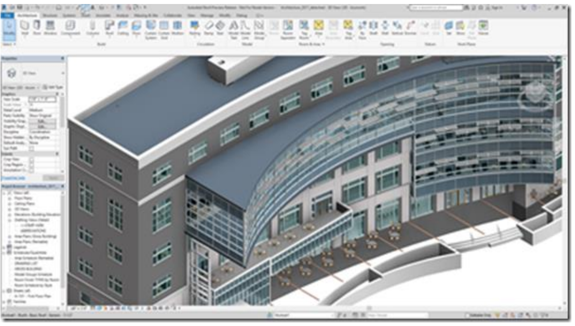
CS 5001: Midterm Hybrid Synthesis

1. Chapter 1: Computational Thinking

1.1. Explain in detail what computational thinking is and how it applies to computer science and programming.

Modern computers are used to do a lot of amazing things such as playing 3A games, computer-aided design, or scientific research modeling, which people want them to do.

Witcher 3



Revit design

Source: https://cdn.pocket-lint.com/r/s/1201x/assets/images/133845-games-review-the-witcher-3-wild-hunt-review-image1-07yik9ul5s.jpg\

https://autodesk.blogs.com/.a/6a00d8341bfd0c53ef022ad39c2fb4200c-pi

To fully participate in an environment with computers, one must have a series of skills for addressing complicated issues. **Computational thinking is to take a difficult situation, identify the issue, and create potential solutions in a way that a computer, a person, or both, can comprehend.**

To be more specific, computers do not understand our natural language (such as English or Chinese) itself, grammar, and logic. Instructions to a computer have to be accurate since computers are quite naïve and can only understand the programming language and algorithm specially designed for them.

Nature language:

“Hey computer, print ‘hi!’ for me!”

Computer language (in python):

```
print (“hi!”)
```

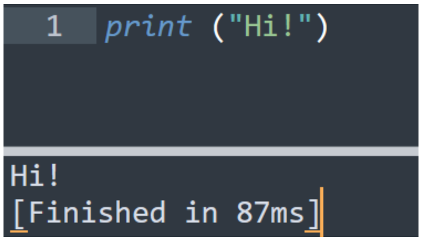
Nature language:

Check number x, and print it for me if it is positive.

Computer language (in python):

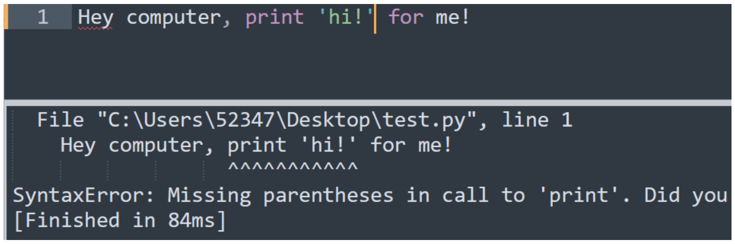
```
If x > 0:
        print (x)
```

As can be seen in the picture below, only the python language is executed, and natural language caused an error.



```
print ("Hi!")
```

```
Hi!
[Finished in 87ms]
```



```
Hey computer, print 'hi!' for me!
```

```
  File "C:\Users\52347\Desktop\test.py", line 1
    Hey computer, print 'hi!' for me!
                  ^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you
[Finished in 84ms]
```

If we want the computer to solve problems for us, it is necessary to understand how the computer thinks and transfer the idea into the language that the computer could understand. Computational thinking gives us a way to understand the issue at hand and the potential solutions necessary.

1.2. What is an algorithm? Be detailed.

An algorithm is a collection of instructions for resolving a problem or carrying out a task. Almost all aspects of information technology employ algorithms extensively to carry out their operations in hardware or software routines. In short, an **algorithm is a finite, unambiguous, and step-by-step process to transfer an input into an output**, with a clear beginning and end.

An initial input and a list of instructions are used by algorithms. The input, which may be expressed as either words or numbers, is the initial information sent to an algorithm to make further operations and judgments.

The input data is subjected to a series of instructions, or calculations, which may involve mathematical operations and judgment calls. The final step of an algorithm is called the output, and the output information of the algorithm is given to the whom/what needs it.

