

NAME - SUTIRTHA SAMANTA
SEC – A ROLL - 64
ENROLLMENT NO.- 12023006015045
SUBJECT – DSDA LAB5 ASSIGNMENT

ASSIGNMENT-5(DSDA LAB)

1) The following is a list of 10 students ages:

ages = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24]

I. Sort the list and find the min and max age

II. Add the min age and the max age again to the list

III. Find the median age (one middle item or two middle items divided by two)

IV. Find the average age (sum of all items divided by their number)

V. Find the range of the ages (max minus min)

VI. Compare the value of (min - average) and (max - average), use `_abs()` method

```
ages = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24]
```

```
# I. Sort the list and find the min and max age
```

```
ages.sort()
```

```
min_age = ages[0]
```

```
max_age = ages[-1]
```

```
print(f"Sorted ages: {ages}")
```

```
print(f"Min age: {min_age}")
```

```
print(f"Max age: {max_age}")
```

```
# II. Add the min age and the max age again to the list
```

```
ages.append(min_age)
```

```
ages.append(max_age)
```

```
print(f"Ages after adding min and max again: {ages}")
```

```
# III. Find the median age (one middle item or two middle items divided by two)
```

```
n = len(ages)
```

```
if n % 2 == 0:
```

```
    median_age = (ages[n//2 - 1] + ages[n//2]) / 2
```

else:

```
    median_age = ages[n//2]

print(f"Median age: {median_age}")

# IV. Find the average age (sum of all items divided by their number)

average_age = sum(ages) / len(ages)

print(f"Average age: {average_age}")

# V. Find the range of the ages (max minus min)

age_range = max_age - min_age

print(f"Range of ages: {age_range}")

# VI. Compare the value of (min - average) and (max - average), use _abs() method

min_minus_avg = abs(min_age - average_age)

max_minus_avg = abs(max_age - average_age)

print(f"|Min age - Average age|: {min_minus_avg}")

print(f"|Max age - Average age|: {max_minus_avg}")
```

OUTPUT-

```
sorted ages: [19, 19, 20, 22, 24, 24, 24, 25, 25, 26]
Min age: 19
Max age: 26
Ages after adding min and max again: [19, 19, 20, 22, 24, 24, 24, 25, 25, 26, 19, 26]
Median age: 24.0
Average age: 22.75
Range of ages: 7
|Min age - Average age|: 3.75
|Max age - Average age|: 3.25
```

2) Iterate through the list, ['Python', 'Numpy', 'Pandas', 'Django', 'Flask'] using a for loop and print out the items.

```
frameworks = ['Python', 'Numpy', 'Pandas', 'Django', 'Flask']

for item in frameworks: print(item)
```

OUTPUT-

```
Python
Numpy
Pandas
Django
Flask
```

3) Create fruits, vegetables and animal products tuples.

I. Join the three tuples and assign it to a variable called food_stuff_tp.

II. Change the about food_stuff_tp tuple to a food_stuff_lt list

III. Slice out the middle item or items from the food_stuff_tp tuple or food_stuff_lt list.

IV. Slice out the first three items and the last three items from food_staff_t tuple

V. Delete the food_staff_tp tuple completely

```
fruits = ('apple', 'banana', 'cherry', 'date', 'elderberry')
vegetables = ('carrot', 'broccoli', 'spinach', 'pepper', 'onion')
animal_products = ('milk', 'cheese', 'butter', 'yogurt', 'eggs')

# I. Join the three tuples and assign it to a variable called food_stuff_tp
food_stuff_tp = fruits + vegetables + animal_products

print("Joined tuple:", food_stuff_tp)

# II. Change the food_stuff_tp tuple to a food_stuff_l tuple
food_stuff_l = list(food_stuff_tp)

print("Converted list:", food_stuff_l)

# III. Slice out the middle item or items from the food_stuff_tp tuple
middle_index = len(food_stuff_tp) // 2

if len(food_stuff_tp) % 2 == 0:
    middle_items = food_stuff_tp[middle_index - 1:middle_index + 1]
else:
    middle_items = food_stuff_tp[middle_index]

print("Middle item(s) from tuple:", middle_items)

# IV. Slice out the first three items and the last three items from food_stuff_l tuple
first_three_items = food_stuff_l[:3]
last_three_items = food_stuff_l[-3:]

print("First three items from list:", first_three_items)
print("Last three items from list:", last_three_items)

# V. Delete the food_stuff_tp tuple completely
del food_stuff_tp

print("Food stuff tuple after deletion:", food_stuff_tp)
```

