weniv. world

Teacher's Edition

Weniv World Adventure

Game-based Python Learning Platform

world-en.weniv.co.kr | **WENIV Corp.**

Weniv World Adventure

Game-based Python Learning Platform

- Teacher's Edition -

weniv.world

Resources

Usage Scope

- You can access the resource through the Notion link below (Please note that the short link may not work depending on the service status).
- The PDF can be printed for use as a textbook without permission.
- The Korean version of this e-book is available for free download on Ridibooks, Kyobo Bookstore, YES24, Aladdin Bookstore, and Millie's Library.

- Notion Link : https://www.notion.so/paullabworkspace/7e691173fea444038981905d35b827c9?pvs=4
- Short Link:

  https://url.kr/927tlq
- QR Code:

- PDF:

  `You can download it from the notion page.`

Contents

01

Introduction

About Us

Hello,
This is Weniv Inc.

We specialize in creating ICT educational content and conducting online and offline software (SW) courses. Our online courses operate under the name **Jeju Coding Basecamp**, derived from our company's location in Jeju, South Korea.

Weniv Inc. aims to be a stepping stone, helping individuals navigate the ICT career paths, professions, further education, and employment. We are committed to addressing challenges faced by regional communities and young individuals, ensuring that everyone has access to ICT education without discrimination based on location or resources. Through service development, ICT education, and community activities, we collaborate with local communities and youth to overcome these challenges.

With experience in various programs, including ‘Jeju Coding Basecamp,’ corporate training for new hires, tertiary education, and K-12 education, we strive to make a positive impact in the field of ICT.

Preface

Weniv World is an educational software designed to make Python programming more accessible. Users can learn Python and develop computational thinking skills by solving missions that involve controlling the protagonist, Licat, with simple commands.

For students, the installation process and login can be obstacles. To facilitate learning without such difficulties, the platform provides a web service that can be accessed from anywhere without the need for a separate login.

You can download both student and teacher manuals in PDF format from the website below. We hope you enjoy an exciting Python learning experience with Weniv World.

Weniv World Beta

Python Coding Expedition to Weniv World

http://world-en.weniv.co.kr/

Weni World Beta

Weniv Notebook

Run Python in Weniv Notebook(code editor)

https://notebook.weniv.co.kr/

Weniv Notebook

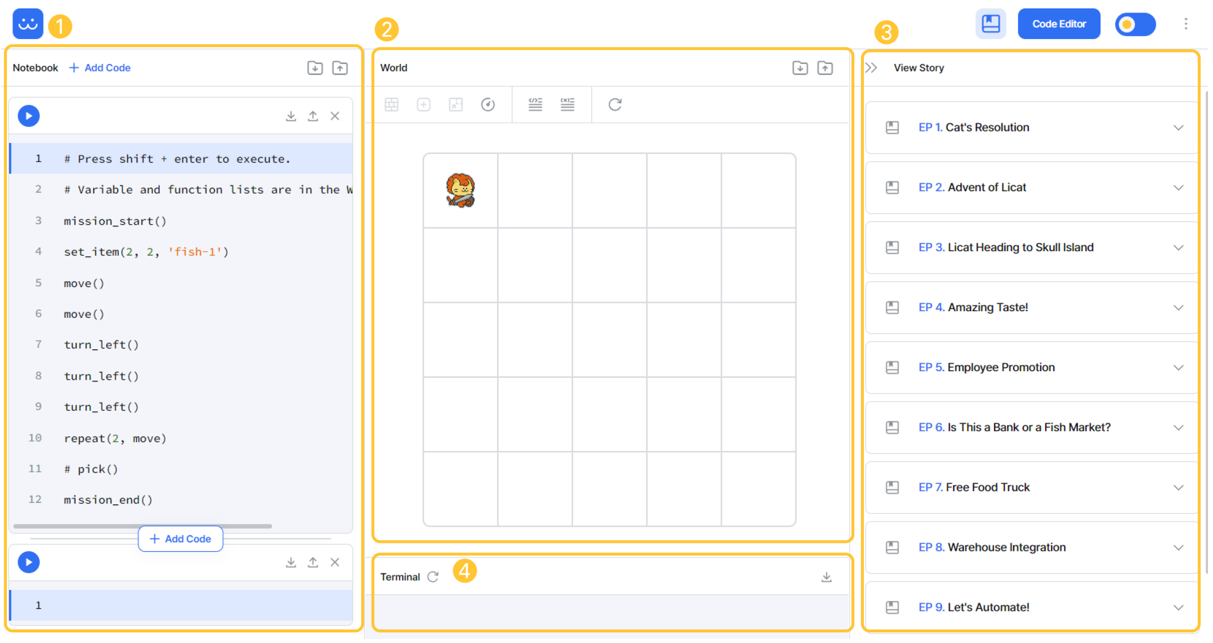
About Platform

1. Weniv World

Weniv World Beta

Python Coding Expedition to Weniv World

http://world-en.weniv.co.kr/



```
# Press shift + enter to execute.
# Variable and function lists are in the W
mission_start()
set_item(2, 2, 'fish-1')
move()
move()
turn_left()
turn_left()
turn_left()
repeat(2, move)
# pick()
mission_end()
```

EP 1. Cat's Resolution
EP 2. Advent of Licat
EP 3. Licat Heading to Skull Island
EP 4. Amazing Taste!
EP 5. Employee Promotion
EP 6. Is This a Bank or a Fish Market?
EP 7. Free Food Truck
EP 8. Warehouse Integration
EP 9. Let's Automate!

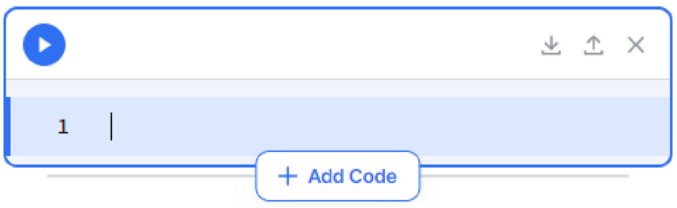
The platform is divided into four sections.

1. **Notebook**: You can write code in this section.
2. **World**: Executes code to move Licat.
3. **Story**: Provides story-based missions.
4. **Terminal**: Outputs results of the code.

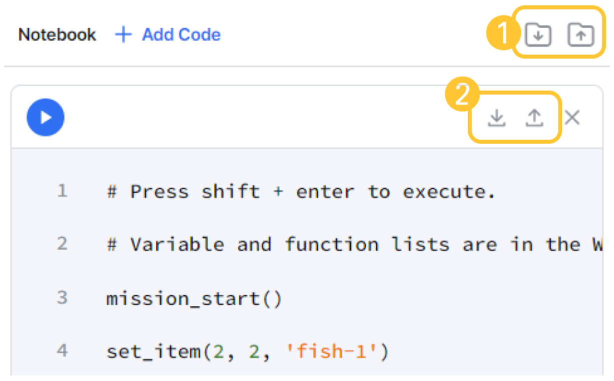
1.1. Notebook

This is a space where you can write and execute code. You can run the code by pressing the `Run` button at the top left corner of the code block. On Windows, you can use the `Shift` + `Enter` or `Alt` + `Enter` shortcut to execute the code, while on macOS, you can use the `Shift` + `Enter` or `Option` + `Enter` shortcut.

The code can be written in multiple cells. You can add a code cell using the `Add Code` button at the top of the notebook area or the `Add Code` button that appears at the bottom of each code cell when you hover over it. To delete a code cell, you can click the `x` button on the right side of each code cell. Please be aware that deleted cells cannot be recovered.



You can download the written code as a file. Clicking the `Download Notebook` button (❶), you can download the entire code as a `.ipynb` file extension. The downloaded file can be executed in Jupyter Notebook or Google Colab. You can also import the `.ipynb` file and use it in Weniv World with `Upload Notebook` button. Code can be downloaded and uploaded on a cell-by-cell basis, and you can click the button on the right side of each code cell (❷) to download it as a `.py` file extension.



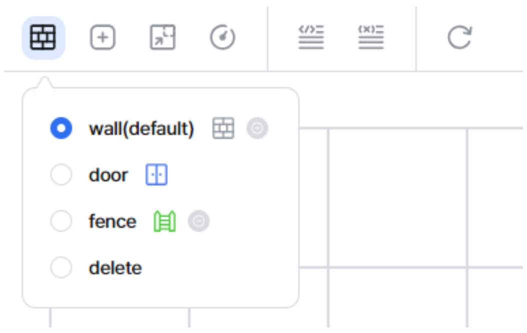
```
# Press shift + enter to execute.
# Variable and function lists are in the W
mission_start()
set_item(2, 2, 'fish-1')
```

1.2. World

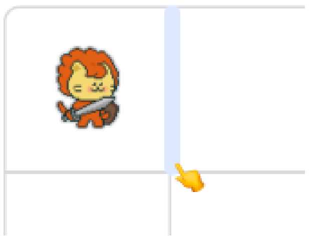
The World section is a space that contains information about the world, including walls, items, world size, character details, and so on. You can see the results of the executed code in the World section.

1.2.1 Walls

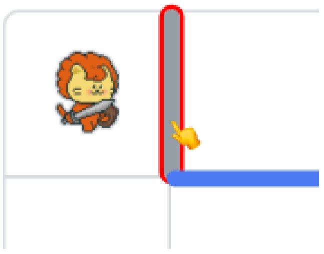
Walls are obstacles that should be detoured. You cannot pass any type of wall, including the default wall, door, and fence. Doors can be removed using the `open_door()` command. The fence is functionally identical to the wall, but in different colors.



To add a wall, choose the type of wall to add, and when you hover the mouse over the world, the positions where walls can be added are displayed like below. Click on the desired spot to build the wall.

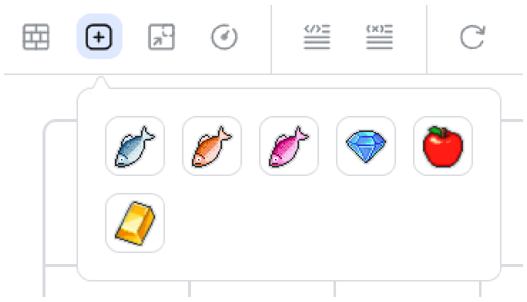


Select "delete" in the menu, hover over the wall to be deleted (highlighted in red), and click to remove it.

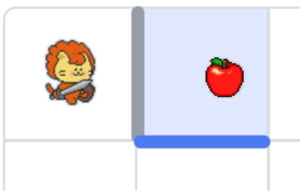


1.2.2 Items

Items are used in various missions and come in six types: `fish-1`, `fish-2`, `fish-3`, `diamond`, `apple`, `goldbar`.



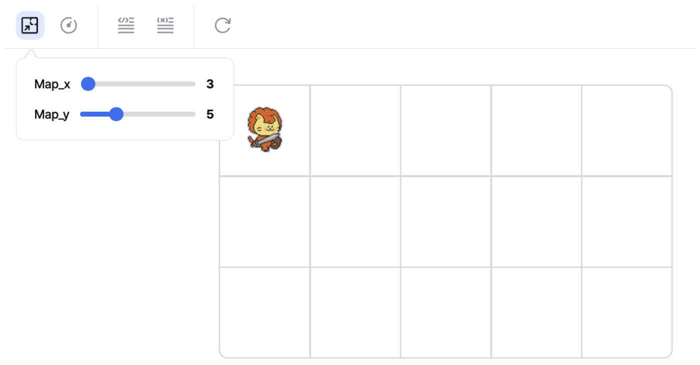
To add items, use the `set_item()` command or select an item and click on the world. Item information in the world is stored in the `item_data` variable.



When there is an item at the character's location, you can pick it up with the `pick()` command. You can also use the `put(item-name)` command to place an item you are carrying under the character's feet. The character's list of items can be checked using the `item()` command.

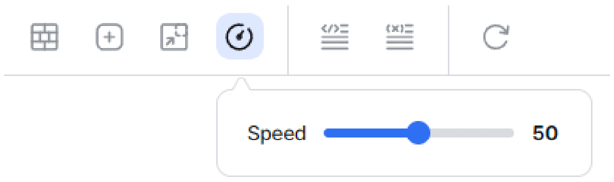
1.2.3 Size

You can adjust the size of the world. The x-axis represents rows, and the y-axis represents columns.


Map_x 3
Map_y 5

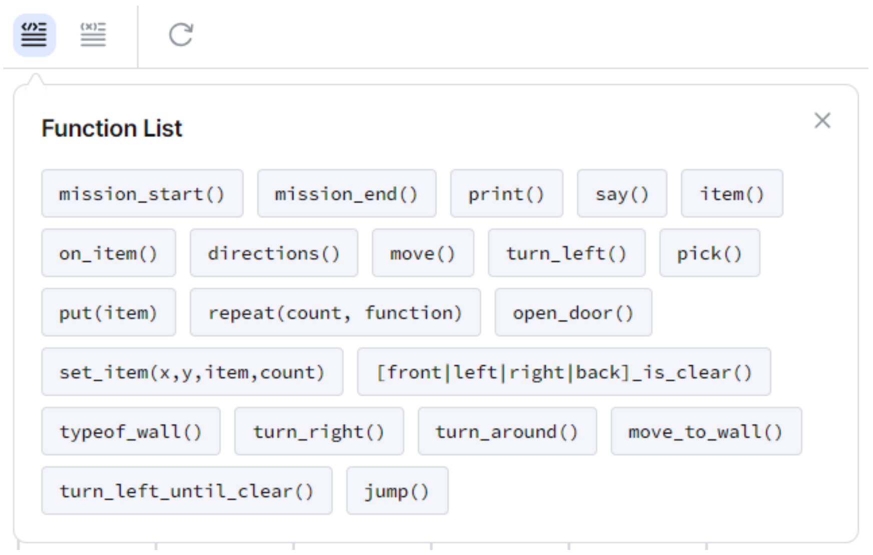
1.2.4 Speed

It indicates the speed of the character. A higher value executes the code at a faster speed.


Speed 50

1.2.5 Functions and Variables Lists

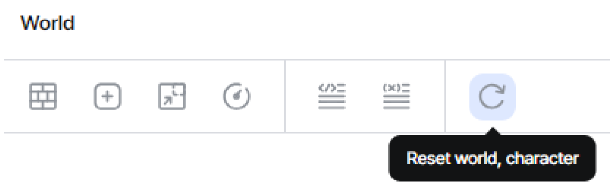
You can see the description of each function and variable when you mouse over each item. It also provides explanations about functions that require specific modules. Clicking on each item copies the code to the clipboard for use.