For a very simple example of how to make mapo tofu:

- Heat a pan on a stovetop.
- Add ground pork. Cook until it turns light brown.
- Pour sesame oil and add ginger, garlic, half of the scallions, and chopped chilies. Stir fry for 2 minutes.
- Stir in the chili bean sauce and fermented soybeans. Pour 1 1/2 cups of chicken broth. Let boil. Cover and cook on medium heat for 15 minutes.

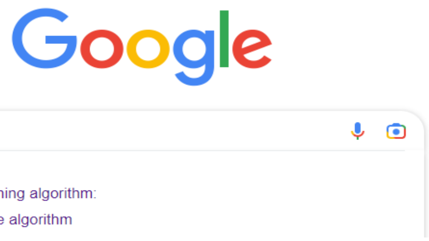
- Add tofu. Cook until sauce reduces to half.
- Combine cornstarch and 1/2 cup chicken broth. Stir. Pour the mixture into the wok. Cook until the texture thickens.
- Transfer to a serving bowl. Top with remaining scallions.
- Enjoy!

Source: https://thewoksoflife.com/ma-po-tofu-real-deal/

At first sight, a recipe seems to have no relation with the algorithm or computer science, however, when we see the definition of algorithm again and take the ingredients as input, we will find a recipe is a finite and step-by-step process, and the recipe will turn the ingredients into a delicious dish, which could be considered as an output, very clear start and end

Some other examples of algorithm application:

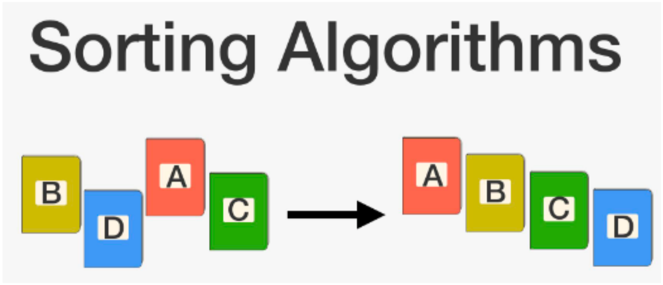
Search engine algorithm: This algorithm give advertisement or web page ranking based on the user's search history, location, and other collected data.



Google

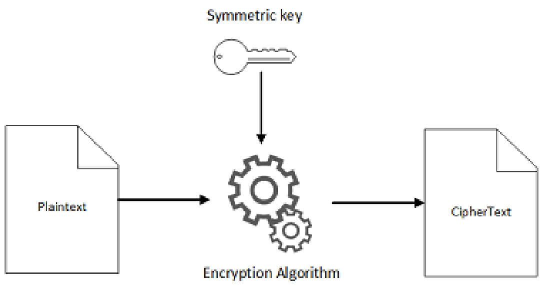
algorithm
machine learning algorithm
search engine algorithm

Sorting algorithm: Sorting algorithm can be used to sort data in a certain pattern.



Source: https://brilliant.org/wiki/sorting-algorithms/

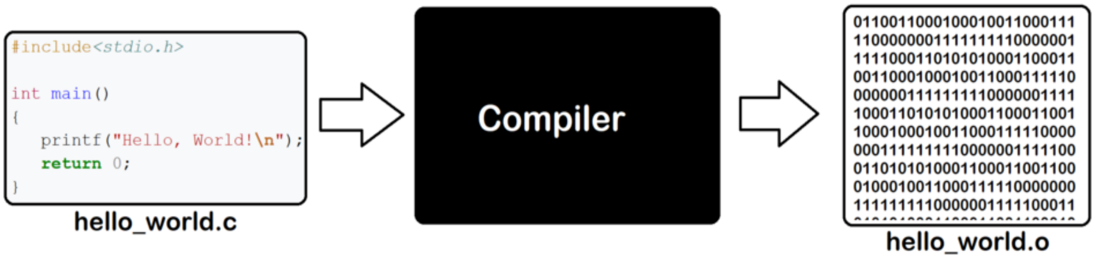
Encryption algorithm: Sophisticated algorithm to encode data and enforce security.