OUTPUT-

```
>>> Joined tuple: ('apple', 'banana', 'cherry', 'date', 'elderberry', 'carrot', 'broccoli', 'spinach', 'pepper', 'onion', 'milk', 'cheese', 'butter', 'yogurt', 'eggs')
>>> Converted list: ['apple', 'banana', 'cherry', 'date', 'elderberry', 'carrot', 'broccoli', 'spinach', 'pepper', 'onion', 'milk', 'cheese', 'butter', 'yogurt', 'eggs']
>>> Middle item(s) from tuple: spinach
>>> First three items from list: ['apple', 'banana', 'cherry']
>>> Last three items from list: ['butter', 'yogurt', 'eggs']
>>>
```

4) Create a set given below

`it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}`

`A = {19, 22, 24, 20, 25, 26}`

`B = {19, 22, 20, 25, 26, 24, 28, 27}`

`age = [22, 19, 24, 25, 26, 24, 25, 24]`

I. Find the length of the set `it_companies`

II. Add `'Twitter'` to `it_companies`

III. Insert multiple IT companies at once to the set `it_companies`

IV. Remove one of the companies from the set `it_companies`

V. What is the difference between `remove` and `discard`

`it_companies = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}`

`A = {19, 22, 24, 20, 25, 26}`

`B = {19, 22, 20, 25, 26, 24, 28, 27}`

`age = [22, 19, 24, 25, 26, 24, 25, 24]`

I. Find the length of the set `it_companies`

`length_it_companies = len(it_companies)`

`print("Length of it_companies:", length_it_companies)`

II. Add `'Twitter'` to `it_companies`

`it_companies.add('Twitter')`

`print("it_companies after adding 'Twitter':", it_companies)`

III. Insert multiple IT companies at once to the set `it_companies`

`new_companies = {'LinkedIn', 'Snapchat', 'Slack'}`

`it_companies.update(new_companies)`

`print("it_companies after adding multiple companies:", it_companies)`

IV. Remove one of the companies from the set `it_companies`

`it_companies.remove('Snapchat')` # Note: 'Snapchat' must be present in the set to remove it

`print("it_companies after removing 'Snapchat':", it_companies)`

V. Difference between remove and discard

try:

```
it_companies.remove('NonExistentCompany')
```

except KeyError:

```
print("remove() raised a KeyError because 'NonExistentCompany' was not found.")
```

```
it_companies.discard('NonExistentCompany')
```

```
print("it_companies after discard operation:", it_companies)
```

OUTPUT-

```
>>> Length of it_companies: 7
it_companies after adding 'Twitter': {'IBM', 'Amazon', 'Microsoft', 'Apple', 'Twitter', 'Oracle', 'Facebook', 'Google'}
it_companies after adding multiple companies: {'Slack', 'LinkedIn', 'Oracle', 'Microsoft', 'Google', 'Twitter', 'Amazon', 'Apple', 'Snapchat', 'IBM', 'Facebook'}
it_companies after removing 'Snapchat': {'Slack', 'LinkedIn', 'Oracle', 'Microsoft', 'Google', 'Twitter', 'Amazon', 'Apple', 'IBM', 'Facebook'}
remove() raised a KeyError because 'NonExistentCompany' was not found.
it_companies after discard operation: {'Slack', 'LinkedIn', 'Oracle', 'Microsoft', 'Google', 'Twitter', 'Amazon', 'Apple', 'IBM', 'Facebook'}
```