Function List

`mission_start()` `mission_end()` `print()` `say()` `item()`
`on_item()` `directions()` `move()` `turn_left()` `pick()`
`put(item)` `repeat(count, function)` `open_door()`
`set_item(x,y,item,count)` `[front|left|right|back]_is_clear()`
`typeof_wall()` `turn_right()` `turn_around()` `move_to_wall()`
`turn_left_until_clear()` `jump()`

1.2.6 World Initialization

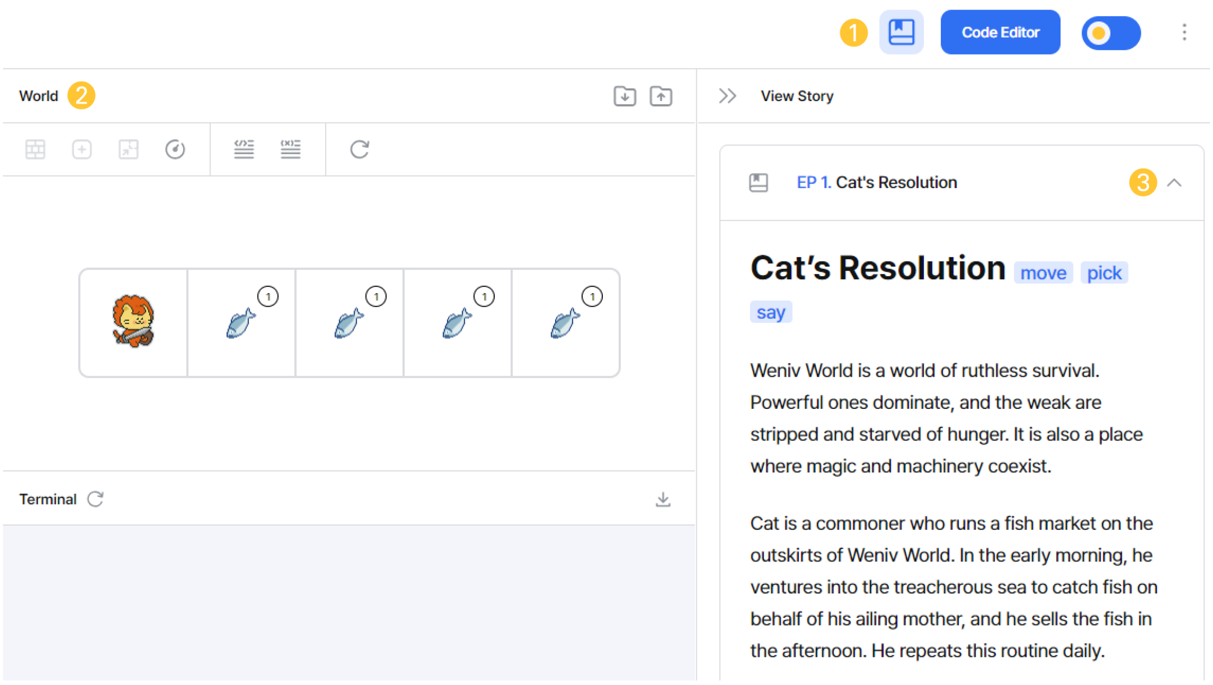
This button initializes the information in the world. You can reset wall information, items, size, and character information. Please note that the initialized world information cannot be restored, so if you need to keep it, it is recommended to download it before initializing.



1.3. Story

You can activate Story Mode by clicking the `Story` button (❶). When Story Mode is active, editing in the world (adding walls or items, changing size, etc) becomes disabled.

Clicking on each story's right button (❸) reveals `Story`, `Mission`, and `Hints`. The hints are more extensive than the actual code used in the mission.



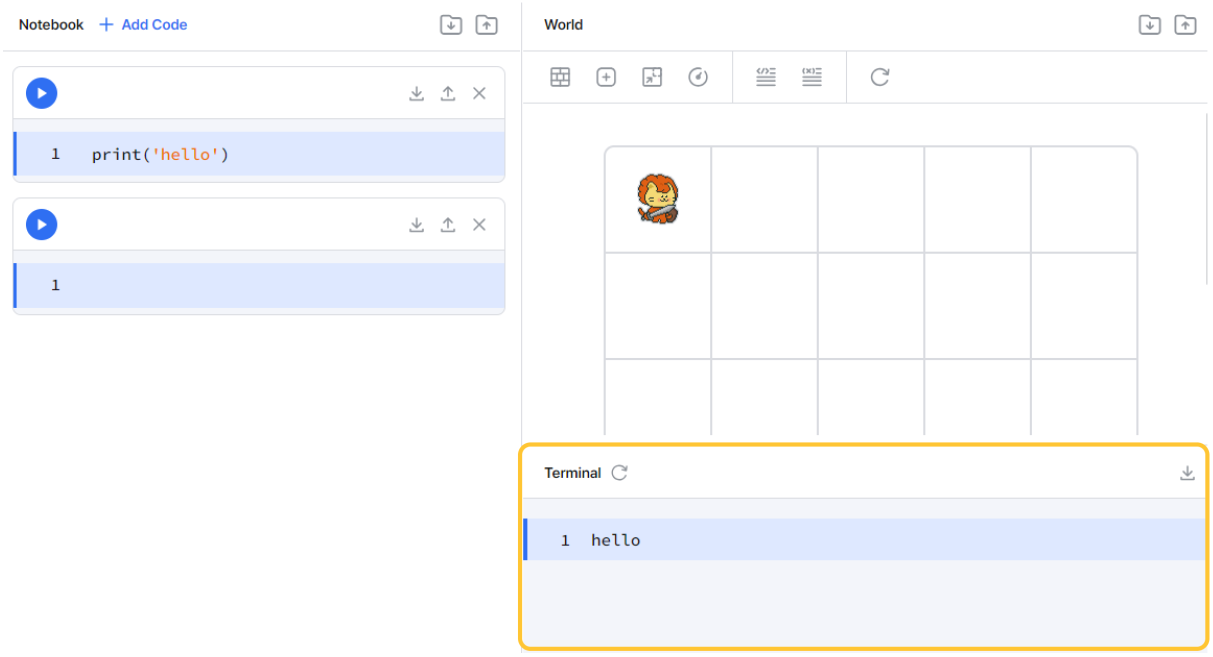
Cat's Resolution move pick say

Weniv World is a world of ruthless survival. Powerful ones dominate, and the weak are stripped and starved of hunger. It is also a place where magic and machinery coexist.

Cat is a commoner who runs a fish market on the outskirts of Weniv World. In the early morning, he ventures into the treacherous sea to catch fish on behalf of his ailing mother, and he sells the fish in the afternoon. He repeats this routine daily.

1.4. Terminal

This is an area where you can check the output results of the code. You can see results or error messages using the `print()` function. The output displayed in the terminal can be downloaded or cleared. If you click the `init terminal` button, all text will disappear.

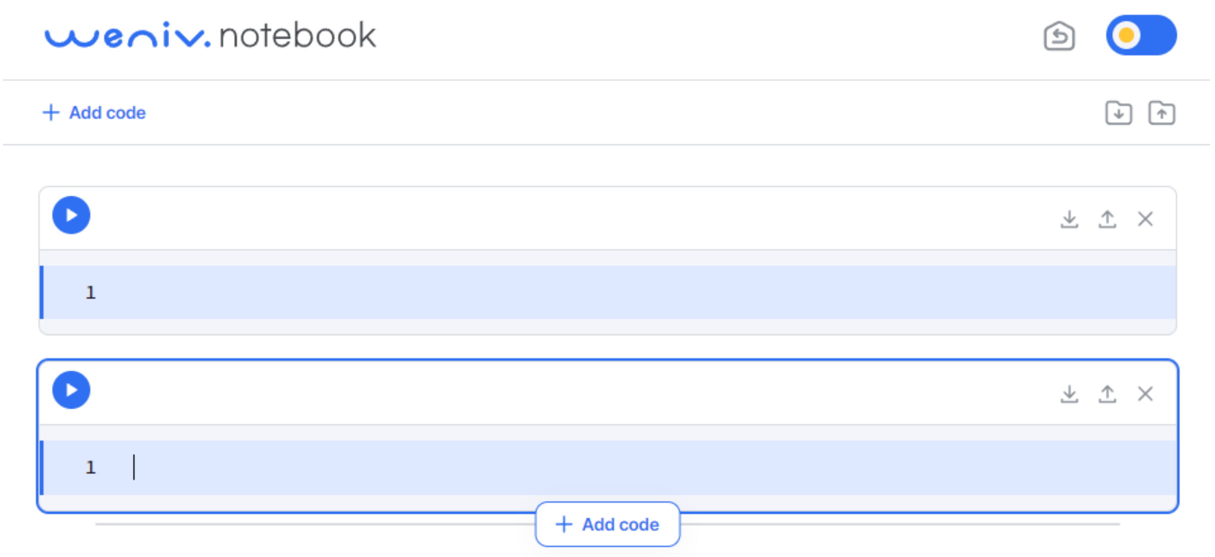


```
print('hello')
```

```
1  hello
```

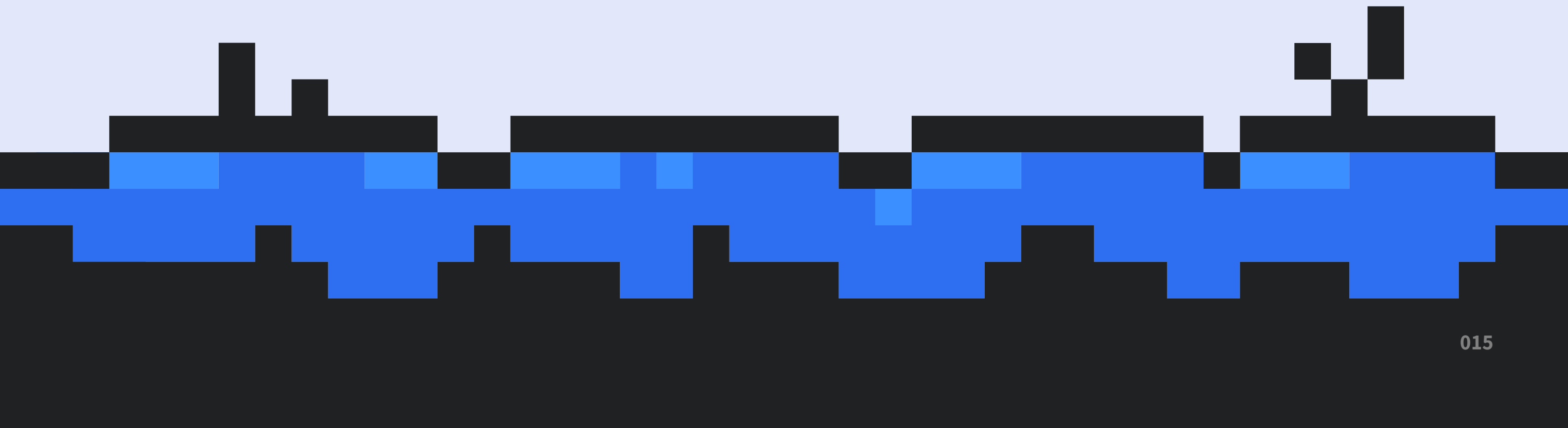
2. Weniv Notebook

Weniv Notebook provides a web environment where you can execute Python without logging in. You can run it by clicking the `Run` button on the left or using the shortcut `Shift` + `Enter` or `Alt` + `Enter`. The usage is the same as in Weniv World.



02

Weniv World Adventure



Cat’s Resolution

1. Chapter Objectives

`move` : You can move the character one space using the `move()` function.

`pick` : You can pick up an item under the character using the `pick()` function.

`say` : The character can speak using the `say()` function.

2. Story

Weniv World is a world of ruthless survival. Powerful ones dominate, and the weak are stripped and starved of hunger. It is also a place where magic and machinery coexist.

Cat is a commoner who runs a fish market on the outskirts of Weniv World. In the early morning, he ventures into the treacherous sea to catch fish on behalf of his ailing mother, and he sells the fish in the afternoon. He is an extremely ordinary boy who repeats this life every day.

> “Please take care of this fish market, my son…”

His mothers' illnesses worsened. In Weniv World, however, the hospital is a place only nobles can access. It’s a brutal place for commoners where even death goes unnoticed.

> "What I need to do best now is catch and sell fish, meow!"

Cat decided to do what he could. He got up every day no matter how hard it was. He tried to keep his mind not to show his weakness.

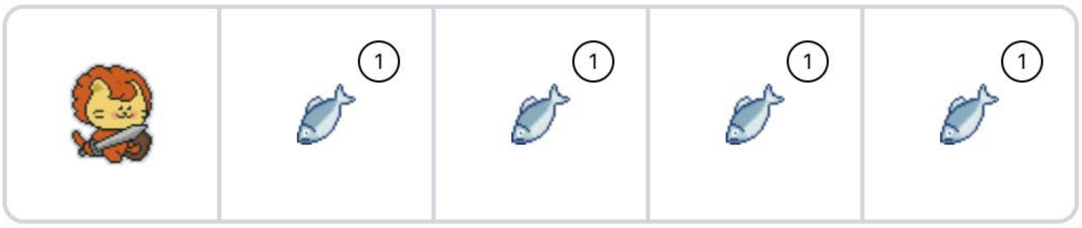
> "I'll build a hospital that even commoners can come, meow!"

Cat got healthy food and medicine with the money he earned every day to take care of his sick mother. He also trained himself to catch more fish and explored difficult skills day and night.

As time went on, his skills improved day by day. No one in Weniv World could catch as many fish as Cat.

2.1 Mission

Catch all the fish and return to the first spot, then say "hello, world!" Use the `say` function instead of the `print` function.



2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
move()
turn_left()
repeat(2, move)
pick()
say('hello world!')
```

3. Solution

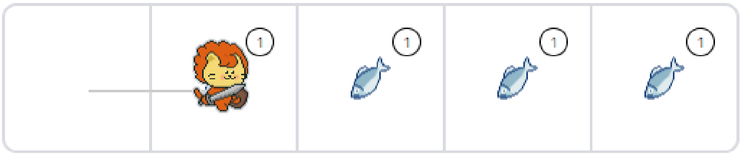
If there is a function using the world (where the character moves) in the code cell to be executed, the code must be used as shown below. If the following code is not included, errors such as delayed output of print or delayed movement of the character may occur.

```
mission_start()
# Code using the world
mission_end()
```

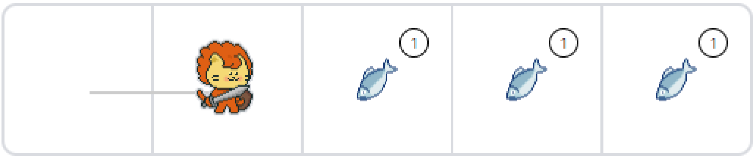
First, let's move one space as follows.