Source: https://docs.aws.amazon.com/crypto/latest/userguide/concepts-algorithms.html

1.3. What are the three ways to compile described in class and what are the pros/cons of each?

Programming languages such as Java, python are higher-level languages, these kinds of languages can not be read by a computer directly, and the higher-level languages need a compiler to transfer into a low-level language that can be used by the computer hardware, and this process is called compile.



```c
#include<stdio.h>

int main()
{
   printf("Hello, World!\n");
   return 0;
}
```

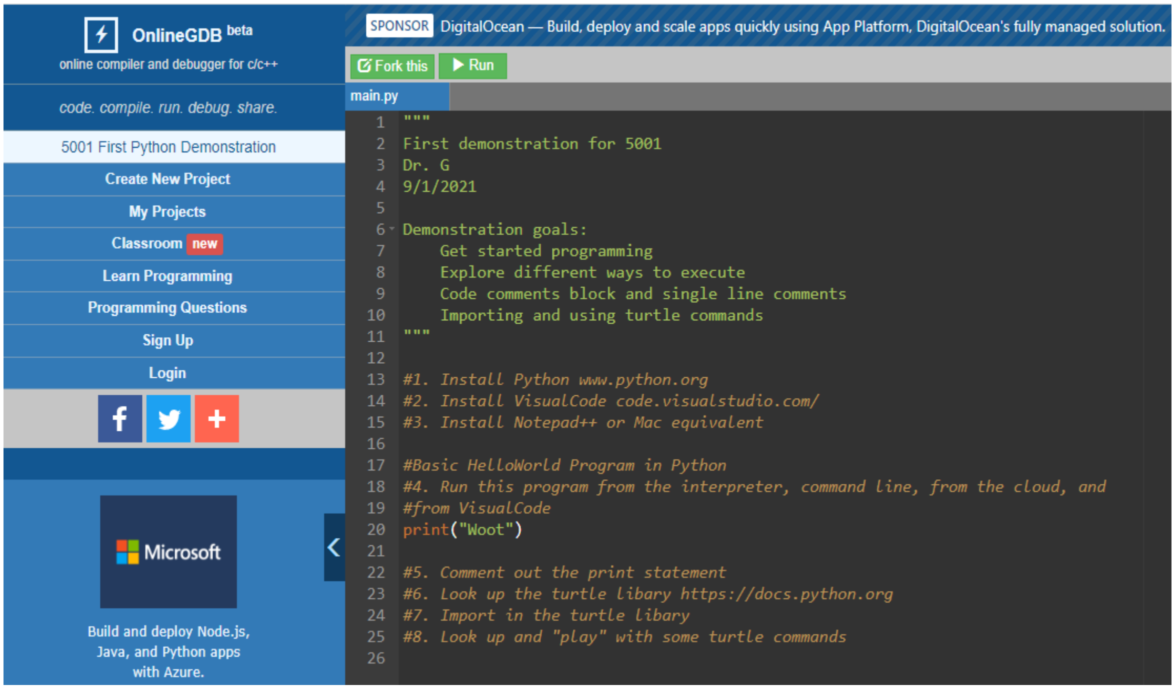
```
0110011000100010011000111
1100000001111111110000001
1111000110101010001100011
0011000100010011000111110
0000001111111110000001111
1000110101010001100011001
1000100010011000111110000
0001111111110000001111100
0110101010001100011001100
0100010011000111110000000
1111111110000001111100011
```

Source: https://hpc-wiki.info/hpc/Compiler

In conclusion, programs are executed by compilers and transformed from text format to executable format. There are three kinds of commonly used ways of compilers.

1.3.1. Cloud compiler

The online compiler uses the idea of cloud computing to help user compile their programs online.



OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.
5001 First Python Demonstration
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Sign Up
Login
Microsoft
Build and deploy Node.js, Java, and Python apps with Azure.
SPONSOR DigitalOcean — Build, deploy and scale apps quickly using App Platform, DigitalOcean's fully managed solution.
Fork this
Run
main.py

```python
"""
First demonstration for 5001
Dr. G
9/1/2021

Demonstration goals:
    Get started programming
    Explore different ways to execute
    Code comments block and single line comments
    Importing and using turtle commands
"""

#1. Install Python www.python.org
#2. Install VisualCode code.visualstudio.com/
#3. Install Notepad++ or Mac equivalent

#Basic HelloWorld Program in Python
#4. Run this program from the interpreter, command line, from the cloud, and
#from VisualCode
print("Woot")

#5. Comment out the print statement
#6. Look up the turtle libary https://docs.python.org
#7. Import in the turtle libary
#8. Look up and "play" with some turtle commands

```

GDB cloud compiler

Pros:

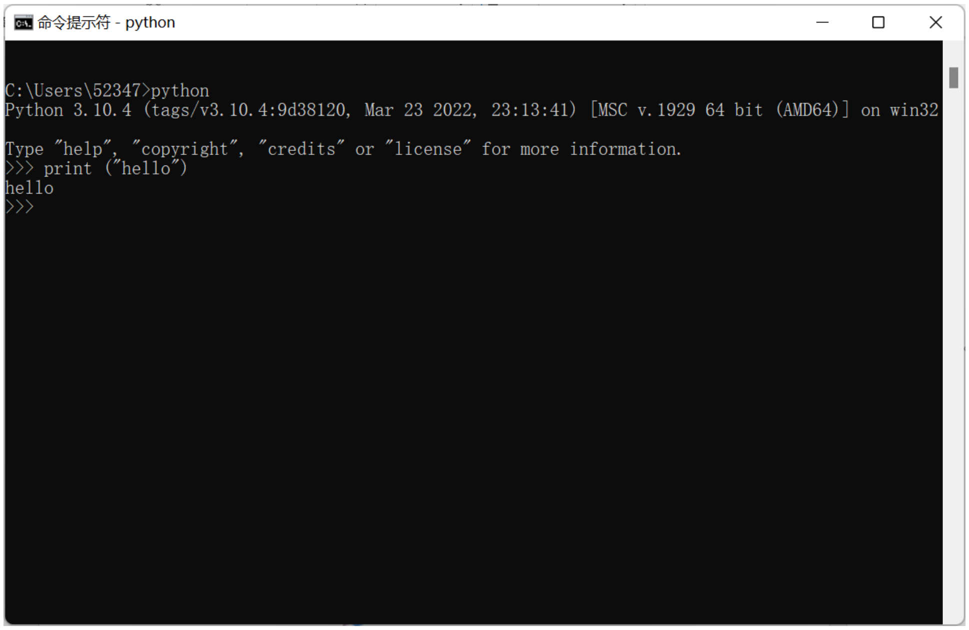
- No complicated setups, with automatic software integration, open the web page and you can use it
- Access from anywhere with the network, can use different devices
- Real-time editing, good for teamwork in different locations
- Break hardware limitations, the compiler does not rely on your hardware, but using a bigger capacity cloud server can break the limit
- Other plugins such as debuggers and cloud storage and data restore

Cons:

- Need to be online and rely on the bandwidth
- Performance can vary depending on the cloud environment
- Less customize options

1.3.2. Command line

The command-line compiler is executed from the command-line interface (CLI).



```
C:\Users\52347>python
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.
>>> print ("hello")
hello
>>>
```

Command Line Interface

Pros

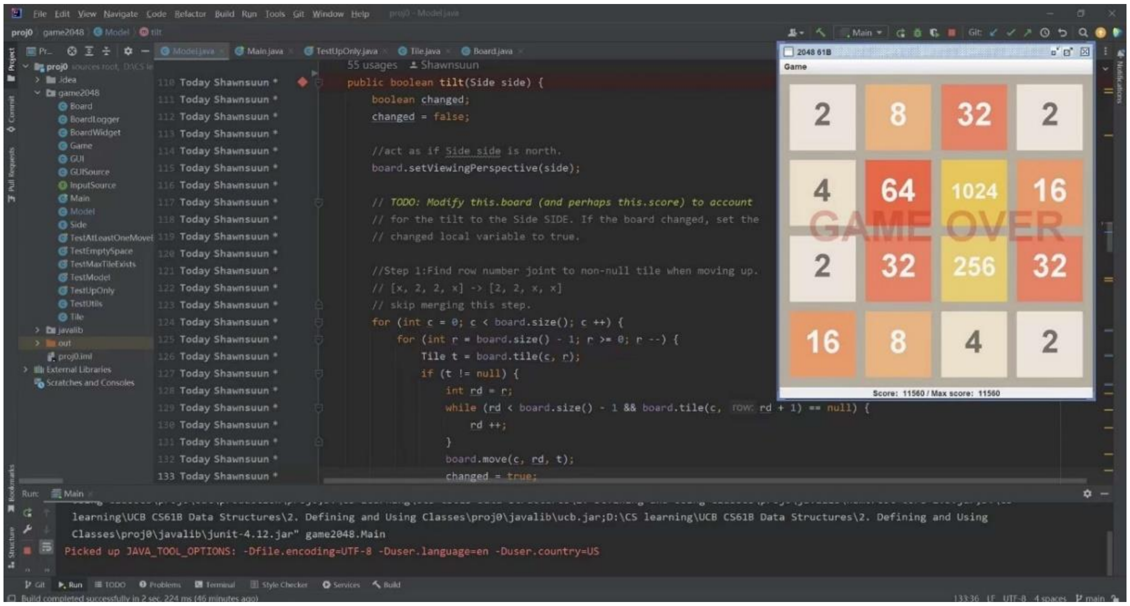
- Faster and more efficient than other ways of compiling
- Require fewer resources (CPU, RAM, high-resolution monitor)
- Do not need to run on windows