5) From the above sets A and B

I. Join A and B

II. Find A intersection B

III. Is A subset of B

IV. Are A and B disjoint sets

V. Join A with B and B with A

VI. What is the symmetric difference between A and B

VII. Delete the sets completely

A = {19, 22, 24, 20, 25, 26}

B = {19, 22, 20, 25, 26, 24, 28, 27}

I. Join A and B (Union)

```
union_ab = A | B
```

```
print("Union of A and B:", union_ab)
```

II. Find A intersection B

```
intersection_ab = A & B
```

```
print("Intersection of A and B:", intersection_ab)
```

III. Is A subset of B

```
is_subset = A.issubset(B)
```

```
print("Is A a subset of B:", is_subset)
```

IV. Are A and B disjoint sets

```
are_disjoint = A.isdisjoint(B)
```

```
print("Are A and B disjoint sets:", are_disjoint)
```

V. Join A with B and B with A (Union in both directions)

```
union_ab_ba = A | B | B | A # Union is commutative, so this is effectively the same as A | B
```

```
print("Union of A with B and B with A:", union_ab_ba)
```

VI. Symmetric difference between A and B

```
symmetric_difference_ab = A ^ B
```

```
print("Symmetric difference between A and B:", symmetric_difference_ab)
```

VII. Delete the sets completely

```
del A
```

```
del B
```

```
print("Set A after deletion:", A)
```

```
print("Set B after deletion:", B)
```

OUTPUT-

```
Union of A and B: {19, 20, 22, 24, 25, 26, 27, 28}
Intersection of A and B: {19, 20, 22, 24, 25, 26}
Is A a subset of B: True
Are A and B disjoint sets: False
Union of A with B and B with A: {19, 20, 22, 24, 25, 26, 27, 28}
Symmetric difference between A and B: {27, 28}
```

6) Create an empty dictionary called dog. Add name, color, breed, legs, age to the dog dictionary.

```
dog = {}
```

```
dog['name'] = 'Buddy'
```

```
dog['color'] = 'Brown'
```

```
dog['breed'] = 'Golden Retriever'
```

```
dog['legs'] = 4
```

```
dog['age'] = 5
```

```
print("Dog dictionary:", dog)
```

OUTPUT-

```
Dog dictionary: {'name': 'Buddy', 'color': 'Brown', 'breed': 'Golden Retriever', 'legs': 4, 'age': 5}
```

7) Create a student dictionary and add first_name, last_name, gender, age, marital status, skills,

country, city and address as keys for the dictionary

I. Get the length of the student dictionary

II. Get the value of skills and check the data type, it should be a list

III. Modify the skills values by adding one or two skills

IV. Get the dictionary keys as a list

V. Get the dictionary values as a list

VI. Change the dictionary to a list of tuples using _items()_ method

VII. Delete one of the items in the dictionary

VIII. Delete one of the dictionaries

```
student = {  
    'first_name': 'John',  
    'last_name': 'Doe',  
    'gender': 'Male',  
    'age': 20,  
    'marital_status': 'Single',  
    'skills': ['Python', 'Java', 'SQL'],  
    'country': 'USA',  
    'city': 'New York',  
    'address': '123 Main St'  
}
```

I. Get the length of the student dictionary

```
length_student_dict = len(student)  
  
print("Length of student dictionary:", length_student_dict)
```

II. Get the value of skills and check the data type

```
skills = student['skills']  
  
print("Value of skills:", skills)  
  
print("Data type of skills:", type(skills))
```

```

# III. Modify the skills values by adding one or two skills
student['skills'].extend(['Django', 'Machine Learning'])

print("Modified skills:", student['skills'])

# IV. Get the dictionary keys as a list
keys_list = list(student.keys())

print("Dictionary keys as a list:", keys_list)

# V. Get the dictionary values as a list
values_list = list(student.values())

print("Dictionary values as a list:", values_list)

# VI. Change the dictionary to a list of tuples using items() method
items_list = list(student.items())

print("Dictionary as a list of tuples:", items_list)

# VII. Delete one of the items in the dictionary
del student['address']

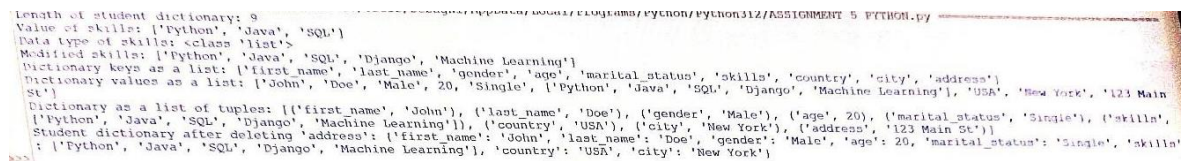
print("Student dictionary after deleting 'address':", student)

# VIII. Delete the dictionary completely
del student

# print("Student dictionary after deletion:", student)