```
mission_start()
move()
mission_end()
```

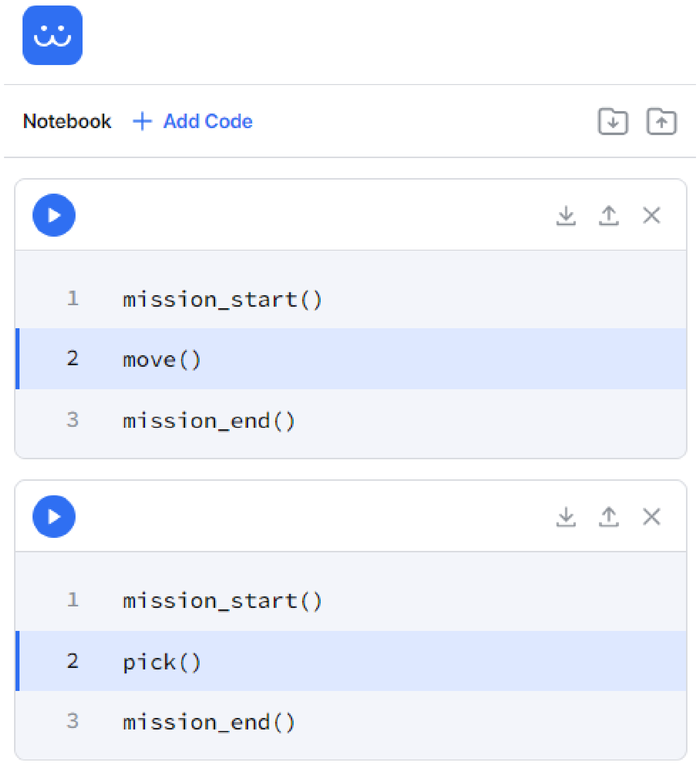
Write the above code in the notebook and execute it by clicking the `Run` button or pressing `Shift` + `Enter` or `Alt` + `Enter` shortcut.



You can see that Cat has moved one space as shown above. Now, let's try picking up the item below. Add another code cell and execute the `pick()` function.



You can see that one fish has been picked up as shown above. The notebook should look like this:



```
mission_start()
move()
mission_end()
```

```
mission_start()
pick()
mission_end()
```

Now, press the reset button to return to the initial state of the story. Go back to the first notebook and enter the following code to move forward and pick up all the fish.

```
mission_start()
move()
pick()
move()
pick()
move()
pick()
move()
pick()
mission_end()
```

Now you need to return to the starting point. Using `turn_left()`, you will rotate 90 degrees to the left. If you execute `turn_left()` twice, you will face the back. Add a cell and enter the following code:

```
mission_start()
turn_left() # Rotate 90 degrees to the left
turn_left()
mission_end()
```

Now, you just need to move forward four times. Enter the following in the next cell.

```
mission_start()
move()
move()
move()
move()
mission_end()
```

Now you need to say "hello, world!" Enter the following code. `say()` and `print()` do not require `mission_start()` and `mission_end()`. `say()` outputs text in a speech bubble of a character, while `print()` outputs to the terminal.

```
say("hello, world!")
```

Using `repeat()` can provide a more efficient solution. It can be used in the form of `repeat(number of repetitions, function name)`. For example, the last code where `move()` is used four times can be easily shortened as follows:

```
mission_start()
repeat(4, move)
```

```
mission_end()
```

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()
move()
pick()
move()
pick()
move()
pick()
move()
pick()
repeat(2, turn_left)
repeat(4, move)
say('hello, world!')
mission_end()
```

Advent of Licat

1. Chapter Objectives

`move`: You can move the character one space using the `move()` function.

`pick`: You can pick up an item under the character using the `pick()` function.

`say`: The character can speak using the `say()` function.

2. Story

Cat's mother eventually couldn't overcome her worsening condition. Holding back tears, Cat did his best to show a smile on her final journey. Nevertheless, he was in full despair for being helpless to do anything.

Not to let sorrow pull him down, he kept to stay busy. He had lost his family, but he wanted to protect someone else's family by building a hospital. As time passed, Cat grew. He could catch more fish than anyone else, and there was no shop in town larger than Cat’s fish market. However, the hospital remained an unachievable dream.

> ”At this rate, it would take 100 million and 3299 years to build a hospital, meow…”

Cat decided to enter **Lion Town** to earn a huge amount of money. Lion Town is a place where only royal and noble lions can live. For commoners, only selected individuals could enter this town.

After a lot of thought, Cat came up with a clever idea.

Licat in a lion mask

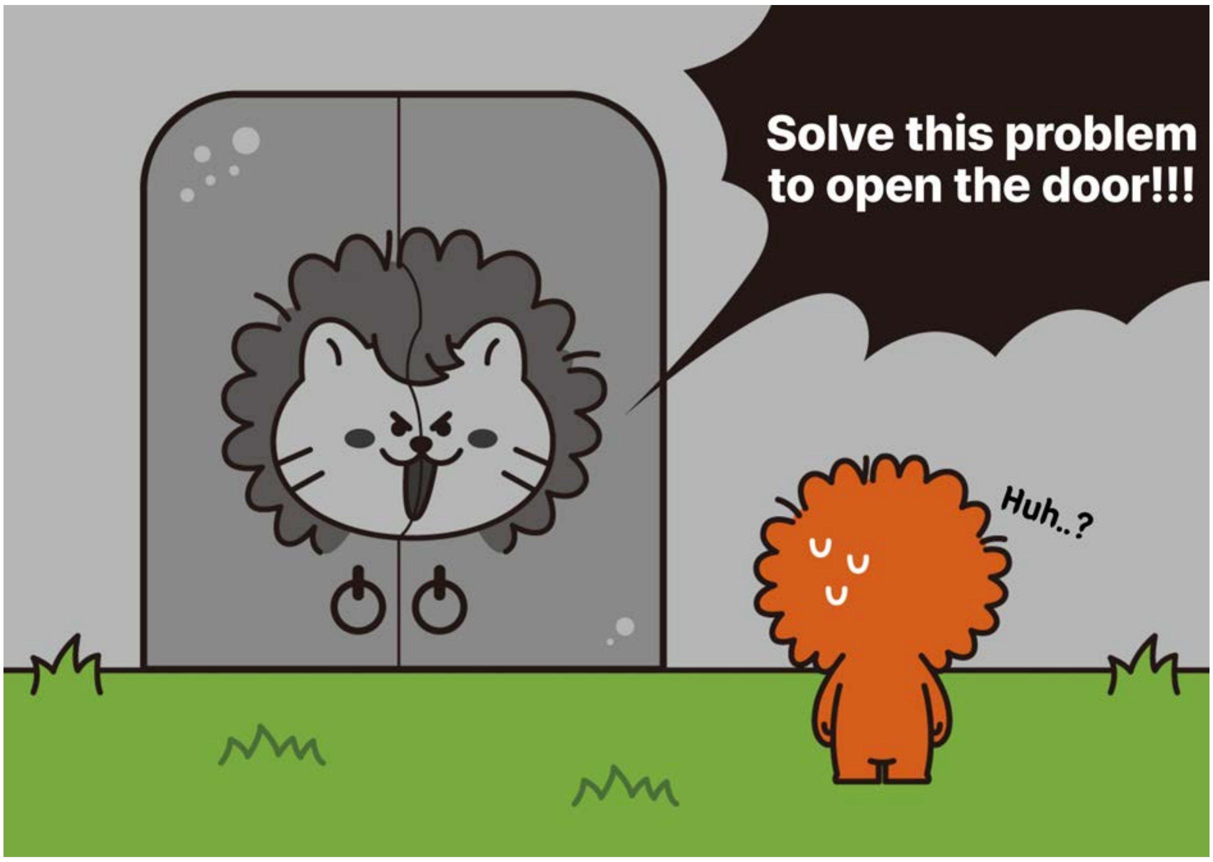
> "I can just wear a lion mask and get in, meow!”

He called himself Licat after Cat wearing a lion mask. Licat thought it would be easy to enter Lion Town, but the gatekeepers were not as lenient as he expected. Denied entry after a strict inspection, he had to agonize again.

At that moment, a staff named Mura came to give him information. Mura looked calm and cold-hearted, so it was difficult to know what she was thinking. But every time she said a word, she was of great help to Licat.

> "There is a secret passage to Lion Town. If you pass there, it would lead to an outskirt of Lion Town. There are few people and they don't conduct inspections, so you should be able to live there comfortably in those clothes. I hope you achieve the dream you want.”

Licat expressed gratitude to Mura and moved to the location. The secret passage was at the end of a complex maze. When Licat placed his hand on the door, it began to speak!

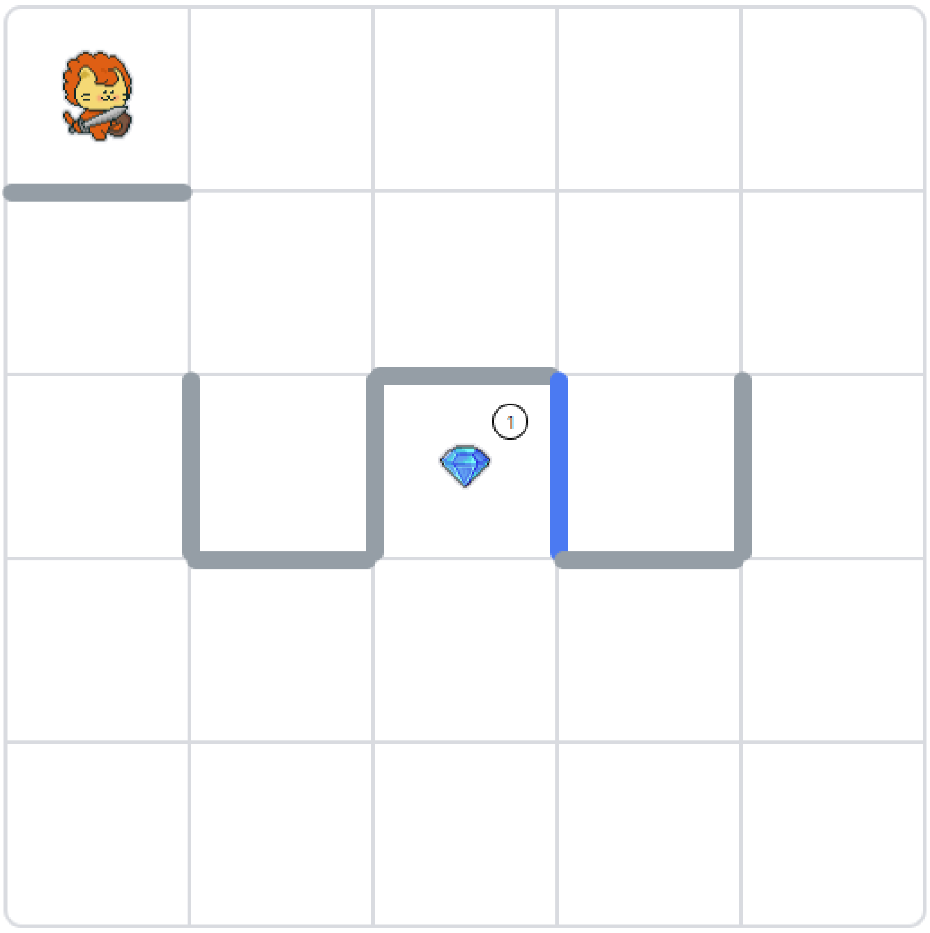


Licat in front of the talking door

> "This door is only for those with the qualifications of the king!
> From now on, I will verify your qualifications! Grrrr!!"

2.1 Mission

The key to the talking door, the `diamond`, has been generated in the maze. Find the `diamond` and shout **'Open the door!'**.

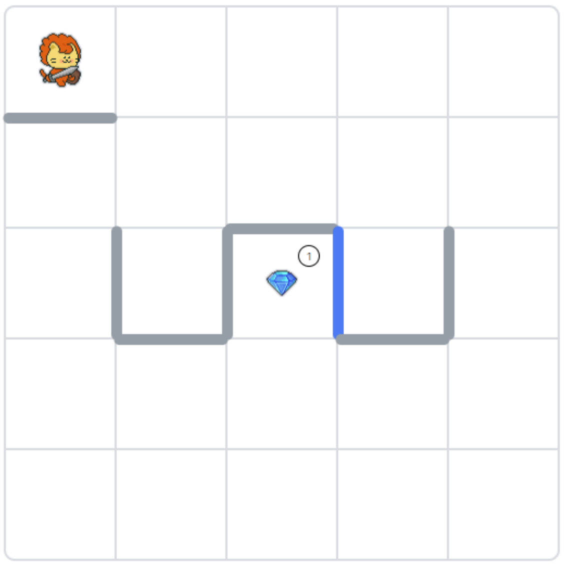


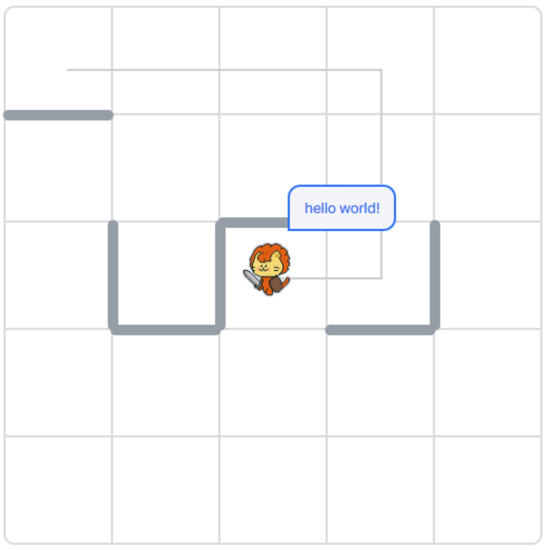
2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
move()
turn_left()
repeat(2, move)
pick()
say('hello world!')
open_door()
```

3. Solution


Before



After

To complete the mission, you should go through the following steps: move forward 3 spaces, turn right, move forward 2 spaces, turn right again, open the door, move forward 1 space, and pick up the diamond.

Let's proceed step by step with what we've learned so far. It is recommended to create a separate cell for each step. First, let's move forward 3 spaces.

```
mission_start()
repeat(3, move)
mission_end()
```

In case this code is unfamiliar, executing the following will produce the same result. If you execute the code below after running the above code, the character will go out of the map since `move()` is executed a total of 6 times. In this case, an alert message will appear, so please use only one of these.

```
mission_start()
move()
move()
move()
mission_end()
```

Licat can rotate only to the left using `turn_left()`. So, how can he face to the right? If he rotates to the left three times, he can face to the right. Later, we are going to learn about functions and modules to create or add `turn_right()`. For now, let's rotate to the right by repeating `turn_left()` three times.

```
mission_start()
repeat(3, turn_left)
mission_end()
```

Now, move forward 2 spaces to make the character face the door.

```
mission_start()
repeat(2, move)
repeat(3, turn_left)
mission_end()
```

The blue wall in front of the character represents the door. To pass through the door, you need to use the `open_door()` command to open it. Once the door is open, it is deleted, and it cannot be closed again.

```
mission_start()
open_door()
mission_end()
```

Now, let’s move forward 1 space, pick up the diamond, and say `"Open the door!”`.

```
mission_start()
move()
pick()
say('Open the door!')
mission_end()
```

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()
repeat(3, move)
repeat(3, turn_left)
repeat(2, move)
repeat(3, turn_left)
open_door()
move()
pick()
say('Open the door!')
mission_end()
```

Licat Heading to Skull Island

1. Chapter Objectives

`variable`: You can assign a value to a variable.

`print`: You can use `print()` to output a value to the terminal.

`type`: You can check the type of a variable using `type()`.