Cons:

- As most users are used to GUI (Graphical User Interface) such as programs running in windows instead of CLI, the CLI compiler is tougher to use for a user who is not familiar with CLI
- Commands have to be very precise, with no auto-complete or corrections and other plugins.
- Plain visual effect

1.3.3. IDE

Integrated Development Environment, or IDE for short, is a desktop software program that you may setup and use to develop programs. they are upgraded applications that can also have additional capabilities that help to improve the procedure related to software development, such as Visual Studio, IntelliJ IDEA, and pyCharm, you may probably hear of them.



```
public boolean tilt(Side side) {
    boolean changed;
    changed = false;

    //act as if Side side is north.
    board.setViewingPerspective(side);

    // TODO: Modify this.board (and perhaps this.score) to account
    // for the tilt to the Side SIDE. If the board changed, set the
    // changed local variable to true.

    //Step 1:Find row number joint to non-null tile when moving up.
    // [x, 2, 2, x] -> [2, 2, x, x]
    // skip merging this step.
    for (int c = 0; c < board.size(); c ++) {
        for (int r = board.size() - 1; r >= 0; r --) {
            Tile t = board.tile(c, r);
            if (t != null) {
                int rd = r;
                while (rd < board.size() - 1 && board.tile(c, row: rd + 1) == null) {
                    rd ++;
                }
                board.move(c, rd, t);
                changed = true;
```

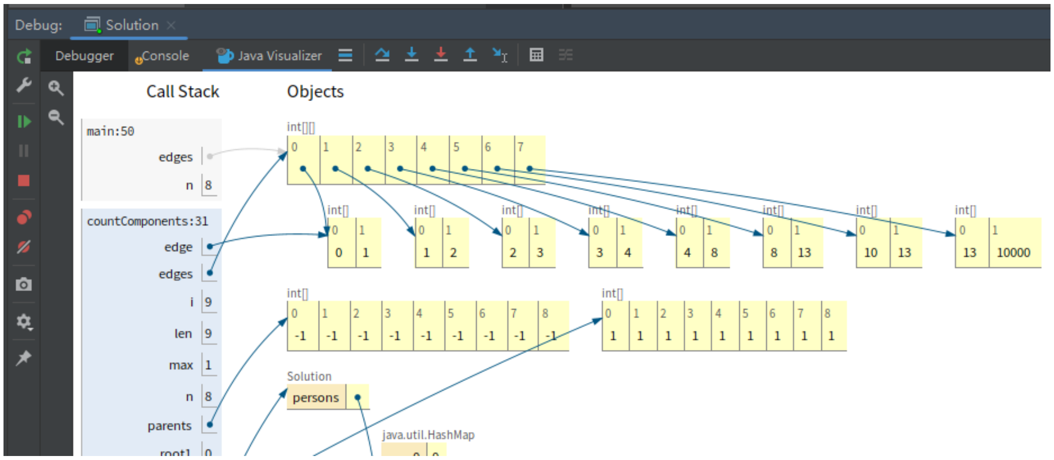
```
learning\UCB CS61B Data Structures\2. Defining and Using Classes\proj0\javalib\ucb.jar;D:\CS learning\UCB CS61B Data Structures\2. Defining and Using
Classes\proj0\javalib\junit-4.12.jar" game2048.Main
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8 -Duser.language=en -Duser.country=US
```

File Edit View Navigate Code Refactor Build Run Tools Git Window Help
proj0 - Model.java
Model.java
Main.java
TestUpOnly.java
Tile.java
Board.java
55 usages Shawnsuun
2048 61B
Game
2 8 32 2
4 64 1024 16
GAME OVER
2 32 256 32
16 8 4 2
Score: 11560 / Max score: 11560
Git Run TODO Problems Terminal Style Checker Services Build
Build completed successfully in 2 sec, 224 ms (46 minutes ago)

