Practical 2: Basic Python Constructs

Liu Yuhao

202283710007

31808610

Department of Computer Science

Follow the instructions to complete each of these tasks.

This set of exercises focuses on writing basic Python code and exploring the use of several data constructs.

This work is not assessed but will help you gain practical experience for the coursework.

Strings 1

Use the `input()` function to **read** in a string from the user and store it in a variable.

Usage: https://www.w3schools.com/python/ref_func_input.asp

In [ ]:
```
# Read an input string
# The built-in input function will prompt for input and return the entered
# text.
x = input()
```

Print a modified version of this string, consisting of the input string's **first**, **middle**, and **last** characters.

For example:

- Input: `x = 'abcde'`
- Output: 'ace'

In [1]:
```
x = input()
mid = len(x) // 2
end = len(x)-1
out = x[0] + x[mid] + x[end]
print(out)
```

```
ace
```

Read **two input strings**, `s1` and `s2`. Then, create a new string, `s3`, by **inserting** `s2` in the middle of `s1`.

For example: `s1 = 'DATA'` and `s2 = 'comp'`.

- Output: 'DAcompTA'

In [10]:
```
# Request two input values
s1 = input()
s2 = input()
mid_s1 = len(s1) // 2 -1
s3 = s1[0:mid_s1]+s1[mid_s1] + s2 + s1[mid_s1+1:]
print(s3)
```

```
DAcompTA
```

Lists

Begin with a list of `numbers = [10, 25, 8, 42, 15]`.

Find the **minimum**, **maximum**, and **sum** of the numbers. Also, find the **mean** of the numbers.

Helpful resources:

- Built-in Functions: https://docs.python.org/3.8/library/functions.html
- List methods: https://www.w3schools.com/python/python_ref_list.asp

In [11]:
```
numbers = [10, 25, 8, 42, 15]
max_num = max(numbers)
min_num = min(numbers)
meam_num = sum(numbers) / len(numbers)
print("Max number:",max_num)
print("Min number:",min_num)
print("Mean number:",meam_num)
```

```
Max number: 42
Min number: 8
Mean number: 20.0
```

Begin with a list of `grades = ['A', 'B', 'X', 'C', 'B', 'A']`.

- Use a method to find the **index** number of the item 'X'.
- Then **replace** this item with 'A'.

In [12]:
```
grades = ['A', 'B', 'X', 'C', 'B', 'A']
index = grades.index('X')
grades[index] = 'A'
print(grades)
```

```
['A', 'B', 'A', 'C', 'B', 'A']
```

Begin with list of `grades = ['A', 'B', 'A', 'C', 'B', 'A']`.

Count the numbers of **each** grade.

In [1]:
```
grades = ['A', 'B', 'A', 'C', 'B', 'A']
grades_counts = {}
for grade in grades:
    if grade in grades_counts:
        grades_counts[grade] += 1
    else:
        grades_counts[grade] = 1

print(grades_counts)
```
```
{'A': 3, 'B': 2, 'C': 1}
```

Create **two lists**: one called `autumn` and one called `spring`.

Populate them with the modules you have scheduled in these semesters. Then, **join/concatenate** the lists together and **print** the result.

In [3]:
```
# Create 2 lists of strings
autumn = ["Java","database","DIP"]
spring = ["Python","Software system design","Operating System"]
print(autumn + spring)
```
```
['Java', 'database', 'DIP', 'Python', 'Software system design', 'Operating System']
```

Tuples

Define a tuple with these elements:

- 1
- -1
- 'two'
- 3.14
- [4, 5, 6]
- 'abc'

Change the entry `'abc'` to `{'key': 66}`. **Print** the resulting tuple.

In [ ]:
```
# Define tuple
t = (1,-1,'two',3.14,[4,5,6],'abc')
t = list(t)
index = t.index('abc')
t[index] = {'key':66}
t = tuple(t)
print(t)
print(type(t))
```
```
(1, -1, 'two', 3.14, [4, 5, 6], {'key': 66})
<class 'tuple'>
```

Dictionaries

Begin with a **dictionary** of `person:age` pairs:

```
age = {'Felix':72, 'Ben':40, 'Chris':35,
       'David':60, 'Eva':10, 'Amy':55}
```

Inspect and analyse the dictionary to find the person who is the **youngest**.