2. Story

After successfully sneaking into Lion Town, Licat was able to quickly settle down. The reason is that he is one of the brave fishermen who can catch fish around the Skull Island, where the most fish are caught in Weniv World. As the name implies, Skull Island had so many reefs and strong currents that countless fishermen lost their lives.

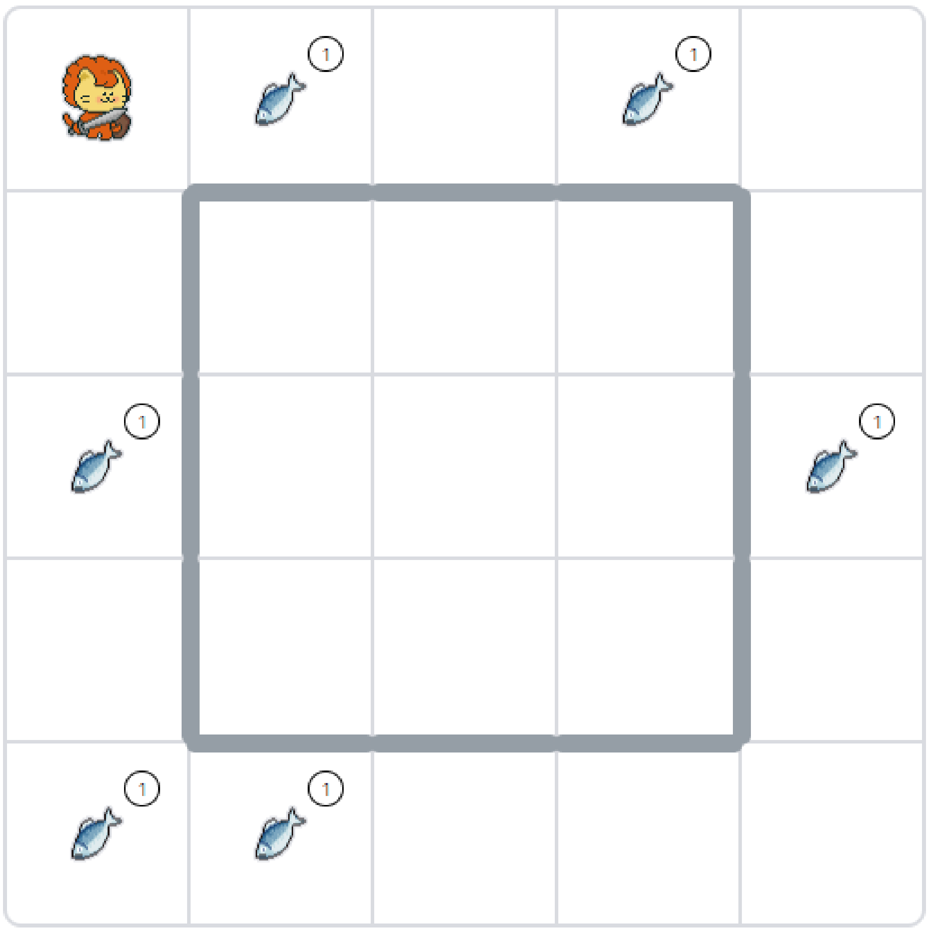
Licat heading to Skull Island

Licat grasped the movement of fish, made a map, added iron plates below to prevent the boat from sinking on the reefs, and improved the net with a better rope.

> "Now, it’s time to sail, meow!"

2.1 Mission

Catch all the `fish` while avoiding the maze. Then output the number of caught fish to the terminal in the form of **'Licat caught 3 fish!**'.



2.2.1 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
move()
turn_left()
repeat(2, move)
pick()
print('hello world!')
print('hello', 'world')
print(f'hello world')
item()
item()['fish-1']
```

2.2.2 Intermediate-Level Hints

```
if
while
in
character_data[0]['x']
character_data[0]['y']
item_data
```

3. Solution

3.1 Variables

Variables are used when you want to represent a value. To use a variable, you need to give it a name and assign a value using the `=` symbol. The `=` symbol is called the assignment operator.

Let's declare a variable to represent the number of fish. Using variables makes it easy to retrieve values.

```
count = 0
print(count)
```

💡 **Must-Know Rules for Variable Names**

1. Variable names must start with an English letter, or an underscore `_`.
2. Uppercase and lowercase letters are distinct. For example, "Apple" and "apple" are different variable names.
3. Avoid using reserved keywords already used in Python. Words like "for" or "def" fall into this category.
4. Variable names cannot contain spaces. If you want to separate words, use underscores (snake_case) or capitalize each word (camelCase). In Python, using underscores is the standard style. (e.g., count_fish)

3.2 Arithmetic Operation

In this problem, you need to understand arithmetic operations to get the number of caught fish. Arithmetic operations include addition, subtraction, multiplication, division, and remainder. The explanation for each functionality is provided in comments below. Text following the `#` symbol on the right side of the code won't be executed. These comments are called **annotations** and are used to add explanations to the code.

```
count = 10
print(count + 3)  # Addition
print(count - 3)  # Subtraction
print(count / 3)  # Division (float)
print(count // 3) # Division (integer, floor)
print(count * 3)  # Multiplication
print(count ** 3) # Exponentiation
print(count % 3)  # Modulo
```

This `*` character represents multiplication, and `**` represents exponentiation. If count is currently 10, `count ** 3` means multiplying 10 three times (`10 * 10 * 10`).

```
print(count * 3)  # Multiplication
print(count ** 3) # Exponentiation
```

There are two types of division in Python. Using a single slash (`/`) will output a quotient with a float datatype, such as `3.333...`. This form with decimals is referred to as a float. If you use two slashes (`//`), it will output an integer result, like `3`. This form, without decimals, is called an integer.

```
print(count / 3)  # Division (float)
print(count // 3) # Division (integer, floor)
```

You can check the type of each with the following.

```
print(type(3.33))
print(type(3))
```

The remainder operation gives you the remainder when dividing. When dividing 10 by 3, the quotient is 3, and the remainder is 1, so it prints 1.

```
print(count % 3) # Modulo
```

Now let's go back to the code where the fish variable was declared.

```
count = 0
```

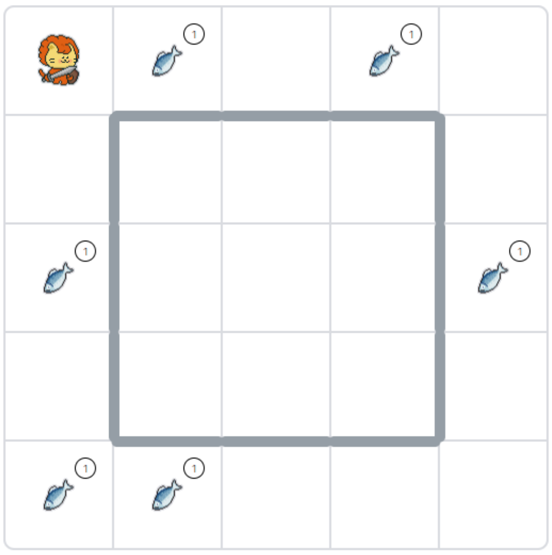
In the code above, if you want to increase the count by 1 each time you catch a fish, you need the following code. The addition is calculated before the assignment. Therefore, the result of `count + 1` (which is 1) is stored in the count.

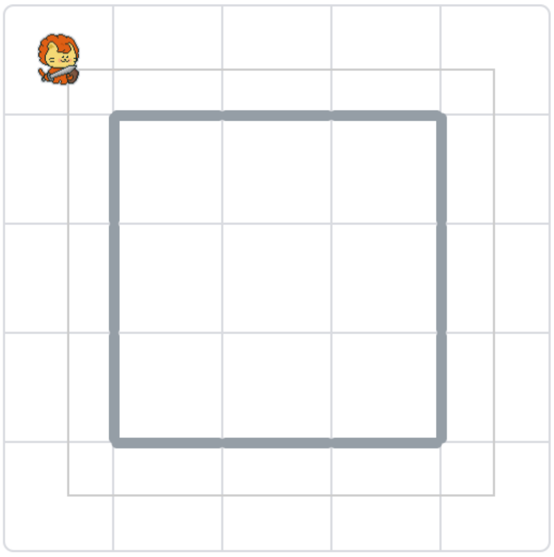
```
count = count + 1
```

This code can be shortened as shown below. Since the shortened code might not be familiar, we'll mainly use the former code for now. However, the following code is more common in working-level.

```
count += 1
```

3.2 Solution


Before


After

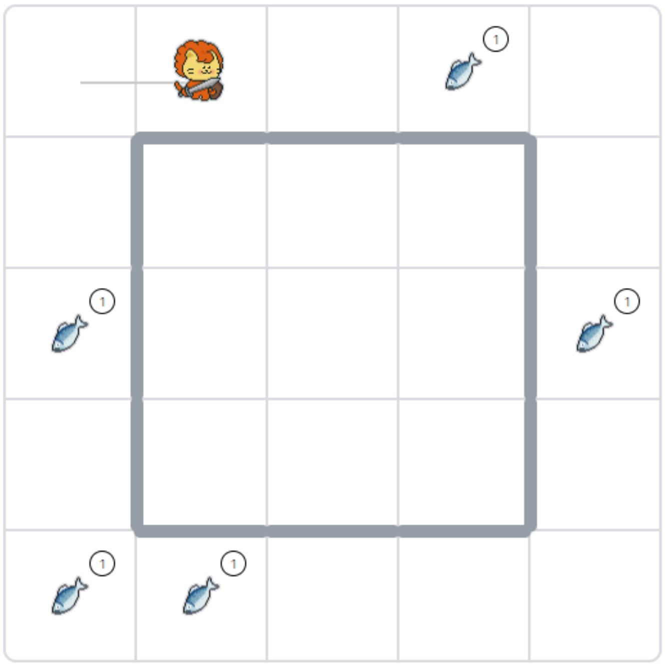
This problem doesn't use automated code. Later, you will be able to combine functions like `front_is_clear()` to check if the front is empty, `on_item()` to check if there's an item beneath, and `while` loops to solve it with a more elegant code.

First, let's try to get all the fish in the first row. Here, we need to count the fish, so we are going to declare a variable to store the number of fish we catch. If you don't plan to use the world, exclude `mission_start()` and `mission_end()`.

```
count = 0
```

As learned before, you can perform arithmetic operations on declared variables. Now, let's increment the variable when catching fish. If you haven't declared count = 0 above, please uncomment `count = 0` and execute the code below.

```
mission_start()

# count = 0
move()
pick()
count = count + 1
print(count)

mission_end()
```



When you run the above code, the character would have caught a fish after moving one space, and the terminal would have printed 1. You can catch the rest of the fish in the same way. Let's write the code for catching one more fish and turning right.

```
mission_start()

repeat(2, move)
```

```
pick()
count = count + 1
move()
repeat(3, turn_left)

mission_end()
```

If you've caught all the fish, you should output in the form of `Licat caught 3 fish!` using the `print()` function. You can print the answer with the following code:

```
count = 3
print('Licat caught 3 fish!')
print('Licat caught ', 3, ' fish!')
print('Licat caught ', count, ' fish!')
print(f'Licat caught {count} fish!')
```

All the above outputs will be the same, saying `Licat caught 3 fish!`. The method used in the last line is called **f-string formatting**. It allows you to directly insert variables for more convenient use and is commonly used in working-level.

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()

count = 0
move()
pick()
count = count + 1
repeat(2, move)
pick()
count = count + 1
move()
repeat(3, turn_left)
repeat(2, move)
pick()
count = count + 1
repeat(2, move)
repeat(3, turn_left)
repeat(3, move)
```

```
pick()
count = count + 1
move()
pick()
count = count + 1
repeat(3, turn_left)
repeat(2, move)
pick()
count = count + 1
repeat(2, move)
print(f'Licat caught {count} fish!')

mission_end()
```

5. Advanced Code

It is an advanded level of code. A description is not given here because it requires a compound description of several concepts.

```
def move_pick():
    move()
    if on_item():
        pick()

mission_start()

repeat(4, move_pick)
repeat(3, turn_left)
repeat(4, move_pick)
repeat(3, turn_left)
repeat(4, move_pick)
repeat(3, turn_left)
repeat(4, move_pick)
repeat(3, turn_left)
print(f'Licat caught {item()["fish-1"]} fish!')

mission_end()
```

```
from modules import turn_left_until_clear, turn_right

mission_start()
```

```
visited = []
while True:
    visited.append((character_data[0]['x'], character_data[0]
['y']))
    if front_is_clear():
        move()
        while on_item():
            pick()
    else:
        turn_right()
        move()
    if (character_data[0]['x'], character_data[0]['y']) in visite
d:
        break
print('Licat caught ', item()['fish-1'], ' fish!')

mission_end()
```

Amazing Taste!

1. Chapter Objectives

`Arithmetic Operators`: You can freely perform addition, subtraction, division, and multiplication.

`Dictionary`: You can understand the dictionary data type and extract values using keys.

`Print`: You can format the output in the desired form using f-string syntax.

2. Story

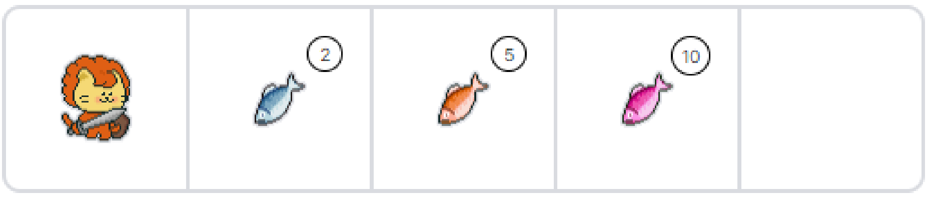
The fish caught on Skull Island is so plump and delicious that its popularity has been increasing as days go by. Even people from other villages were willing to pay more to buy the fish.

We are running out of fish...
Is there any way to catch more fish efficiently?
I'm first!!!!
Get out of the way!!!

Contemplating Licat

Today, let's calculate how much revenue we can make when all the displayed fish are sold.

2.1 Mission



Pick up all the `fish` in the market and calculate how much revenue can be achieved when selling fish-1 for 1000 nodes, fish-2 for 2000 nodes, and fish-3 for 3000 nodes. Output the result as shown below in the terminal. The `count` should be printed using `item()`, and the total should be the value obtained by multiplying `price` and `count`.

```
type    count    price    total
fish-1 2         1000     2000
fish-2 3         2000     6000
fish-3 5         3000     15000
total                     23000
```

2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
move()
repeat(2, move)
pick()
print('hello world!')
print('hello', 'world')
print(f'hello world')
item()
item()['fish-1']
10 + 10
10 - 3
10 / 3
10 // 3
10 * 3
10 ** 3
```

3. Solution

3.1 Dictionary

A dictionary consists of pairs of keys and values. By using this data type, you can retrieve values using the key. For example, `d['one']` will output 1, and `d['two']` will output 2.

```
d = {'one': 1, 'two': 2}
d['one']
d['two']
```

When you check the data type of variable `d` using the `type()` function, it will output `<class 'dict'>`. It is called a dictionary.