IntelliJ IDE

Pros:

- A lot of useful plugins, make it easier to program, such as syntax highlighter, auto-complete, debugger (with visualizer), built-in function names



```
main:50
  edges
  n      8

countComponents:31
  edge
  edges
  i      9
  len    9
  max    1
  n      8
  parents
  root1  0
```

```
int[][]
0 1 2 3 4 5 6 7

int[]    int[]    int[]    int[]    int[]    int[]    int[]     int[]
0  1     0  1     0  1     0  1     0  1     0  1     0  1      0  1
0  1     1  2     2  3     3  4     4  8     8  13    10  13    13  10000

int[]
0  1  2  3  4  5  6  7  8
-1 -1 -1 -1 -1 -1 -1 -1 -1

int[]
0  1  2  3  4  5  6  7  8
1  1  1  1  1  1  1  1  1

Solution
persons

java.util.HashMap
```

Java visualizer and debugger in intelliJ IDEA

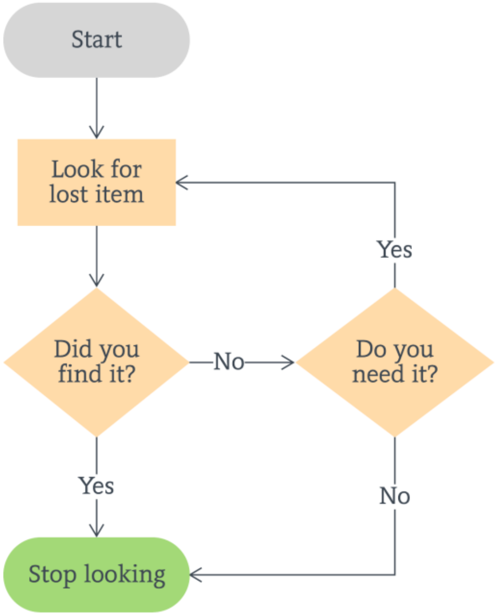
- Graphical User Interface with nice visual effects, customize themes
- IDEs are designed for software development, which can be comfortable for developers to use if the settings cater to the need of developers.

Cons:

- Very complicated setups and environment settings
- Not friendly to beginners, IDEs are powerful but also challenging to learn how to maximize their value
- Rely on the hardware and more resources are needed compared to command line compiling

1.4. Explain the different symbols of flowcharting and how they related to computer science.

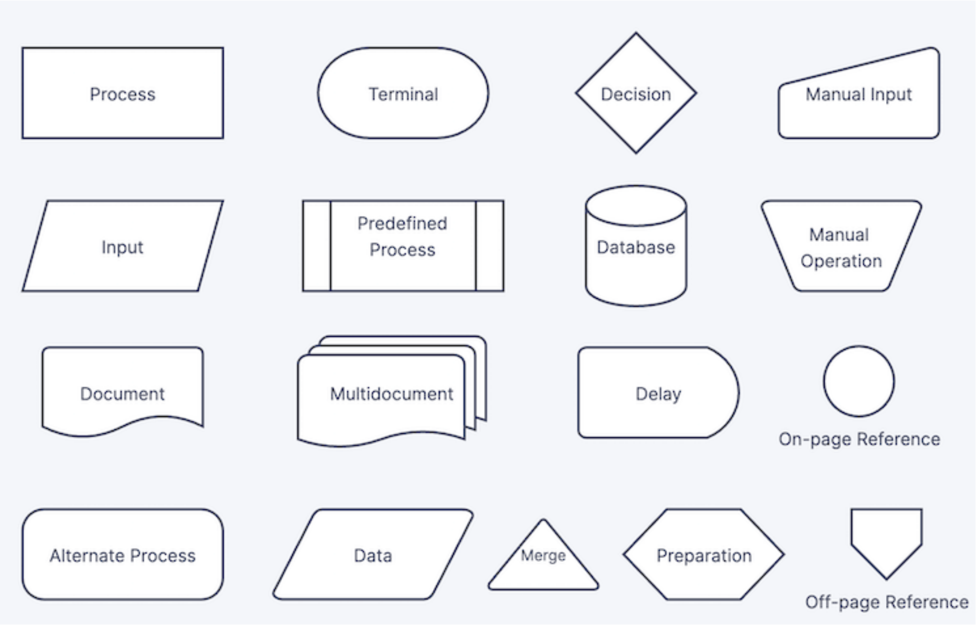
A flowchart is a diagram that shows how a system, computer algorithm, or process works. For example, the picture below is a flowchart diagram showing looking for a lost item.