```

OUTPUT-



```

Length of student dictionary: 9
Value of skills: ['Python', 'Java', 'SQL']
Data type of skills: <class 'list'>
Modified skills: ['Python', 'Java', 'SQL', 'Django', 'Machine Learning']
Dictionary keys as a list: ['first_name', 'last_name', 'gender', 'age', 'marital_status', 'skills', 'country', 'city', 'address']
Dictionary values as a list: ['John', 'Doe', 'Male', 20, 'Single', ['Python', 'Java', 'SQL', 'Django', 'Machine Learning'], 'USA', 'New York', '123 Main St']
Dictionary as a list of tuples: [(('first_name', 'John'), ('last_name', 'Doe'), ('gender', 'Male'), ('age', 20), ('marital_status', 'Single'), ('skills', ['Python', 'Java', 'SQL', 'Django', 'Machine Learning']), ('country', 'USA'), ('city', 'New York'), ('address', '123 Main St'))]
Student dictionary after deleting 'address': {'first_name': 'John', 'last_name': 'Doe', 'gender': 'Male', 'age': 20, 'marital_status': 'Single', 'skills': ['Python', 'Java', 'SQL', 'Django', 'Machine Learning'], 'country': 'USA', 'city': 'New York'}

```

8) Create a person dictionary.

```

person={
'first_name': 'Asabeneh',
'last_name': 'Yetayeh',
'age': 250,
'country': 'Finland',

```



```
'is_married': True,  
'skills': ['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],  
'address': {  
  'street': 'Space street',  
  'zipcode': '02210'  
}  
}
```

I. Check if the person dictionary has skills key, if so print out the middle skill in the skills list.

II. Check if the person dictionary has skills key, if so check if the person has skill and 'Python' skill and print out the result.

III. If a person skills has only JavaScript and React, print('He is a front end developer'), if the

person skills has Node, Python, MongoDB, print('He is a backend developer'), if the person

skills has React, Node and MongoDB, Print('He is a fullstack developer'), else print('unknown

title') - for more accurate results more conditions can be nested!

IV. If the person is married and if he lives in Finland, print the information in the following

format:

```
```py
```

**Asabeneh Yetayeh lives in Finland. He is married.**

```
person = {
 'first_name': 'Asabeneh',
 'last_name': 'Yetayeh',
 'age': 250,
 'country': 'Finland',
 'is_married': True,
```

```
'skills': ['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],
'address': {
 'street': 'Space street',
 'zipcode': '02210'
}
}
```

# I. Check if the person dictionary has skills key, if so print out the middle skill in the skills list.

if 'skills' in person:

```
skills = person['skills']

middle_index = len(skills) // 2

middle_skill = skills[middle_index]

print("Middle skill:", middle_skill)
```

# II. Check if the person dictionary has skills key, if so check if the person has 'Python' skill and print out the result.

if 'skills' in person:

```
has_python = 'Python' in person['skills']

print("Has Python skill:", has_python)
```

# III. Check skills and print the appropriate title

if 'skills' in person:

```
skills = person['skills']

if 'JavaScript' in skills and 'React' in skills and len(skills) == 2:

 print('He is a front end developer')

elif 'Node' in skills and 'Python' in skills and 'MongoDB' in skills and len(skills) == 3:

 print('He is a backend developer')

elif 'React' in skills and 'Node' in skills and 'MongoDB' in skills and len(skills) == 3:

 print('He is a fullstack developer')

else:

 print('Unknown title')
```

# IV. If the person is married and lives in Finland, print the information in the specified format.

if person['is\_married'] and person['country'] == 'Finland':

```
print(f"{person['first_name']} {person['last_name']} lives in {person['country']}. He is married.")
```

## OUTPUT-

```
Middle skill: Node
Has Python skill: True
Unknown title
Asabeneh Yetayeh lives in Finland. He is married.
>>>|
```

### 9) Print the season name of the year based on the month number using a dictionary.