```
d = {'one': 1, 'two': 2}
type(d)
```

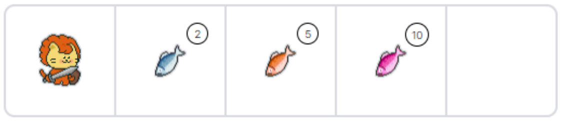
Values in a dictionary can be modified.

```
d = {'one': 1, 'two': 2}
d['one'] = 100
```

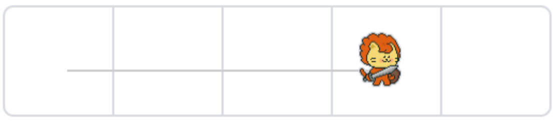
```
d
```

The result of the above code is `{'one': 100, 'two': 2}`.

3.2 Solution



Before

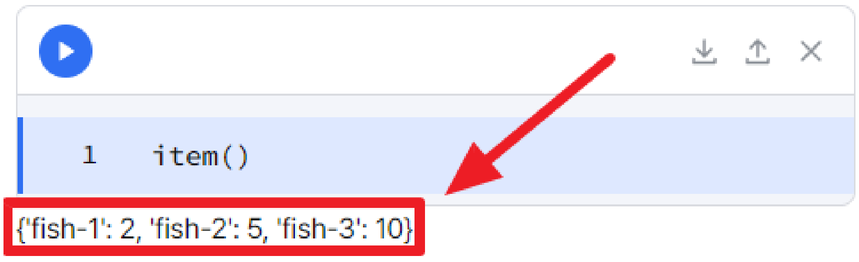

After

This problem focuses on learning arithmetic operations and the dictionary data type rather than moving the character in the world. First, let's pick up all the fish while moving forward.

```
mission_start()

move()
repeat(2, pick)
move()
repeat(5, pick)
move()
repeat(10, pick)

mission_end()
```

We can use the `item()` function to see the picked-up items. In a notebook, when executing a variable or function without `print()`, the result is displayed right below.

```
item()
```



```
item()
```

```
{'fish-1': 2, 'fish-2': 5, 'fish-3': 10}
```

As you can see, the output result is enclosed in curly brackets. It shows that there are two fish-1, five fish-2, and ten fish-3.

Now, let's try to print as follows:

```
type  count    price  total
fish-1 2       1000   2000
fish-2 3       2000   6000
fish-3 5       3000   15000
total                 23000
```

The above code can be printed as follows:

```
print('type   count   price    total')
print('fish-1 2       1000     2000')
print('fish-2 3       2000     6000')
print('fish-3 5       3000     15000')
print('total                   23000')
```

Let's use the variables we can use here. We can calculate and insert the number of caught fish and each total. Here, `item()` is a dictionary. It looks a bit different from what we've learned so far. I'll explain this format further when we learn about functions later.

```
print(f'type   count    price    total')
print(f'fish-1 {item()["fish-1"]}        1000     {item()["fish-
1"] * 1000}')
print(f'fish-2 {item()["fish-2"]}        2000     {item()["fish-
2"] * 2000}')
print(f'fish-3 {item()["fish-3"]}       3000     {item()["fish-3"]
* 3000}')
```

We haven't calculated the sum yet. The code looks very complex, so it is recommended to calculate in advance and then insert the variables. When naming variables, writing in a way that you can easily understand what value it represents will increase readability. Readability is a very important factor in coding. It refers to how easily and clearly code can be understood and interpreted. For instance, `fish1_count` represents the number of fish-1, and `fish_price_all` represents the total price of every fish.

```
fish1_count = item()["fish-1"]
fish2_count = item()["fish-2"]
fish3_count = item()["fish-3"]
fish1_price = fish1_count*1000
fish2_price = fish2_count*2000
fish3_price = fish3_count*3000
fish_price_all = fish1_price + fish2_price + fish3_price
print(f'type   count    price    total')
print(f'fish-1 {fish1_count}        1000     {fish1_price}')
print(f'fish-2 {fish2_count}        2000     {fish2_price}')
print(f'fish-2 {fish3_count}       3000     {fish3_price}')
print(f'total                    {fish_price_all }')
```

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()

move()
repeat(2, pick)
move()
repeat(5, pick)
move()
repeat(10, pick)
item()
fish1_count = item()["fish-1"]
fish2_count = item()["fish-2"]
fish3_count = item()["fish-3"]
fish1_price = fish1_count*1000
fish2_price = fish2_count*2000
fish3_price = fish3_count*3000
fish_price_all = fish1_price + fish2_price + fish3_price
print(f'type   count    price    total')
print(f'fish-1 {fish1_count}        1000     {fish1_price}')
print(f'fish-2 {fish2_count}        2000     {fish2_price}')
print(f'fish-2 {fish3_count}       3000     {fish3_price}')
print(f'total                    {fish_price_all }')

mission_end()
```

5. Advanced Code

It is an advanded level of code. A description is not given here because it requires a compound description of several concepts.

💡 The code below is only printing the final sum.
Please write the code to match the output format.

```
mission_start()

while front_is_clear():
    move()
    while on_item():
        pick()

fish = ['fish-1','fish-2','fish-3']
price = [1000, 2000, 3000]
result = 0
for i in range(len(fish)):
    key = fish[i]
    if key in item():
        result += price[i]*item()[key]

print(result)

mission_end()
```

Employee Promotion

1. Chapter Objectives

`String`: You can understand the characteristics of strings.

`Indexing`: You can call a character using string indexing.

`Slicing`: You can extract specific characters by slicing a string.

`Method`: You can change a string in various ways using string methods.

2. Story

The small fish market has changed its name to `CatsFish` and has become a corporation. Due to such rapid growth, Lion Town's nobles started to get jealous.

Several nobles sent spies to break down Licat. He already knew the spy because Mura had informed him who the spy was.

> 'How do you plan to handle the spy?'

> “She'll come back even if we send her out, and she'll hide even more secretly. So let's rather keep her close!”

Licat had the goal of establishing a hospital that anyone could use. He knew that the hospital could not be established just with the money. Most people in Weniv World should be on his side. Even the enemy.

His eyes sparkled.

2.1 Mission

Take the input string below and appoint Mura as the COO and Hati as the CTO. Copy the template code and modify it so that the output statement is displayed on the terminal.

2.1.1 Basic Code

```
announcement = 'CEO Licat, Team Lead Mura, Manager Hati'
```

2.1.2 Output

```
'Appointing as COO Mura, CTO Hati - CEO Licat'
```

2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
'hello' + 'world'
'hello'[0]
'1, 2, 3'.replace(',', '')
'hello world'[2:6]
```

3. Solution

3.1 String

A string(str) represents characters enclosed in single or double quotation marks. Strings have an order, so you can use position numbers to output specific character as shown below. One thing to note is that spaces are also recognized as a character. In `print(s[5])`, it's not that there's no result value. A space is printed.

```
s = 'hello world'
print(s[0]) # h
print(s[1]) # e
print(s[2]) # l
print(s[3]) # l
print(s[4]) # o
print(s[5]) #
print(s[6]) # w
print(s[7]) # o
print(s[8]) # r
print(s[9]) # l
print(s[10]) # d
```

If you want to extract only "hello", you can use the following.

```
s = 'hello world'
print(s[0], s[1], s[2], s[3], s[4])
print(s[0] + s[1] + s[2] + s[3] + s[4])
```

However, expressing like this seems very cumbersome.

While you can get the character value at a specific position using index numbers in strings, you can also specify a range to extract a portion of the string. Therefore, the code to extract only "hello" can be expressed as follows.

```
s = 'hello world'
s[0:5]
```

With the above code, we can get from 0 to the 5th element (not included). This is called **slicing**.

Methods make it easier to manipulate the corresponding data type. You can check it with `dir()` and is often used together with `type()`.

```
s = 'hello world'
print(type(s))
print(dir(s))
```

By doing this, you can see that a lot of code is printed as shown below.

```
<class 'str'>
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getstate__', '__gt__', '__hash
__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len
__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduc
e__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__seta
ttr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize',
'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'is
ascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnume
ric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'lju
st', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix',
'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartitio
n', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'stri
p', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Something with double underscores indicates magic methods, and without underscores, they are referred to as methods. Magic methods are defined to specify the characteristics of this data type. For example, `__add__` enables the concatenation of strings. Methods are used to conveniently handle the respective data type. For instance, the `__upper__` method transforms all characters into uppercase. Below are commonly used methods and their definitions.

- count : Counts the number of occurrences of the specified characters in the string.

  ```
  s = 'hello world'
  s.count('l') # Outputs 3 as there are three 'l' characters.
  ```

- find: Returns the index of the specified value. Returns -1 if the value is not found.

  ```
  s = 'hello world'
  s.find('l') # Returns 2 as the index of first 'l' is 2.
  ```

- index: Returns the index of the specified value. Raises an error if the value is not found.

  ```
  s = 'hello world'
  s.index('l') # Returns 2 as the index of first 'l' is 2.
  ```

- isdigit: Checks if the string contains only digits.

  ```
  s = 'hello world'
  s.isdigit() # Returns False as it is not composed of digits.
  s = '10'
  s.isdigit() # Returns True as it consists of digits.
  ```

- join: Concatenates into a single string using the specified separator.

  ```
  a = 'hello'
  b = 'world'
  '-'.join([a, b])
  # Outputs 'hello-world' by connecting strings with a preceding
  character.
  ```

  `join` must have one value. To use multiple values, you can group them with brackets (`[]`). The data type grouped in brackets like this is called a list.

- lower: Converts the string to lowercase.

```
s = 'Hello World'
s.lower() # hello world
```

- upper: Converts the string to uppercase.

```
s = 'Hello World'
s.upper() # HELLO WORLD
```

- split: Splits the string into a list based on the specified delimiter.

```
s = 'hello world'
s.split(' ') # ['hello', 'world']
s = '064-000-0000'
s.split('-') # ['064', '000', '0000']
```

- replace: Replaces a text with the specified string.

```
s = 'hello world'
s.replace('hello', 'hi') # hi world
```

- strip: Removes front and back whitespaces.

```
s = '    hello world    '
s.strip() # hello world
```

3.2 Solution

This mission doesn't need to move the world. First, let's change 'Team Lead' and 'Manager' to 'COO' and 'CTO', respectively. Since the code does not move the world, `mission_start()` is not needed.

```
announcement = 'CEO Licat, Team Lead Mura, Manager Hati'
print(announcement.replace('Team Lead', 'COO'))
print(announcement)
```

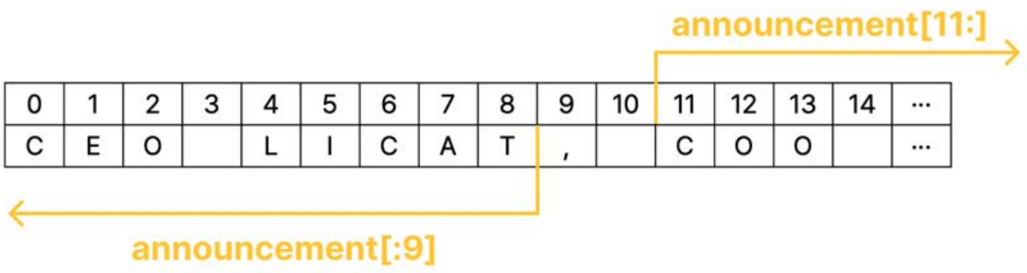
Using the `replace` method as shown above changes the output, but the announcement itself does not change. Therefore, you need to put it back into the announcement as follows:

```
announcement = 'CEO Licat, Team Lead Mura, Manager Hati'
announcement = announcement.replace('Team Lead', 'COO')
print(announcement)
```

Repeat this process:

```
announcement = 'CEO Licat, Team Lead Mura, Manager Hati'
announcement= announcement.replace('Team Lead', 'COO')
announcement= announcement.replace('Manager', 'CTO')
print(announcement)
```

Remove unnecessary characters at the beginning (`CEO Licat,`) using slicing, and add it at the end like below.



```
print(f'Appointing as {announcement[11:]} - {announcement[:9]}')
```

4. Answer Code

```
announcement = 'CEO Licat, Team Lead Mura, Manager Hati'
announcement = announcement.replace('Team Lead', 'COO')
announcement = announcement.replace('Manager', 'CTO')
print(f'Appointing as {announcement[11:]} - {announcement[:9]}')
```

Is This a Bank or a Fish Market?

1. Chapter Objectives

`Comparison Operation`: You can perform comparison operations.

`Built-in Function`: You can understand and utilize various built-in functions.

`List`: You can understand the structure of list and utilize them.

2. Story

Licat wanted to establish a hospital, but he thought earning money could not be the only goal of `CatsFish Inc`. He believed that the company's goal was to produce value together, and above all, the happiness of employees.

To make them happy, Licat made efforts to secure more spare time and not to let them work late into the night and on weekends.

Therefore, they needed a productivity improvement. How could productivity be increased? How could more be produced in less time and more rest be secured?

> "It's a system! We need to grow together by changing the system!”

The size of the company became too big to change the whole system, so they decided to change the small part first.

First of all, they took a day off when the fish were sold least. And on days when there were the most sales, they held events so that they could earn more profits.

Help Licat establish the system of `CatsFish`!

2.1 Mission

Each cell represents the amount of money that fish were sold on Monday, Tuesday, Wednesday, Thursday, and Friday. Fish has not been sold yet on Friday, so it remains as fish only. The gold bar is 100,000 nodes, and fish-3 is 3,000 nodes.

1. Pick up all the gold bars sold each day and store them in a list.
2. What is the minimum amount sold on Monday, Tuesday, Wednesday, Thursday, and Friday? Use `min` to print the minimum amount of money on the terminal.
3. On which day was the most sold? Print the day on the terminal. An event will be held on this day.
4. On which day was the least sold? Print the day on the terminal. On this day, the store is closed.

2.1.1 Output

```
Minimum sales: 30000
Event day: Monday
Day off: Friday
```

2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
l.append()
l.index()
max(1, 2, 3)
min(1, 2, 3)
sum([1, 2, 3])
move()
repeat(2, move)
pick()
print('hello world!')
item()
item()['fish-3']
item()['goldbar']
10 + 10
10 * 3
10 > 20
30 < 10
10 >= 5
```

```
3 == 3
3 != 5
5 <= 10
```

3. Solution

3.1 List

A list is a mutable data type with an order.

```
l = [10, 20, 30]
l[0] = 1000
l # [1000, 20, 30]
```

Similar to the string data type, you can use indexing to access a specific element and replace it with another value. In the example above, the value at index 0 is replaced with 1000.

As a list has an order, you can slice it as below.

```
l = [10, 20, 30, 40, 50, 60, 70]
l[2:4] # [30, 40]
```

Let's look at what methods are available using the `type` and `dir` functions.