In [10]:
```
age = {'Felix': 72, 'Ben': 40, 'Chris': 35,
       'David': 60, 'Eva': 10, 'Amy': 55}
print(min(age,key=age.get))
```

```
Eva
```

Sets

Begin with lists of students belonging to 2 groups:

```
group1 = ['Amy', 'Chris', 'Eva', 'David']
group2 = ['Ben', 'Chris', 'Felix', 'David']
```

1. Considering entries in both sets, **print** a list of **unique** student names, **sorted** in **ascending** alphabetical order.
   - Hint: https://www.w3schools.com/python/ref_func_sorted.asp

1. **Print** the names of those students who are in **both** groups.

In [13]:
```
group1 = ['Amy', 'Chris', 'Eva', 'David']
group2 = ['Ben', 'Chris', 'Felix', 'David']
set1 = set(group1)
set2 = set(group2)
print(sorted(set1))
print(sorted(set2))

a = {1,3,7,11}
b = {3,5,7,9}
print(f'Union: {a | b}')
print(f'Intersection: {a & b}')
print(f'Difference: {a - b}')
print(f'Difference: {b - a}')
```

```
['Amy', 'Chris', 'David', 'Eva']
['Ben', 'Chris', 'David', 'Felix']
Union: {1, 3, 5, 7, 9, 11}
Intersection: {3, 7}
Difference: {1, 11}
Difference: {9, 5}
```

Begin with a list of:

```
user_feelings = ['angry', 'angry', 'neutral', 'happy',
                 'neutral', 'excited', 'happy', 'neutral',
                 'happy', 'happy'].
```

Print the **unique** feeling categories.

Expected output: `{'angry', 'excited', 'happy', 'neutral'}`

In [15]:
```
user_feelings = ['angry', 'angry', 'neutral', 'happy',
                 'neutral', 'excited', 'happy', 'neutral',
                 'happy', 'happy']
print(sorted(set(user_feelings)))
```

```
['angry', 'excited', 'happy', 'neutral']
```

Strings 2

Python has a set of built-in methods that you can use on Strings.

- https://www.w3schools.com/python/python_strings_methods.asp

Note: All string methods return **new values**. They do not change the original string.

Using string methods, **convert** the string:

`str1 = '    CS2PP:Programming in Python     '`

to

'cs2pp:programming_in_python'.

In [20]:
```
str1 = '    CS2PP:Programming in Python     '
str1 = str1.strip()
str1 = str1.lower()
#str1 = ''.join([char for char in str1 if not char.isdigit()])


str1 = str1.replace(' ', '_')
print(str1)
```

```
cs2pp:programming_in_python
```

In the film 'Supertroopers', Vermont state troopers, Mac and Foster, play a the '*Cat Game*' to see how many times they can say "*meow*" when speaking to someone about a driving infraction.

The dialog from the script is provided as a **docstring** below.

Their co-worker previously made it to 6. Using string methods, **determine** whether Foster works in 10 mentions of "meow".

In [11]:
```
dialog = '''
[Mac laughs - they walk up to the car, and Foster taps on the driver side]

Larry Johnson : Sorry about the...
```

```
Foster : All right meow. (1) Hand over your license and registration.

[the man hands him his license]

Foster : Your registration? Hurry up meow. (2)

[Mac ticks off two fingers]

Larry Johnson : Sorry.

[the man laughs a little]

Foster : Is there something funny here boy?

Larry Johnson : Oh, no.

Foster : Then why you laughing, Mister... Larry Johnson?

[pause]

Foster : All right meow, (3) where were we?

Larry Johnson : Excuse me, are you saying meow?

Foster : Am I saying m-e-o-w? [edit: this one doesn't count!]

[Mac puts his hands up for the fourth one, but makes an "eehhh" facial expression, a

Larry Johnson : I thought...

Foster : Don't think boy. Meow, (4) do you know how fast you were going?

[man laughs]

Foster : Meow. (5) What is so funny?

Larry Johnson : I could have sworn you said meow.

Foster : Do I look like a cat to you, boy? Am I jumpin' around all nimbly bimbly fro

[Mac is gut-busting laughing]

Foster : Am I drinking milk from a saucer?

[feigned anger]

Foster : Do you see me eating mice?

Foster : [Mac and the man are laughing their heads off now] You stop laughing right

Larry Johnson : [the man stops and swallows hard] Yes sir.

Foster : Meow, (7) I'm gonna have to give you a ticket on this one. No buts meow. (8

[rips off the ticket and hands it to the man]

Foster : Not so funny meow, (9) is it?

Foster : [Foster gets up to leave, but Mac shakes his hands at him, indicating only
'''
```