Source: https://www.lucidchart.com/blog/flowchart-templates

1.4.1. Symbols of flowchart

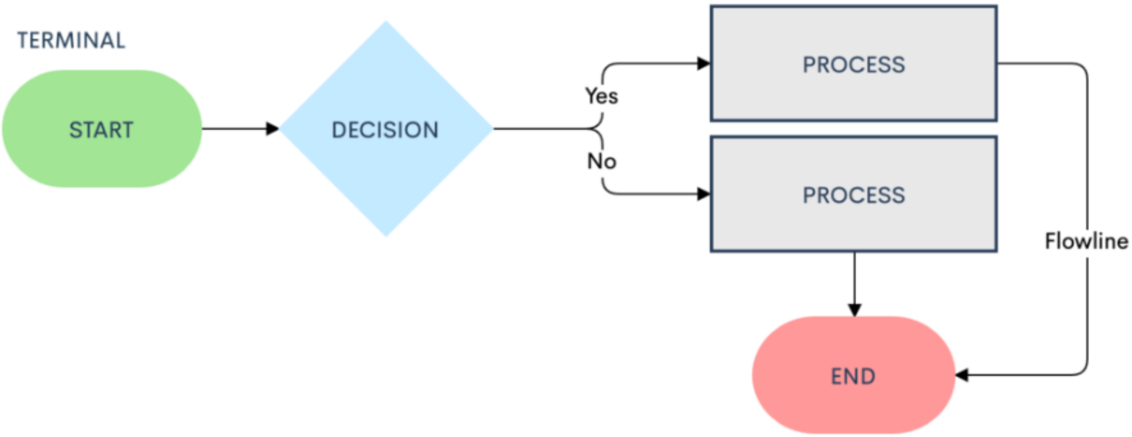
As can be seen in the flowchart above, in the diagram there are different symbols, which represent different meanings that can have to be used precisely in a flowchart to make it logically rigorous.



Flowchart Symbols

Source: https://www.zenflowchart.com/flowchart-symbols

Common use flowchart symbols



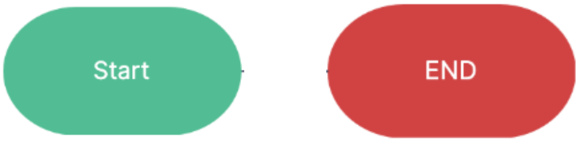
Top 4 Flowchart Symbols in Use

Source: https://www.zenflowchart.com/flowchart-symbols

Arrow(Flowline): Show directions and connections of different symbols.

Flowline

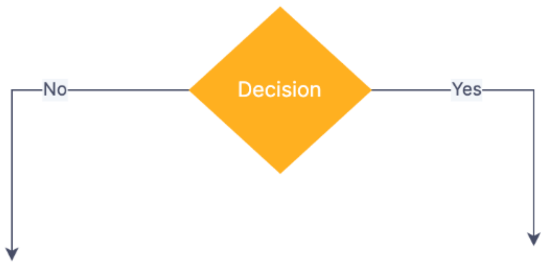
Terminal: The beginning or finish of the flowchart is indicated by the terminator symbol. The term "Start" or "End" is typically present.



Process: Represent a single action in a flow chart.

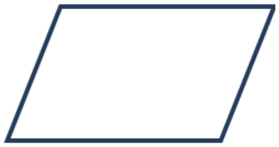
Process

Decision: To make decisions from different choices. Different decisions are represented by lines that emanate from various spots on the diamond.



Decision Symbol

Input / Output: Somewhere you can enter information inside the flowchart(input), or get information from the flowchart(output).



Input / Output

1.4.2. Flowchart, algorithm, and computer science

Flowcharts can provide a visual depiction of processes, and are helpful when designing computer programs or algorithms since they can present the invisible algorithm into a visible diagram, which is structure clear and easier to understand.

A flowchart can be a schematic that depicts the algorithm and its stages in visual form, which can be used to outline the process under the hood of an algorithm. Before writing a program, the flowchart can be used to give an outline in advance and achieve it step by step to make programming and presentation easier.