```
l = [10, 20, 30]
print(type(l))
print(dir(l))
```

Here are the output results:

```
<class 'list'>
['__add__', '__class__', '__class_getitem__', '__contains__', '__d
elattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__forma
t__', '__ge__', '__getattribute__', '__getitem__', '__getstate__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_
subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '_
_reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof_
_', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'cou
nt', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sor
t']
```

As shown above, the data type is called 'list.' Similar to string, it has both magic methods and regular methods. Below are commonly used methods.

- append: Adds a value to the end.

  ```
  l = [10, 20, 30, 40, 50]
  l.append(60) # [10, 20, 30, 40, 50, 60]
  ```

- count: Counts the occurrences of specified elements, similar to the count method for strings.

  ```
  l = [10, 20, 30, 40, 10]
  l.count(10) # Outputs 2, as there are two occurrences of 10.
  ```

- index: Outputs the index of the first occurrence of a specified value, similar to the index method for strings.

  ```
  l = [10, 20, 30, 40, 50]
  l.index(30) # Outputs 2, as 30 is at index 2.
  ```

- insert: Inserts a specified value at a specified position.

  ```
  l = [10, 20, 30, 40, 50]
  l.insert(2, 1000) # Inserts 1000 at index 2.
  l # [10, 20, 1000, 30, 40, 50]
  ```

- pop: Removes and returns the last element.

  ```
  l = [10, 20, 30, 40, 50]
  l.pop() # 50
  l # [10, 20, 30, 40]
  ```

- remove: Removes the first occurrence of a specified value.

  ```
  l = [10, 20, 30, 40, 50]
  l.remove(30) # Removes the value 30.
  l # [10, 20, 40, 50]
  ```

- reverse: Reverses the order of elements.

```
l = [10, 20, 30, 40, 50]
l.reverse()
l # [50, 40, 30, 20, 10]
```

- sort: Sorts the elements in ascending order.

```
l = [20, 30, 10, 40, 50]
l.sort()
l # [10, 20, 30, 40, 50]
```

3.2 Built-in Functions

Built-in functions are functions that we use without declaring. For example, `print`, `dir`, and `type` are all built-in functions. You can view the list of built-in functions on the official website below. You can also search for Python built-in functions on Google.

Built-in Functions

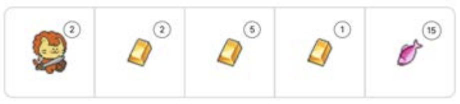
The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.,,,, Built-in Functions,,, A, abs(), aiter(), all(), a...

https://docs.python.org/3/library/functions.html

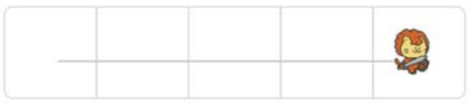
Among them, we will use `min`, `max`, and `sum` functions. As the names suggest, these functions output the minimum, maximum, and sum values, respectively.

```
l = [10, 20, 30, 40, 50]
print(min(l)) # 10
print(max(l)) # 50
print(sum(l)) # 150
```

3.3 Solution



Before


After

We need to create lists for sales and days of the week, then pick up items while moving and add the items obtained to the end of the sales list. Then write a code to reset the picked-up items.

```
mission_start()

sales = []
days_of_week = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

repeat(2, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0

move()

mission_end()
```

First, pick up two items below and put the sales amount in the sales list. The sales amount is calculated by multiplying the number of items by the value of the picked up items. This is because the types of obtained items are not all 'goldbar'. It needs to be converted to values so that `min`, `max`, and `sum` functions can be used.

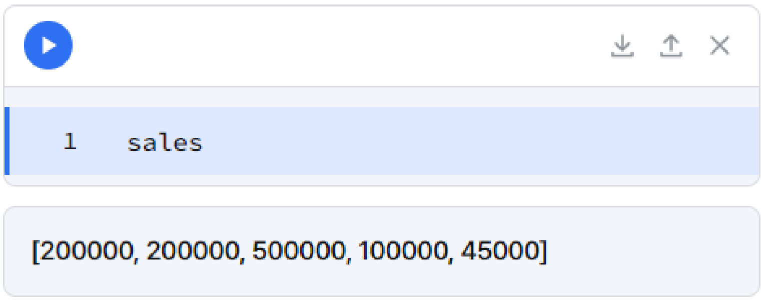
Use this code to calculate the sales amounts for Monday, Tuesday, Wednesday, Thursday, and Friday.

```
mission_start()

sales = []
days_of_week = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

repeat(2, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()

repeat(2, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()

repeat(5, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()

repeat(1, pick)
sales.append(item()['goldbar']*100000)
```

```
item()['goldbar'] = 0
move()

repeat(15, pick)
sales.append(item()['fish-3']*3000)
item()['fish-3'] = 0

mission_end()
```

Now, print the sales amounts as follows:

```
sales
```



```
sales
```

```
[200000, 200000, 500000, 100000, 45000]
```

You can find the minimum amount in the sales amounts as shown below:

```
print(f'Minimum sales: {min(sales)}')
```

Event days and days off cannot be determined solely from the sales list; the days of the week list needs to be used.

```
print(f'Event day: {days_of_week[sales.index(max(sales))]}')
print(f'Day off: {days_of_week[sales.index(min(sales))]}')
```

Note that in this example, the `index()` function is used to find the position of the highest value in the sales amounts, and then this index is used for indexing the days of the week list. The final output is from the days of the week list.

Although not directly used in this code, let's learn comparison operations in this chapter. Comparison operations compare two values, and the result is represented as `True` or `False`.

```
x = 10
y = 3
print(x > y)  # Is x greater than y? True
print(x >= y)  # Is x greater than or equal to y? True
print(x < y)  # Is x less than y? False
print(x <= y)  # Is x less than or equal to y? False
print(x == y)  # Is x equal to y? False
print(x != y)  # Is x not equal to y? True
```

These values, represented as `True` or `False`, are called Boolean values.

```
x = True
print(type(x)) # <class 'bool'>
```

In this problem, you can compare as follows:

```
sales[0] > sales[1]
```

4. Answer Code

```
mission_start()

sales = []
days_of_week = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

repeat(2, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()

repeat(2, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()

repeat(5, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()

repeat(1, pick)
sales.append(item()['goldbar']*100000)
item()['goldbar'] = 0
move()
```

```
repeat(15, pick)
sales.append(item()['fish-3']*3000)
item()['fish-3'] = 0

print(f'Minimum sales: {min(sales)}')
print(f'Event day: {days_of_week[sales.index(max(sales))]}')
print(f'Day off: {days_of_week[sales.index(min(sales))]}')
```

Free Food Truck

1. Chapter Objectives

`Conditional Statement`: You can use `if`, `elif`, and `else` statements.

`Logical Operation`: You can perform `and`, `or`, and `not` operations.

2. Story

Licat has been secretly moving gold bars outside of Lion Town.

> “They can buy snacks for 1 gold in Lion Town, but in Weniv World, 10 families can live with the money for 10 months, meow!”

He selected reliable employees and raised money with gold bars. And with the money, he’s been running a food truck for a long time.

> "I'll take care of the food, meow! So don't worry about what to eat, what to wear, and where to sleep. Spend time for bigger value, meow!"

Most didn't know who ran the food truck, but many thanked for it. As time passed, some of them figured out who was running the food truck and secretly followed Licat.

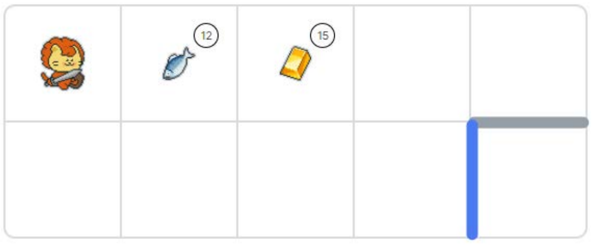
Among them, Hati had been watching the food truck coming to the village where she was born for a long time.

Working as a spy on `CatsFish` market, she could find out who was running a food truck. Even though she was a spy, she became swayed by his sincerity.

2.1 Mission

If there are 10 or more fish-1 and 10 or more gold bars, Licat enters the space (1, 4), puts a fish-1, and says that he is running a food truck.

Otherwise, he should say that the food truck is not running in the space (1, 4). Both conditions must be satisfied.



2.1.1 Basic Code

```
if condition:
    say('We are running a food truck!')
else:
    say('We are not running a food truck!')
```

2.2 Hints

Complete the mission by combining the codes below.

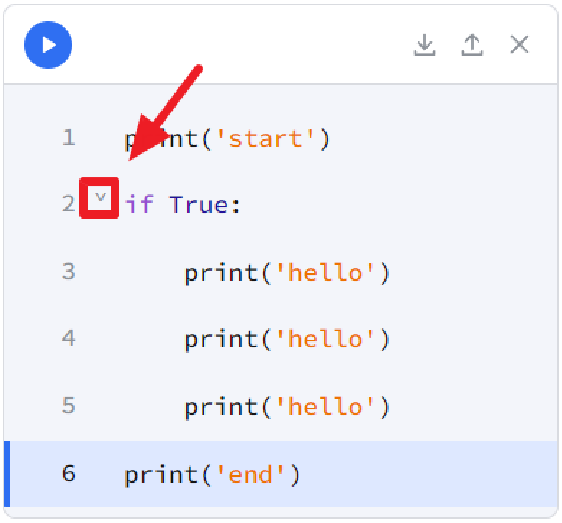
```
mission_start()
mission_end()
True and False
True or False
10 > 5 and False
move()
repeat(2, move)
pick()
say('hello world!')
show_item()
show_item()['fish-1']
10 > 20
30 < 10
10 >= 5
3 == 3
3 != 5
5 <= 10
```

3. Solution

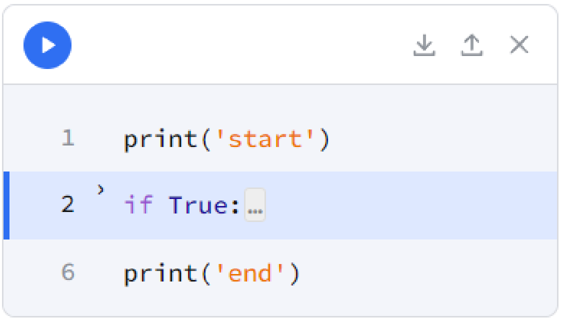
3.1 Conditional Statement

A conditional statement is used to execute the code within the `if` block when the specified condition is `True`.

```
print('start')
if True:
    print('hello')
    print('hello')
    print('hello')
print('end')
```



```
print('start')
if True:
    print('hello')
    print('hello')
    print('hello')
print('end')
```



```
print('start')
if True:...
print('end')
```

The code block's scope can be verified by clicking the square button above. The folded area belongs to the range of the `if` statement, indicated by an indentation of 4 spaces.

Let's see each result when the condition of the `if` statement is `True` and `False`.

```
start
hello
hello
hello
end
```

```
start
end
```

When the `if` statement is `False`, the code inside the `else` statement is executed. Here is an example:

```
print('start')
if True:
    print('hello')
else:
    print('world')
print('end')
```

In this code, if the condition after the `if` statement is `True`, it prints 'hello', otherwise, it prints 'world'. While the `if` statement can be used independently, the `else` statement cannot stand alone. In addition to `if` and `else`, there is also `elif` in conditional statements.

```
x = 10
if x > 10:
    print('x is greater than 10')
elif x == 10:
    print('x is 10')
else:
    print('x is less than 10')
```

`elif` is a shorthand for `else if`. It is executed if the previous condition is `False` and the subsequent condition is `True`. Multiple `elif` statements can be used as shown below.

```
x = 76
if x >= 90:
    print('A')
elif x >= 80:
    print('B')
elif x >= 70:
    print('C')
else:
    print('D')
```

Here, the code will sequentially check if the conditional statement is `True` or `False`. In the first `if` statement, it moves to the next code since `x` is not greater than or equal to 90. In the first `elif` statement, it moves to the next code again since `x` is not greater than or equal to 80. In the next `elif` statement, `x` is greater than or equal to 70, so it prints 'C' and does not proceed to the `else` statement. This is because a `True` condition has already been found.

Logical operations may seem unfamiliar, but they are essential concepts in computer science. Therefore, it is recommended to practice and summarize them as shown below:

```
# True is 1, False is 0
# 'and' is multiplication, 'or' is addition
# 'not' is negation
True and False  # First example
True or False   # Second example
True or True    # Third example
not True         # Fourth example
```

In the first example, `True and False` is equivalent to 1 multiplied by 0. Since it is a multiplication, if either one is 0, the result will be 0. Therefore, the `and` operation returns `False` if either is `False`.

In the second example, `True or False` is equivalent to 1 plus 0. Since it is an addition, if either one is 1, the result will be 1 or more. Therefore, the `or` operation returns `True` if either is `True`.

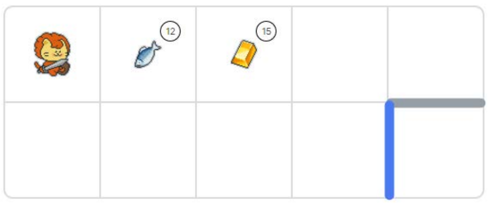
The third example results in 1 plus 1, and any number other than 0 is considered `True`.

In the fourth example, `not True` returns `False`, and `not False` returns `True`.

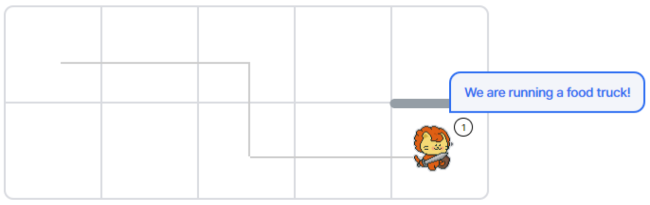
Here used addition and multiplication to simplify the explanation. However, in actual Python, the interpreter does not work in this way but evaluates statements. For example, if `False` is encountered first, and `and` operator follows, the subsequent statement is not evaluated and `False` is returned.

3.2 Solution

Our mission is as follows: to pick up all the items, open the door, enter, place one fish, and then say whether to operate a food truck.



Before



After

First, move forward and pick up all the items.

```
mission_start()

move()
repeat(12, pick)
move()
repeat(15, pick)
```

```
mission_end()
```

Then move to the door.

```
mission_start()

repeat(3, turn_left)
move()
turn_left()
move()

mission_end()
```

Licat moves to open the door and sees if there are 10 or more fish-1 and 10 or more gold bars. If there are, he puts down one fish-1 and says "We are running a food truck!" Otherwise, say "We are not running a food truck!"

```
mission_start()

open_door()
move()

if item()['fish-1'] >= 10 and item()['goldbar'] >= 10:
    put('fish-1')
    say("We are running a food truck!")
else:
    say("We are not running a food truck!")

mission_end()
```

What if the condition is `or` instead of `and`? In that case, the food truck will run if either condition is satisfied.

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()

move()
repeat(12, pick)
move()
```

```
repeat(15, pick)

repeat(3, turn_left)
move()
turn_left()
move()
open_door()
move()

if(item()['fish-1'] >= 10 and item()['goldbar'] >= 10):
    put('fish-1')
    say('We are running a food truck!')
else:
    say('We are not running a food truck!')

mission_end()
```

```
mission_start()

move()
repeat(12, pick)
move()
repeat(15, pick)

repeat(3, turn_left)
move()
turn_left()
move()
open_door()
move()

if item()['fish-1'] >= 10 or item()['goldbar'] >= 10:
    put('fish-1')
    say("We are running a food truck!")
else:
    say("We are not running a food truck!")

mission_end()
```

Warehouse Integration

1. Chapter Objectives

`List`: You can retrieve values and modify them using list indexing.