In [ ]:
```python
meow_count = 0
```

```
for line in dialog.split("\n"):
    if line.startswith("Foster :"):  # 只统计 Foster 说的话
        meow_count += line.lower().count("meow")



if meow_count == 10:
    print("Yes, Foster successfully said 'meow' 10 times!")
else:
    print(f"No, Foster only said 'meow' {meow_count} times.")
```

```
Yes, Foster successfully said 'meow' 10 times!
```

Conditional Statements

The following example was shown in the lecture:

```
grade = 55

if grade < 40:
    print('Fail')
elif grade < 50:
    print('Threshold Pass')
else:
    print('Third Class or above')
```

Add more `elif` blocks to **print**:

- 50-59: Third Class,
- 60-69: Second Class
- greater than or equal to 70: First Class

Test your solution with an input grade of 70.

In [2]:
```
grade = 70
if grade < 40:
    print('Fail')
elif grade < 50:
    print('Threshold Pass')
elif grade < 59:
    print('Third Class or above')
elif grade < 69:
    print('Second Class')
else:
    print('First Class')
```

```
First Class
```

The term "*Daffodil number*" indicates a three-digit number whose sum of the cubes of its digits is equal to the number itself.

Write code to ask the user to **input** a three-digit positive integer `n`.

If `n` is a “Daffodil number”, **print** "YES", otherwise, **print** "NO".

- e.g.: 153, 370, 371, or 407 will print "YES"

- e.g.: 123, 345, 567, or 890 will print "NO"

In [3]:
```
# Request input
n = int(input())
if n == (n // 100)**3 + ((n // 10) % 10)**3 + (n % 10)**3:
    print('Yes')
else:
    print('No')
```
```
Yes
```

Ask the user to **input** the 3-letter abbreviated name of a calendar month. Then, **output** the number of days in that month.

Write an `if` statement to output the number of days in the input month.

- e.g. input = Jan, output = 31 days
- e.g. input = Feb, output = 28/29 days
- e.g. input = Apr, output = 30 days
- e.g. invalid input, output = Invalid month name

Test your solution with valid and invalid input to ensure that it is working as expected.

In [4]:
```
print(
    "List of months: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec"
)
thirty_one = ('Jan','Mar','May','Jul','Aug','Oct','Dec')
thirty = ('Apr','Jun','Sep','Nov')
month_name = input("Input the name of Month: ")
if month_name == 'Feb':
    print('28/29 days')
elif month_name in thirty_one:
    print('31 days')
elif month_name in thirty:
    print('30 days')
else:
    print('Invalid month name')
```
```
List of months: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
31 days
```

In [ ]:
```
```

Loops

Begin with a list of `marks = [35, 58, 67, 78, 20, 85]`. A passing mark is **40**.

Count and **output** the number of passing and failing marks.

In [9]:
```
marks = [35, 58, 67, 78, 20, 85]
passing_mark = 40
pass_num = 0
pass_arr = []
fail_num = 0
fail_arr = []
```

```
for mark in marks:
    if mark > passing_mark:
        pass_num += 1
        pass_arr.append(mark)
    else:
        fail_num += 1
        fail_arr.append(mark)

print(f'{pass_num} pass {pass_arr}')
print(f'{fail_num} fail {fail_arr}')
```

```
4 pass [58, 67, 78, 85]
2 fail [35, 20]
```

Read an **input** string. Then, **count** and **output** occurrences of **all unique characters** within the input string.