```
seasons = {
 1: 'Winter',
 2: 'Winter',
 3: 'Spring',
 4: 'Spring',
 5: 'Spring',
 6: 'Summer',
 7: 'Summer',
 8: 'Summer',
 9: 'Fall',
 10: 'Fall',
 11: 'Fall',
 12: 'Winter'
}

def get_season(month_number):
 return seasons.get(month_number, "Invalid month number")

test_months = [1, 3, 6, 9, 12, 13] # Includes an invalid month number for testing
for month in test_months:
 print(f'Month {month}: {get_season(month)}')
```

## OUTPUT-

```
Month 1: Winter
Month 3: Spring
Month 6: Summer
Month 9: Fall
Month 12: Winter
Month 13: Invalid month number
>
```

---

## MISCELLANEOUS

**10) Write a Python program to find duplicate elements in a 1D array and find their frequency of occurrence.**

```
from collections import Counter

array = [1, 2, 3, 4, 2, 3, 5, 1, 6, 1, 2]

frequency = Counter(array)

duplicates = {key: value for key, value in frequency.items() if value > 1}

print("Duplicate elements and their frequency:")

for key, value in duplicates.items():

 print(f"Element: {key}, Frequency: {value}")
```

**OUTPUT-**

```
Duplicate elements and their frequency:
Element: 1, Frequency: 3
Element: 2, Frequency: 3
Element: 3, Frequency: 2
>>>|
```

**11) Write a Python program to print every alternate number of a given array.**

```
array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

alternate_numbers = array[::2]

print("Every alternate number in the array:")

print(alternate_numbers)
```

**OUTPUT-**

```
Every alternate number in the array:
[10, 30, 50, 70, 90]
>>>|
```

**12) Given are two one-dimensional arrays A & B, which are sorted in ascending order. Write a Python program to merge them into single sorted array C that contains every item from arrays A & B, in ascending order.**

```
A = [1, 3, 5, 7, 9]

B = [2, 4, 6, 8, 10]

C = sorted(A + B)

print("Merged sorted array C:")

print(C)
```

**OUTPUT-**

```
Merged sorted array C:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>>|
```

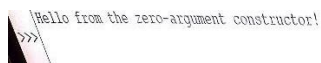
### 13) Write a Python program to show 0-arguments constructor.

```
class ExampleClass:
 def __init__(self):
 self.message = "Hello from the zero-argument constructor!"

 def display_message(self):
 print(self.message)

obj = ExampleClass()
obj.display_message()
```

#### OUTPUT-



```
>>> |Hello from the zero-argument constructor!
```

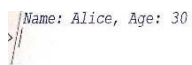
### 14) Write a Python program to show parameterized constructor.

```
class ExampleClass:
 def __init__(self, name, age):
 self.name = name
 self.age = age

 def display_info(self):
 print(f"Name: {self.name}, Age: {self.age}")

obj = ExampleClass("Alice", 30)
obj.display_info()
```

#### OUTPUT-



```
>>> |Name: Alice, Age: 30
```

### 15) Write a Python program to show constructor overloading.

```
class ExampleClass:
 def __init__(self, name=None, age=None):
 if name is not None and age is not None:
 self.name = name
 self.age = age
 self.message = f"Name: {self.name}, Age: {self.age}"
```

```

elif name is not None:
 self.name = name
 self.message = f"Name: {self.name}"
else:
 self.message = "No name or age provided."
def display_info(self):
 print(self.message)
obj1 = ExampleClass("Alice", 30)
obj2 = ExampleClass("Bob")
obj3 = ExampleClass()
obj1.display_info()
obj2.display_info()
obj3.display_info()

```

## OUTPUT-

```

Name: Alice, Age: 30
Name: Bob
No name or age provided.
>>

```

**16) Write a class, Grader, which has an instance variable, score, an appropriate constructor and appropriate methods. A method, letterGrade() that returns the letter grade as O/E/A/B/C/F. Now write a demo class to test the Grader class by reading a score from the user, using it to create a Grader object after validating that the value is not negative and is not greater than 100. Finally, call the letterGrade() method to get and print the grade.**

