args)
75  # * is used to grab arbitrary number of positional arguments!
76  def add(*args):
77      total = 0.0
78      # by convention we call variable number of positional arguments using par
79      # * is used to collect excess arguments
80      for item in args:
81          total = total + item
82      return total
83
84  print(add())
85  print(add(1))
86  print(add(1, 2))
87  print(add(10, 30, 45))
88  print(add(1000, 46273, 84545, 9834958, 4587583))
89  nums = [1, 2, 3, 4]
90  print(add(*nums))
91  # ------------------------------------------------------------------------
92  def greet(*names):
93      for name in names:
94          print(f'hello {name}')
95
96  greet("steve")      # one argument
97  greet("steve", "bill")  # two arguments
98  greet("steve", "bill", "gates", "jobs", "joe")      # five arguments
99  greet() # zero arguments
100 # ------------------------------------------------------------------------
101 # "a" is mandatory argument.
102 def func(a, *args):
103     print(a, args)
104
105 # Function that accepts any number of keyword arguments
106 def func(**kwargs):
107     print(kwargs)
108
109 # Keyword variable arguments (**kwargs)
110 def func2(a, **kwargs):
111     print(a, kwargs)
112
113 # Variable number of keyword arguments (**kwargs)
114 # * is used to grab arbitrary number of positional arguments!
115 def greet(name, **info):
116     print(f'hello {name} below is your information')
117     for key, value in info.items():
```

```
118         print(f'{key}: {value}')
119 
120 greet("Steve", phone=1234567890, city="Bangalore", country="India")     # Thr
121 greet("Steve", state="Karnataka")      # One arbitrary keyword argument
122 greet("Steve")      # Zero keyword argument
123 # ------------------------------------------------------------------------
124 # Combining both
125 def anyargs(*args, **kwargs):
126     print(args)     # Tuple
127     print(kwargs)   # Dictionary
128 
129 anyargs(1, 2, 3, fname='steve', lname='jobs')
130 
131 # Unpacking arguments
132 def greet(name, age, pay):
133     print(f'Hello {name} you are {age} years and you have {pay} pay')
134 
135 data = ['Steve', 26, 1000]
136 
137 greet(data[0], data[1], data[2])
138 greet(*data)    # Equivelent to greet("Steve", 26, 1000)
139 
140 d_data = {"name": "steve", "age": 26, "pay": 1000}
141 greet(d_data['name'], d_data['age'], d_data['pay'])
142 greet(**d_data)     # Equivelent to greet(name="Steve", age=26, pay=1000)
143 
144 # Returning Multiple Values from a Function
145 def div(a, b):
146       r = a % b
147       q = a / b
148       return r, q  # returns a tuple
149 
150 remainder, quotent = div(4, 2)
151 # ------------------------------------------------------------------------
152 # passing reference of one function to another function
153 def greet():
154     return "Hello world"
155 
156 def greeting(name):
157     return f"hello {name}"
158 
159 def add(a, b):
160     return a + b
161 
162 def mul(a, b, c):
```

```
163     return a * b * c
164 
165 # "spam" executes or calls the function that is being passed to it.
166 # it is "spam"'s responsibility to call the function with correct signature
167 def spam(func, *args, **kwargs):
168     result = func(*args, **kwargs)
169     return result
170 
171 # In the below function, wrapper is the one which calls "func". "spam" return
172 # to the inner function
173 def spam(func):
174     def wrapper(*args, **kwargs):
175         result = func(*args, **kwargs)
176         return result
177     return wrapper
178 # ------------------------------------------------------------------------
179 # Function Annotations
180 # Annotations are only type hints. But it does not enforce type check!
181 def add(a: int, b: int) -> int:
182     return a + b
183 
184 def greetings(name: str, age: int, pay: float, isMarried: bool) -> None:
185     print(f"Hello {name} You are {age} years old and your is {pay}")
186     if isMarried:
187         print('Congratulations')
188     else:
189         print('You are free')
190 
191 def greet(name: str = "Spider") -> None:
192     print(f'Hello {name}')
193 
194 # ------------------------------------------------------------------------
195 # Default values are evaluated only once at the time of function defnition
196 age = 10
197 def myinfo(my_name, my_age=age):
198     print(my_name, my_age)
199 
200 print(myinfo('steve', my_age=50))      # Prints (steve, 50)
201 print(myinfo('steve'))      # Prints(steve, 10)
202 age = 20
203 print(myinfo('steve'))      # Prints (steve, 10)
204 
205 # Default arguments are evaluated only ONCE
206 """
207     names=[ ] in the function declaration makes Python essentially do this:
```