- Example input: data science
- Expected output:

  ```
  {'d': 1, 'a': 2, 't': 1, ' ': 1, 's': 1, 'c': 2, 'i': 1, 'e': 2,
  'n': 1}
  ```

Hints:

- Use a dictionary.
- `aString.count(c)` counts the number of occurrences of `c` in `aString`.

In [10]:
```
strInput = input()  # read input
dic = {}
for i in strInput:
    dic[i] = strInput.count(i)

print(dic)
```

```
{'d': 1, 'a': 2, 't': 1, ' ': 1, 's': 1, 'c': 2, 'i': 1, 'e': 2, 'n': 1}
```

The Collatz conjecture is a conjecture sequence begins with an integer value, operates on that value iteratively, and always reaches 1.

The sequence is determined by the following procedure:

- start with a **positive integer** number n;
- the next number in the sequence is $n/2$, if n is **even**
- and $3n + 1$, if n is **odd**.

Write code to **read** in a positive integer number n and **print** each step in the Collatz conjecture sequence.

- e.g.:
  - Input: 5
  - Expected Output: 5, 16, 8, 4, 2, 1
- e.g.:
  - Input: 12
  - Expected Output: 12, 6, 3, 10, 5, 16, 8, 4, 2, 1

In [13]:
```
# Request input
n = int(input('Input a positive integer : '))
print(n)
while n > 1:
    if n % 2 == 0:
        n = n / 2
    else:
        n = 3 * n + 1
    print(int(n))
```

```
5
16
8
4
2
1
```

Begin with this list of numbers:

```
numbers = [12, 6, 3, 10, 5, 16, 8, 4, 2, 1]
```

Use a combination of a `for` loop and **conditional statements** to find and **print** the maximum value in the list.

In [5]:
```
numbers = [12, 6, 3, 10, 5, 16, 8, 4, 2, 1]
max_value = numbers[0]

for number in numbers:
    if number > max_value:
        max_value = number

# Print the maximum value
print("The maximum value is:", max_value)
```

```
The maximum value is: 16
```

---

match/case Statements

You have a **list of shipments**, where each shipment is itself a list. Some shipments contain:

- Two items: a product name and quantity, e.g., `["Gadgets", 100]`
- Three items: a product name, quantity, and a special shipment flag, e.g., `["Widgets", 50, "URGENT"]`

Other patterns (including different lengths) which you will handle as the default catch-all case.

Loop through the shipments and use a match statement to:

1. Print a custom message for the 2-item pattern.
2. Print a different custom message for the 3-item pattern that includes the special flag.
3. Handle any other list patterns with a default message.

Use a `match` statement to handle each pattern differently. Include relevant output like "Processing 2-item shipment..." or "Found special flag...".

What assummptions have we made?

Sample shipments to try:

In [7]:
```
shipments = [
    ["Gadgets", 100],
    ["Widgets", 50, "URGENT"],
    [],
    ["Widgets", 25, "SAFE"],  # Another 3-element pattern
    ["Gizmo", 10],
    [10, "Gizmo"]
]

for shipment in shipments:
    match shipment:
        # Handle 2-item shipments (product: str, quantity: int)
        case [str(product), int(quantity)]:
            print(f"Processing 2-item shipment: {quantity} units of {product}.")

        # Handle 3-item shipments with a special flag
        case [str(product), int(quantity), str(flag)]:
            print(f"Found special flag '{flag}' for shipment: {quantity} units of {p

        # Handle all other patterns
        case _:
            print("Unknown shipment format. Skipping this shipment.")
```

```
Processing 2-item shipment: 100 units of Gadgets.
Found special flag 'URGENT' for shipment: 50 units of Widgets.
Unknown shipment format. Skipping this shipment.
Found special flag 'SAFE' for shipment: 25 units of Widgets.
Processing 2-item shipment: 10 units of Gizmo.
Unknown shipment format. Skipping this shipment.
```

In [ ]:

You have a list of **user access records**, where each record is a **dictionary**. Some sample records could look like:

```
user_records = [
    {"username": "xy993241", "role": "admin", "last_login": "2024-03-
01"},
    {"username": "hu183337", "role": "guest"},
    {"username": "oo262911", "last_login": "2024-02-28"},  # Missing
"role" key
    {"username": "ne338010", "role": "moderator", "last_login": "2024-
01-15"}
]
```

Using `match`:

1. If the record has `{"role": "admin"}` and a `last_login`, print a message praising the admin.
2. If the record has a `"role"` key that is not `"admin"`, print a short greeting referencing that role.