```

class Grader:
 def __init__(self, score):
 if 0 <= score <= 100:
 self.score = score
 else:
 raise ValueError("Score must be between 0 and 100.")
 def letterGrade(self):
 if 90 <= self.score <= 100:
 return 'O'
 elif 80 <= self.score < 90:

```

```


 return 'E'
 elif 70 <= self.score < 80:
 return 'A'
 elif 60 <= self.score < 70:
 return 'B'
 elif 50 <= self.score < 60:
 return 'C'
 else:
 return 'F'

def main():
 try:
 score = int(input("Enter the score (0-100): "))
 grader = Grader(score)
 grade = grader.letterGrade()
 print(f"The letter grade is: {grade}")
 except ValueError as e:
 print(e)

main()

```

## OUTPUT-



```

Enter the score (0-100): 87
The letter grade is: E
>>

```

**17) Write a class, Commission, which has an instance variable, sales; an appropriate constructor; and a method, commission() that returns the commission. Now write a demo class to test the Commission class by reading a sale from the user, using it to create a Commission object after validating that the value is not negative. Finally, call the commission() method to get and print the commission. If the sales are negative, your demo should print the message “Invalid Input”.**

```

class Commission:
 def __init__(self, sales):
 if sales >= 0:
 self.sales = sales

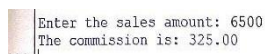
```

```

else:
 raise ValueError("Invalid Input: Sales cannot be negative.")
def commission(self):
 if self.sales >= 10000:
 return self.sales * 0.1 # 10% commission for sales >= 10000
 elif self.sales >= 5000:
 return self.sales * 0.05 # 5% commission for sales >= 5000
 else:
 return self.sales * 0.02 # 2% commission for sales < 5000
def main():
 try:
 sales = float(input("Enter the sales amount: "))
 commission_obj = Commission(sales)
 commission_amount = commission_obj.commission()
 print(f"The commission is: {commission_amount:.2f}")
 except ValueError as e:
 print(e)
main()

```

## OUTPUT-



```

Enter the sales amount: 6500
The commission is: 325.00

```

## 18) Write a Python program to implement the concept of inheritance.

```

class Vehicle:
 def __init__(self, brand, model):
 self.brand = brand
 self.model = model
 def display_info(self):
 print(f"Brand: {self.brand}, Model: {self.model}")
class Car(Vehicle):
 def __init__(self, brand, model, doors):
 super().__init__(brand, model)

```



```

 self.doors = doors

 def display_info(self):
 super().display_info()
 print(f"Doors: {self.doors}")

class Bike(Vehicle):
 def __init__(self, brand, model, type_of_bike):
 super().__init__(brand, model)
 self.type_of_bike = type_of_bike

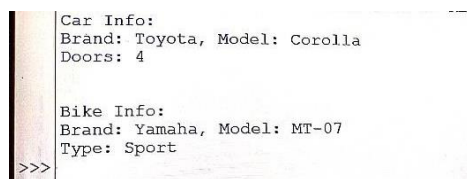
 def display_info(self):
 super().display_info()
 print(f"Type: {self.type_of_bike}")

def main():
 car = Car("Toyota", "Corolla", 4)
 print("Car Info:")
 car.display_info()
 print("\n")
 bike = Bike("Yamaha", "MT-07", "Sport")
 print("Bike Info:")
 bike.display_info()

main()

```

## OUTPUT-



```

Car Info:
Brand: Toyota, Model: Corolla
Doors: 4

Bike Info:
Brand: Yamaha, Model: MT-07
Type: Sport
>>>

```

## 19) Write a Python program to show method overloading.

```

class Calculator:
 def add(self, *args):
 if len(args) == 0:
 return 0
 elif len(args) == 1:

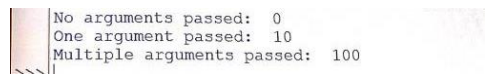
```

```
 return args[0]
 else:
 return sum(args)

def main():
 calc = Calculator()
 print("No arguments passed: ", calc.add())
 print("One argument passed: ", calc.add(10))
 print("Multiple arguments passed: ", calc.add(10, 20, 30, 40))

main()
```

## OUTPUT-



```

No arguments passed: 0
One argument passed: 10
Multiple arguments passed: 100
```