`Arithmetic Operation`: You can perform arithmetic operations.

`Iteration Statement`: You can use `for` and `while` loops.

`Type Casting`: You can change the data type using `int`, `str`, `float`, etc.

2. Story

Pika is a strategist. With much experience in launching services, Pika is taking care of all plans at `CatsFish Inc.`

The warehouse of `CatsFish` is full, and there is no place to store fish anymore. Therefore, Pika wants to build a large warehouse to store fish. The problem is how many floors the warehouse should have.

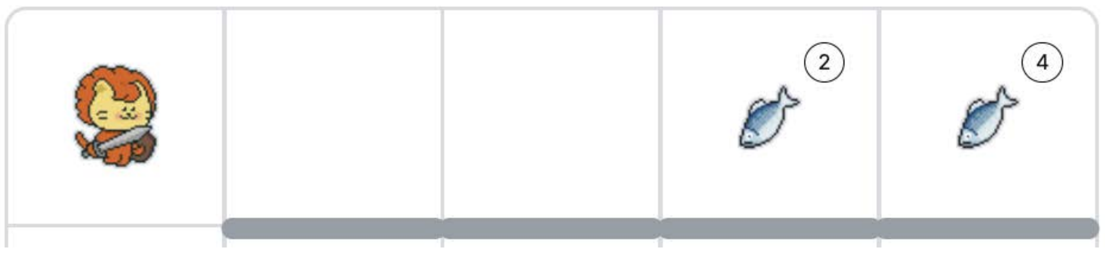
> "Oh, this should be fun."

Pika likes challenging plans. He plans to calculate the growth rate of `CatsFish` and build a warehouse that is 10 times the total number of fish in the warehouse.

Each new floor of the warehouse will be able to store 100 fish. Help Pika calculate how many floors of the warehouse are needed and output the result to the terminal.

2.1 Mission

Each line represents one warehouse. From right to left, it stands for fish in the ones place, tens place, and hundreds place. For example, if there are 2 fish at (0, 3) and 4 fish at (0, 4), the warehouse has a total of 24 fish.



The new warehouse needs to be built to hold 10 times the fish in the warehouse, so we need a warehouse that can store 240 fish. Since each floor can store 100 fish, a total of 3 floors are needed.

Put all the fish in the warehouse into a `list` and output to the terminal how many floors of the warehouse need to be built.

2.1.1 Basic Code

```
l = [0, 0, 0, 0] # [Thousands place, Hundreds place, Tens place, O
nes place]
```

```
for i in l:
    print(i)
```

2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
move()
repeat(2, move)
pick()
print('hello world!')
show_item()
show_item()['fish-1']
10 + 10
10 * 3
10 // 3
```

```
10 >= 20
30 < 10
```

3. Solution

3.1 Iteration Statements

Iteration statements refer to repeating a specific code for a desired number of times. In Python, there are two types of iteration statements: `for` loops and `while` loops.

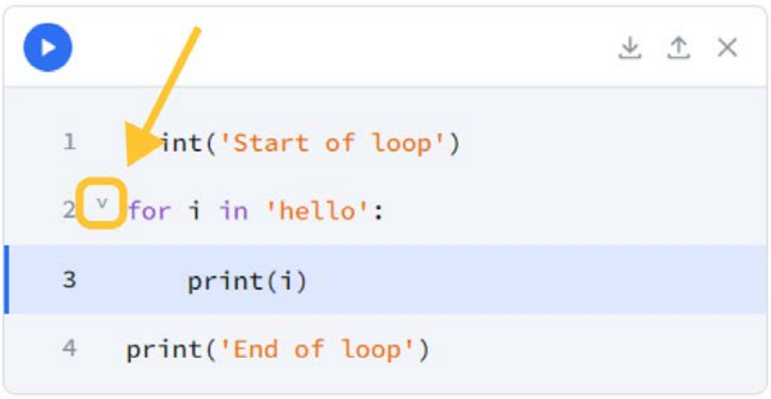
3.1.1 For Loops

A `for` loop is a way to iterate through elements in an iterable. Let's look at an example:

The syntax is as follows:

```
for variable in iterable:
    code_to_repeat
```

In the example below, the variable is 'i', and the iterable is the string 'hello'. Similar to an `if` statement, you can collapse the range of the `for` loop by using the button in the left corner. The scope of the `for` loop is indicated by an indentation of 4 spaces.



```
int('Start of loop')
for i in 'hello':
    print(i)
print('End of loop')
```

```
print('Start of loop')
for i in 'hello':
    print(i)
print('End of loop')
```

```
Start of loop
h
e
l
l
o
End of loop
```

Here, if you add 4 spaces preceding `print('End of loop')`, output is like below. Make sure to keep the correct indentation to avoid unintended repetitions of code.

```
print('Start of loop')
for i in 'hello':
    print(i)
    print('End of loop')
```

```
Start of loop
h
End of loop
e
End of loop
l
End of loop
l
End of loop
o
End of loop
```

Iterable elements are not limited to strings. Lists and dictionaries, which we learned in previous chapters, can also be used. Integer and float types are not iterable.

An example below is iterating elements in a list. It prints until the end of the elements and terminates the loop.

```
print('Start of loop')
for i in [10, 20, 30]:
    print(i)
print('End of loop')
```

```
Start of loop
10
20
30
End of loop
```

The next example demonstrates iterating over a dictionary. In the loop, it iterates through each key in the dictionary. Similarly, the loop stops after printing the last element.

```
print('Start of loop')
for i in {'one': 1, 'two':
2}:
    print(i)
print('End of loop')
```

```
Start of loop
one
two
End of loop
```

If you want a simple repetition and not iterate over any specific data type, you can use `range`. `range` helps to iterate a specified number of times. We won't cover all the forms of `range` in this tutorial.

```
print('Start of loop')
for i in range(3):
    print(i)  # Repeated 3 ti
mes.
print('End of loop')
```

```
Start of loop
0
1
2
End of loop
```

3.1.2 While Loops

`while` loops provide a more intuitive way to iterate. They repeat a block of code as long as the specified condition is `True`.

```
print('Start of loop')
count = 0
while count < 5:
    print(count)
    count = count + 1
print('End of loop')
```

```
Start of loop
0
1
2
3
4
End of loop
```

In the above example, without the `count = count + 1` code, it would be an infinite loop. Be sure to explicitly state the condition for exiting the loop. You can also use `break` to exit the loop:

```
print('Start of loop')
count = 0
while True:
    print(count)
    if count == 3:
        break
    count = count + 1
print('End of loop')
```

```
Start of loop
0
1
2
3
4
End of loop
```

3.2 Type Casting

Type casting refers to the process of converting a variable from one data type to another. Let's go through an example:

```
'10' + '10'
```

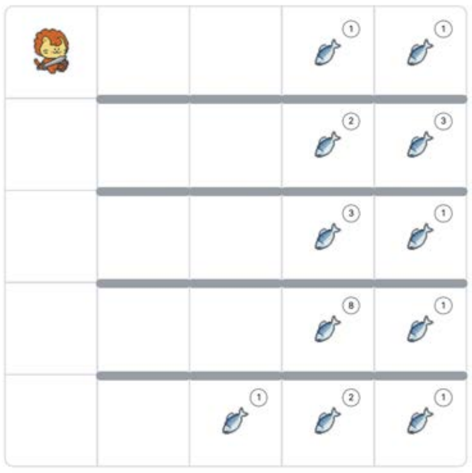
This above operation concatenates two strings. Therefore, the output is '1010'.

```
int('10') + int('10')
```

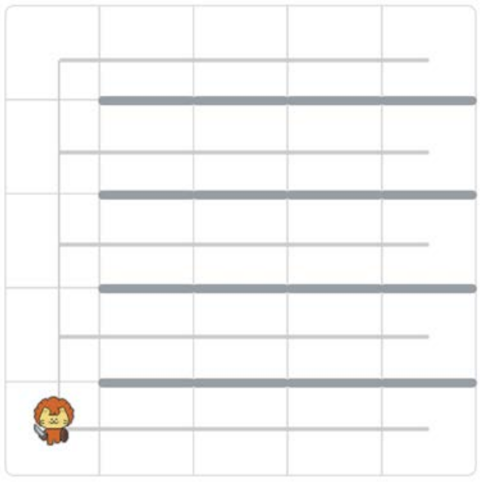
The above operation is converting the strings into integers before addition. It results in numerical addition, so the output is 20.

In Python, type casting functions like `int()`, `float()`, `str()`, `list()`, and `dict()` are used to convert one type to another.

3.3 Solution



Before



After

First, create a list to input each place value.

```
l = [0, 0, 0, 0]
```

Move forward and pick up items. During this process, update the list with the number of picked items in the corresponding positions, and always reset the value of the picked fish.

```
mission_start()

# Code for picking up fish
repeat(3, move)
repeat(1, pick)
l[2] = item()['fish-1']
item()['fish-1'] = 0

move()
repeat(1, pick)
l[3] = item()['fish-1']
```

```
item()['fish-1'] = 0

mission_end()
```

After picking up all the fish in the first row, the list `l` becomes `[0, 0, 1, 1]`, and it should be perceived as a total of 11 fish later. Now that all the fish are picked, it's time to turn around and come out.

```
mission_start()

# Code for turning around and coming out
repeat(2, turn_left)
repeat(4, move)
turn_left()
move()
turn_left()

mission_end()
```

After performing this process 5 times, all the fish on every floor will be caught. If you print the fish that have been caught, the output result will be as follows:

```
print(l) # [0, 1, 16, 7]
```

There is 16 in the tens place without any carryover. In the following code, each digit is multiplied by the corresponding place value (thousands, hundreds, tens, ones), and the results are added to calculate the final value.

```
fish_count = 0
m = 1000
fish_count = fish_count + l[0] * m  # Thousands place
m = 100
fish_count = fish_count + l[1] * m  # Hundreds place
m = 10
fish_count = fish_count + l[2] * m  # Tens place
m = 1
fish_count = fish_count + l[3] * m  # Ones place
print(fish_count) # 267
```

You can simplify it using a loop:

```
fish_count = 0
m = 1000
```

```
for i in range(4):
    fish_count = fish_count + l[i] * m
    m = m / 10
print(fish_count)
```

It was mentioned that one floor is built for every 100 fish. Therefore, if the summed value was 270, you need to multiply it by 10 and divide by 100, which results in building 27 floors. You need to add 1 floor if it is not divisible by 100. For example, if there were 271 fish, you need to build 28 floors.

```
if (fish_count * 10) % 100 == 0:
    print(int(fish_count * 10) / 100)
else:
    print(int((fish_count * 10) / 100 + 1))
```

Alternatively, you can use double slashes to ensure integer division from the beginning.

```
if (fish_count * 10) % 100 == 0:
    print((fish_count * 10) // 100)
else:
    print((fish_count * 10) // 100 + 1)
```

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()

l = [0, 0, 0, 0]

# Code for picking up fish
repeat(3, move)
repeat(1, pick)
l[2] = item()['fish-1']
item()['fish-1'] = 0

move()
repeat(1, pick)
l[3] = item()['fish-1']
item()['fish-1'] = 0

# Code for turning around and coming out
```

```
repeat(2, turn_left)
repeat(4, move)
turn_left()
move()
turn_left()

# Code for picking up fish
repeat(3, move)
repeat(2, pick)
l[2] = l[2] + item()['fish-1']
item()['fish-1'] = 0

move()
repeat(3, pick)
l[3] = l[3] + item()['fish-1']
item()['fish-1'] = 0

# Code for turning around and coming out
repeat(2, turn_left)
repeat(4, move)
turn_left()
move()
turn_left()

# Code for picking up fish
repeat(3, move)
repeat(3, pick)
l[2] = l[2] + item()['fish-1']
item()['fish-1'] = 0

move()
repeat(1, pick)
l[3] = l[3] + item()['fish-1']
item()['fish-1'] = 0

# Code for turning around and coming out
repeat(2, turn_left)
repeat(4, move)
turn_left()
move()
turn_left()

# Code for picking up fish
```

```
repeat(3, move)
repeat(8, pick)
l[2] = l[2] + item()['fish-1']
item()['fish-1'] = 0

move()
repeat(1, pick)
l[3] = l[3] + item()['fish-1']
item()['fish-1'] = 0

# Code for turning around and coming out
repeat(2, turn_left)
repeat(4, move)
turn_left()
move()
turn_left()

# Code for picking up fish
repeat(2, move)
repeat(1, pick)
l[1] = l[1] + item()['fish-1']
item()['fish-1'] = 0

move()
repeat(2, pick)
l[2] = l[2] + item()['fish-1']
item()['fish-1'] = 0

move()
repeat(1, pick)
l[3] = l[3] + item()['fish-1']
item()['fish-1'] = 0

# Code for turning around and coming out
repeat(2, turn_left)
repeat(4, move)
turn_left()

fish_count = 0
m = 1000
fish_count = fish_count + l[0] * m  # Thousands place
m = 100
fish_count = fish_count + l[1] * m  # Hundreds place
```

```
m = 10
fish_count = fish_count + l[2] * m  # Tens place
m = 1
fish_count = fish_count + l[3] * m  # Ones place

# print(int(fish_count)) # Number of caught fish
if (fish_count * 10) % 100 == 0:
    print(int((fish_count * 10) / 100))
else:
    print(int((fish_count * 10) / 100 + 1))

mission_end()
```

Let’s Automate!

1. Chapter Objectives

`Function`: You can define and utilize functions.

2. Story

Licat's fish company has become the fastest-growing distribution company in the entire Weniv World. However, employees became tired of simple and repetitive work.

> "Can't we guarantee autonomy and creative thinking, personal growth, and clear purpose and motivation for each task?"