```
    1. This function has a parameter named "names" its default argument is [
        let's set this particular [ ] aside and use it anytime there's no par
    2. Every time the function is called, create a variable "names", and assi
        the passed parameter or the value we set aside earlier
"""
def func(names=[ ]):  # making mutable data as default value
    names.append(1)
    return names

func()  # returns [1]
func()  # returns [1, 1]
func()  # returns [1, 1, 1]
func([10, 20, 30, 40]) # returns [10, 20, 30, 40, 1]

# Correct version
def func(names = None):
    if names is None:
        names = [ ]
    names.append(1)
    return names

func()  # returns [1]
func()  # returns [1]
func()  # returns [1]
func([10, 20, 30, 40]) # returns [10, 20, 30, 40]
# ------------------------------------------------------------------------
# lambda expressions/functions
# General Syntax
# lambda args: expression         # (expression is something which evaluates

def add(a, b):
    return a+b  # Single expression function

def func(a, b):
    return a ** 2 + b ** 2 + 2 * a * b

def func2(a, b, c):
    return 2*a + 3*b + 4*c

def last(item):
    return item[-1]

# lambda expressions or ananoymous functions
# lambda args_list: expression
add = lambda a, b: a + b
```

```
253 func = lambda a, b: a ** 2 + b ** 2 + 2 * a * b
254 func2 = lambda a, b, c: 2*a + 3*b + 4*c
255 last = lambda item: item[-1]
256 # ----------------------------------------------------------------------------
257 # Passing Immutable data to functions
258 a = 10
259 def spam(some_number):
260     some_number = some_number + 1
261     print(some_number)
262 
263 spam(a)  # Prints 11
264 print(a)  # Prints 10
265 # ----------------------------------------------------------------------------
266 # Passing Mutable data to functions
267 a = [10]
268 
269 def spam(some_list):
270     some_list.append(20)
271     print(some_list)
272 
273 spam(a)  # Prints [10, 20]
274 print(a)  # Prints [10, 20]
275 
276 # ----------------------------------------------------------------------------
277 numbers = [5, 1, 3, 2, 0, 7, 6]
278 
279 def smallest(items, n):
280     items.sort()
281     return items[:n]
282 
283 """
284 1. When an Immutable object is passed to a function, python acts as
285 call by value.
286 2. When a Mutable object is passed to a function, python acts as call
287 by reference.
288 3. Python is neither call by value nor call by reference. It all depends
289 on the type of the object that is being passed to the function
290 """
291 
292 # ----------------------------------------------------------------------------
293 a = 10  # Global variable
294 
295 # defining the function
296 def func(b):
297     return a + b
```

```
298
299 a = 20      # Re-assigning new value to the global variable
300
301 func(10) # prints 30    # executing the function
302 # this is called as late-binding
303 # The "func" uses the value of "a" that happens to be at the time of evaluati
304
305 # ------------------------------------------------------------------------
306 _a = 200
307 # If it is important to use the value of the variable at the time of function
308 # use default arguments.
309 def func2(b, a=_a):
310     return a + b
311
312 _a = 100      # Re-assigning new value to the global variable
313
314 func2(10)       # prints 210
315 # In the function "func2" the parameter "a" takes the value that is assigned
316 # ------------------------------------------------------------------------
317
318 # You can attach arbitrary attributes to the function after the function is d
319
320 def add(a, b):
321     add.count += 1
322     return a+b
323
324 def sub(a, b):
325     sub.count += 1
326     return a-b
327
328 # Attach the attributes to the function
329 add.count = 0
330 sub.count = 0
331
332 add(1, 2)
333 add(10, 20)
334
335 print(add.count)    # prints count = 2
336
337 sub(1, 2)
338 sub(1, 3)
339 sub(1, 4)
340 sub(1, 5)
341
342 print(sub.count)    # prints count = 4
```