3. If the record has no `"role"` key, print a message indicating an unknown role.

Implement the logic with a match statement against each dictionary, demonstrating dictionary key matching.

In [8]:
```
user_records = [
    {"username": "xy993241", "role": "admin", "last_login": "2024-03-01"},
    {"username": "hu183337", "role": "guest"},
    {"username": "oo262911", "last_login": "2024-02-28"},
    {"username": "ne338010", "role": "moderator", "last_login": "2024-01-15"}
]

for record in user_records:
    match record:
        case {"role": "admin", "last_login": _}:
            print(f"Admin {record['username']} recently logged in. Thank you for you
        case {"role": role} if role != "admin":
            print(f"Welcome, {record['username']} with {role} role.")
        case _ if "role" not in record:
            print(f"User {record['username']} has an unknown role. Please update pro
```

```
Admin xy993241 recently logged in. Thank you for your dedication!
Welcome, hu183337 with guest role.
User oo262911 has an unknown role. Please update profile.
Welcome, ne338010 with moderator role.
```

In [ ]:

Comprehensions

List Comprehension

Write a **list comprehension** that returns pairs `(x, y)` where `x` is divisible by `y` for `x` in the range 1 to 22 (exclusive upper bound) and y in the range 1 to 11 (inclusive upper bound).

Hint: there should be 60 element pairs in the resulting list.

In [ ]:

Dictionary Comprehension 1

Two dictionaries have been created below. `dict2` is a **modified** version of `dict1` where the values have been changed for certain keys.

Use **dictionary comprehension** to create a dictionary containing entries which identify each key that had its value modified. The value in each entry of this resulting dictionary should show the **tuple** pair of (original value, modified value).

Assume each dictionary has **exactly the same set of keys**.

You should end up with:

```
{'a': (1, 12),
 'b': (['fish', 'tiger', 'octopus'], ['fish', 'tigger', 'octopus']),
 'c': ((50, 50), (50, 60))}
```

In [9]:
```
dict1 = dict(a=1,
             b=['fish', 'tiger', 'octopus'],
             c=(50, 50),
             d=2,
             e='total',
             f=55,
             g={'chrome': 'tab'},
             h={4, 5, 6})

dict2 = dict(c=(50, 60),
             a=12,
             g={'chrome': 'tab'},
             f=55,
             b=['fish', 'tigger', 'octopus'],
             d=2,
             e='total',
             h={4, 5, 6})

result = {key: (dict1[key], dict2[key]) for key in dict1 if dict1[key] != dict2[key

print(result)
```

```
{'a': (1, 12), 'b': (['fish', 'tiger', 'octopus'], ['fish', 'tigger', 'octopus']),
'c': ((50, 50), (50, 60))}
```

In [ ]:

Dictionary Comprehension 2

This one is a bit **tricky**.

Beginning with the two lists of integers below, use dictionary comprehension to create a **dictionary** where

1. the **keys** are the unique values in `data1` **in increasing order**
2. and the **values** are **lists** of the corresponding values in `data2`. That is, the objects in the lists should be the entries from `data2` that are presently found at the same indices as unique values in `data1`.

```
data1 = [12, 82, 35, 82, 12, 35]
data2 = [24, 56, 66, 74, 28, 28]
```

Hint: This can be solved using **list comprehension** within the **dictionary comprehension** to form the **values** of the resulting dictionary. You might also use the `enumerate()` function described in L02 or a combination of `range()` and `len()` to assist you.

You should end up with:

```
{12: [24, 28], 35: [66, 28], 82: [56, 74]}
```

In [10]:
```
data1 = [12, 82, 35, 82, 12, 35]
data2 = [24, 56, 66, 74, 28, 28]
result = {
    k: [data2[i] for i, num in enumerate(data1) if num == k]
    for k in sorted(set(data1))
}

print(result)
```

```
{12: [24, 28], 35: [66, 28], 82: [56, 74]}
```