Licat analyzed each employee's work to see which tasks they spent the most time on. It was fishing, packaging, and delivery which are the simplest and most repetitive tasks.
The next day, Licat said at the meeting:

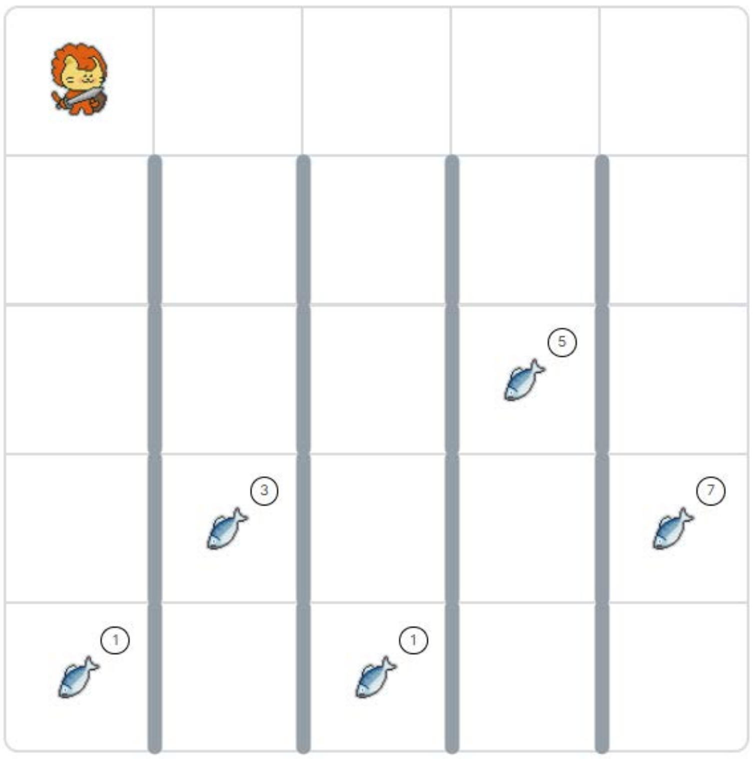
> "Let's adopt robots to replace repetitive tasks! Then we can focus more on creative work, meow!”

However, employees who were already filled with discontent were skeptical about this idea.

> "So who's going to make it!? Is CEO?!”

> "I already made it! So let's take this robot and install it, meow!”

2.1 Mission



Create a function that automatically catches fish and delivers them. You need to fill in the `pass` part in the basic code below.
The function should perform the following: pick up the fish, go up to the top, and say in the form of
`Completed delivery: 3` for each column.

Basic Code

```
def delivery():
    pass
repeat(4, delivery)
```

2.2 Hints

Complete the mission by combining the codes below.

```
mission_start()
mission_end()
move()
turn_left()
repeat(2, move)
pick()
put('fish-1')
```

```
print('hello world!')
front_is_clear()
left_is_clear()
right_is_clear()
```

3. Solution

3.1 Functions

Functions allow you to reuse code and create a more organized structure for better readability. Let's check with the following code:

```
def hello(): # Function definition
    print('hello') # Code inside the function
    print('world')

hello() # Function call
```

When executed like above, Python reads only the function definition and moves on. It doesn't read the code inside the function. When the function is called, Python goes back up, reads, and executes the code inside the function. This way, you can use `hello()` anywhere to print 'hello' and 'world' whenever needed.

Let's create a simple function for addition. When you run the code below, it will output 30. The `return` literally signifies what the function will return. The returned value will be placed where the function was called. In this case, 10 and 20 go into `a` and `b` respectively, and `a` plus `b` is 30, so 30 goes into the place where `add(10, 20)` was.

```
def add(a, b): # Function definition
    return a + b # Code inside the function

add(10, 20) # Function call
```

Here’s another example:

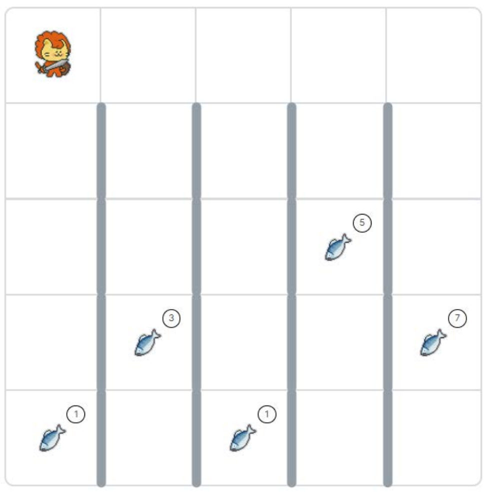
```
def add(a, b): # Function definition
    return a + b # Code inside the function

print(add(10, 20) + add(30, 20) + 30)
```

`add(10, 20)` will have its return value of 30, and `add(30, 20)` will have its return value of 50, so 110 will be output. The results are as follows.

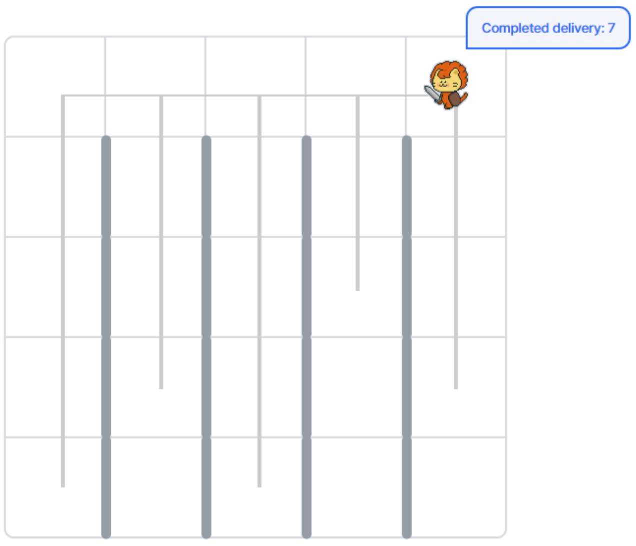
```
print(add(10, 20) + add(30, 20) + 30)
print(         30 +           50 + 30)
```

Here, `a` and `b` are called parameters, and the actual values you put, such as 10 and 20, are called arguments.

3.2 Solution



Before



After

```
def delivery():
    pass
repeat(4, delivery)
```

When looking at the basic code, it is defined to repeat the function four times. However, the same action must be repeated five times as there are five columns. So let's modify it to 5 as shown below:

```
def delivery():
    pass
repeat(5, delivery)
```

Now, it's time to think about what to do inside the `delivery` function. Licat should turn right, pick up all the items below, turn back, and come up to the top again, and finally shout out that the delivery is done. This can be expressed in code as follows. Let's write the code first without using a function:

```
while not on_item():
    move()
```

The above code moves Licat forward until there is an item below using `on_item()`. It returns `True` if there is an item, and `False` if there isn't.

```
while on_item():
    pick()
```

The above code continuously picks up items as long as there is an item below. By using these two codes, you can move to where the items are and pick up all the items below.

Now, you need to turn back and go up to the top. This can be written as follows:

```
repeat(2, turn_left)
while front_is_clear():
    move()
```

The above code turns back Licat and moves him up to the top. `front_is_clear()` returns `True` if the front is empty, and `False` otherwise. It moves only when the front is empty, so Licat can reach the top.

Now, you just need to print how many fish you've picked up and initialize the count:

```
say(f'Completed delivery: {item()["fish-1"]}')
item()["fish-1"] = 0
```

Wrap the above actions in a function to repeat it five times. You should add the following code not to move to the next cell at the very end.

```
if front_is_clear():
    move()
```

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()

def delivery():
    repeat(3, turn_left)
    while not on_item():
        move()
    while on_item():
        pick()
    repeat(2, turn_left)
    while front_is_clear():
        move()
    say(f'Completed delivery: {item()["fish-1"]}')
    item()["fish-1"] = 0
    repeat(3, turn_left)
    if front_is_clear():
        move()

repeat(5, delivery)

mission_end()
```

Let’s Clean up and Organize!

💡 The last problem, "Hospital Establishment," does not have a solution provided. This problem's solution serves as the final one. Try solving the "Hospital Establishment" problem by applying the concepts you've learned so far.

1. Chapter Objectives

`Dictionary`: You can understand the dictionary data type and extract values using keys.

2. Story

As the company grew, Licat was worn out from the busy management and heavy responsibilities.

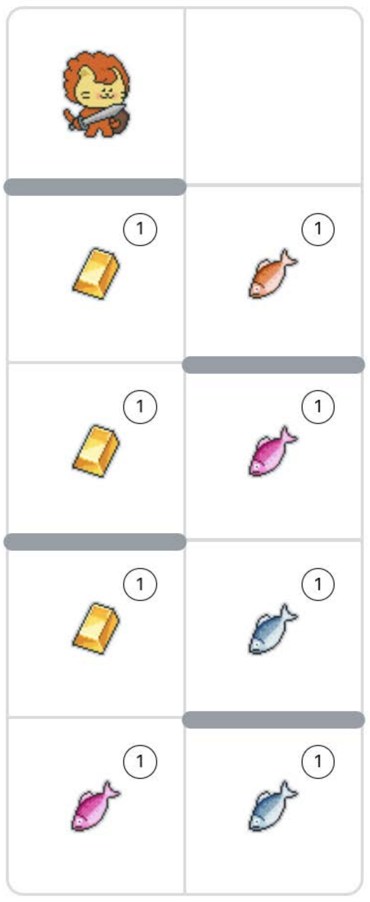
> “Every day is so exhausting. Do I have to do this much? Why do I work and what more can I do in the future?”

With these various thoughts still unorganized, Licat went to `CatsFish` market and started to clean it.

The market was scattered with gold bars and fish. The more he cleaned, the more he could clear his mind.

Letting thoughts just go and focusing on simple tasks, only important things remained and the things had somehow simplified.

2.1. Mission



Create the following dictionary and write a code to print the number of gold bars and the number of fish in the terminal.

Basic Code

```
d = {'goldbar': 0, 'fish': 0}
```

2.2. Hints

Complete the mission by combining the codes below.

```
d['goldbar'] = d['goldbar'] + 1
move()
turn_left()
repeat(2, move)
pick()
print('hello world!')
front_is_clear()
```

```
left_is_clear()
right_is_clear()
```

3. Solution

3.1 Dictionary get()

There is a method called `get` in dictionary. You can print the list of methods using `dir`. Using `get` allows us to extract items more safely. Let's look at the example below.

```
d = {'one': 1, 'two': 2}
d['three']
```

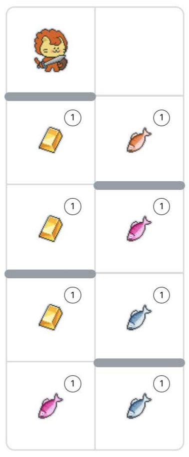
In the above code, it will output `KeyError: 'three'` since the dictionary `d` does not have the key 'three'. To avoid printing an error when the key is not present and instead print a predefined value, you can use `get`.

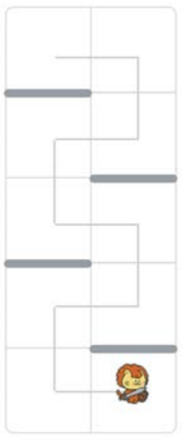
```
d = {'one': 1, 'two': 2}
d.get('three', 'No Value')
```

The above code will output 'No Value.' If you want to output 0 instead, you can modify the code as follows.

```
d = {'one': 1, 'two': 2}
d.get('three', 0)
```

3.2 Solution


Before


After

First, let's create a function that picks up items as long as there are items below and moves to empty spaces continuously .

```
def moving():
    while True:
        if on_item():
            pick()
        if front_is_clear():
            move()
        elif right_is_clear():
            repeat(3, turn_left)
            move()
        elif left_is_clear():
            turn_left()
            move()
        else:
            break
```

Since the code above has an infinite loop due to `while True`, it repeats until the front and both sides are blocked. If both the front and sides are blocked, it breaks out of the infinite loop using the `break` statement.

Now, use the declared function to pick up all items.

```
mission_start()

result = {'goldbar': 0, 'fish': 0}
def moving():
    while True:
        if on_item():
            pick()
        if front_is_clear():
            move()
        elif right_is_clear():
            repeat(3, turn_left)
            move()
        elif left_is_clear():
            turn_left()
            move()
        else:
            break
moving()
```

```
mission_end()
```

Then we need to write a code to organize and put them into gold bars and fish in the dictionary. fish-1, fish-2, and fish-3 should all be contained in the fish of the result.

```
result['fish'] += item().get('fish-1', 0)
result['fish'] += item().get('fish-2', 0)
result['fish'] += item().get('fish-3', 0)
result['goldbar'] += item().get('goldbar', 0)
```

Now, all that's left is to print the result.

```
print(f'There are {result["goldbar"]} gold bars. There are {result
["fish"]} fish.')
```

4. Answer Code

Initialize the world and execute the code below.

```
mission_start()

result = {'goldbar': 0, 'fish': 0}
def moving():
    while True:
        if on_item():
            pick()
        if front_is_clear():
            move()
        elif right_is_clear():
            repeat(3, turn_left)
            move()
        elif left_is_clear():
            turn_left()
            move()
        else:
            break
moving()
result['fish'] += item().get('fish-1', 0)
result['fish'] += item().get('fish-2', 0)
result['fish'] += item().get('fish-3', 0)
result['goldbar'] += item().get('goldbar', 0)
```

```
print(f'There are {result["goldbar"]} gold bars. There are {result
["fish"]} fish.')

mission_end()
```

Appendix

Command Dictionary

Command Dictionary

Function List

- **mission_start()** : indicates the start of the mission
- **mission_end()** : indicates the end of the mission
- **print()** : outputs results to the terminal
- **say()** : displays text in a speech bubble
- **item()** : returns the item held by the character
- **on_item()** : returns whether there is an item under the character
- **directions()** : returns the direction the character is facing
- **move()** : moves the character one space in the direction he is facing
- **turn_left()** : rotates the character to the left(counterclockwise)
- **pick()** : picks up items under the character
- **put(`item`)** : places an item(`item`) under the character if he has the item
- **repeat(`count`, `function`)** : repeats the function(`function`) `count` times
- **open_door()** : removes the wall(`door`) if there is a wall in the character’s movement direction
- **set_item(`x`, `y`, `item`, `count`)** : places the specified quantity(`count`) of items(`item`) at the coordinates (`x, y`) on the map
    - Available `item` types:
        - fish-1
        - fish-2
        - fish-3
        - diamond
        - apple
        - goldbar

- **[`front` | `left` | `right` | `back`]_is_clear()** : determines whether there is a wall in [`front` | `left` | `right` | `behind`] of the character
- **typeof_wall()** : returns the type of wall in the character's movement direction
- **turn_right()** : (from modules import turn_right) rotates the character to the right
- t**urn_around()** : (from modules import turn_around) rotates the character to the opposite direction
- **move_to_wall()** : (from modules import move_to_wall) moves until there's a wall
- **turn_left_until_clear()** : (from modules import turn_left_until_clear) rotates left until the path is clear
- **jump()** : (from modules import jump) jumps over a block even when there is an obstacle

---

Variable List

- **character_data :** shows character information such as location, direction, items, and hp
- **map_data :** shows the size of the map
- **item_data** : shows information about the items placed on the map
- **wall_data['world']** : shows information about the walls placed on the map

Bibliographic Data

[E-book] **Weniv World(Teacher’s Edition) - Game-based Python Education Platform**

First Edition 2024. 01. 22

Author Weniv Inc.

Editor Kyungrim Cha

Chief Editor Hojun Lee

Published by 사도출판

Address 137, Donggwang-ro, Jeju-si, Jeju-do, Republic of Korea

Website http://www.paullab.co.kr

E-mail paul-lab@naver.com

ISBN 979-11-88786-92-3(PDF)

---

weniv. world

https://world.weniv.co.kr/

비매품/무료

45560

9 791188 786923

ISBN 979-11-88786-92-3 (PDF)