1.5. Practical

Assume someone has never programmed in any language before. Create a short tutorial introducing them to what an algorithm is and how to flowchart it. Include a problem/solution.

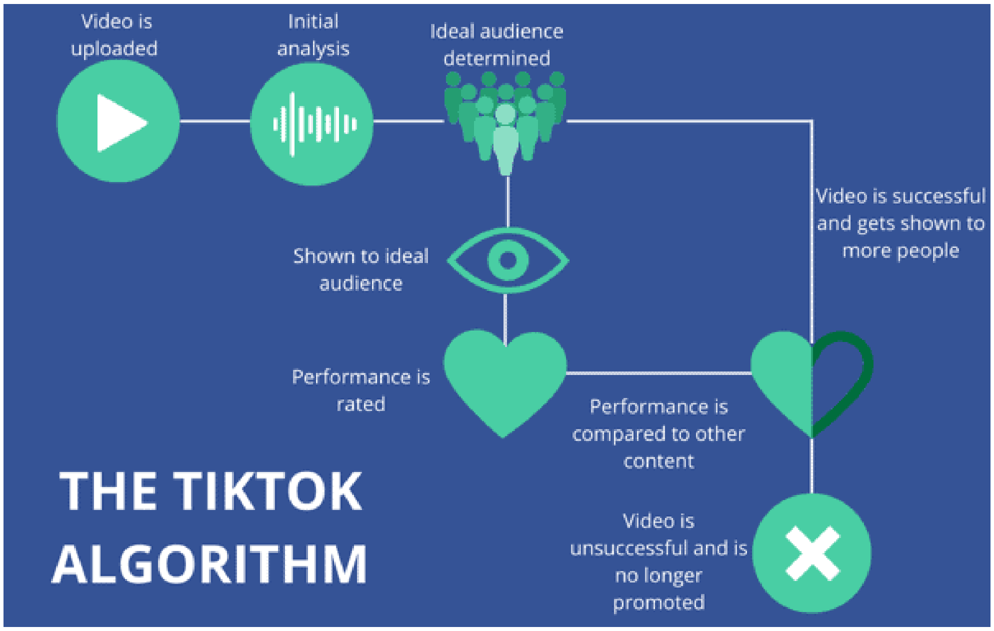
1.5.1. A simple algorithm example

An **algorithm is a finite, unambiguous, and step-by-step process** to transfer an input into an output, with a clear beginning and end.

TikTok is famous for its recommendation algorithm, the reason why TikTok is so easy to make users addicted. The process is as follows.

- Uploading video
- A series of manual and computer analysis
- Based on the data of viewing history the ideal audience is determined

- The video will be shown to the ideal audience
- Rate the performance and compare to other contents
    - ➢ Good performance: Recommend the video to more ideal audience, become a trend
    - ➢ Bad performance: No longer promoted


Video is uploaded
Initial analysis
Ideal audience determined
Shown to ideal audience
Performance is rated
Performance is compared to other content
Video is successful and gets shown to more people
Video is unsuccessful and is no longer promoted
THE TIKTOK ALGORITHM

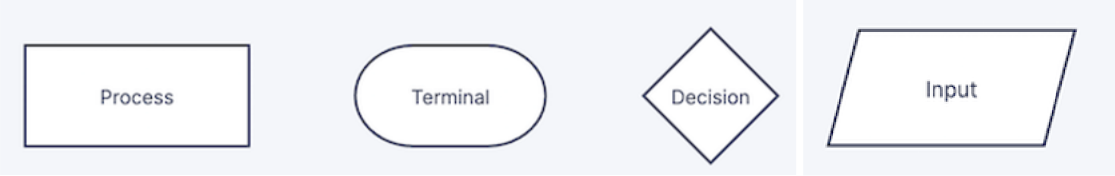
Source：https://socialmarketingwriting.com/get-more-followers-fans-tiktok/tiktok-algorithm-infographic/

In this algorithm, the video is an input, and after the step-by-step TikTok algorithm, it could be possible to a good consequence as an output, recommend the video to more users, or no longer be promoted as another possible bad output.

Of course, this algorithm is much more complicated than my description, which is a patent of TikTok, not that easy.

1.5.2. Algorithm to flow chart

Now we are trying to transfer the algorithm into a simple flow chart.

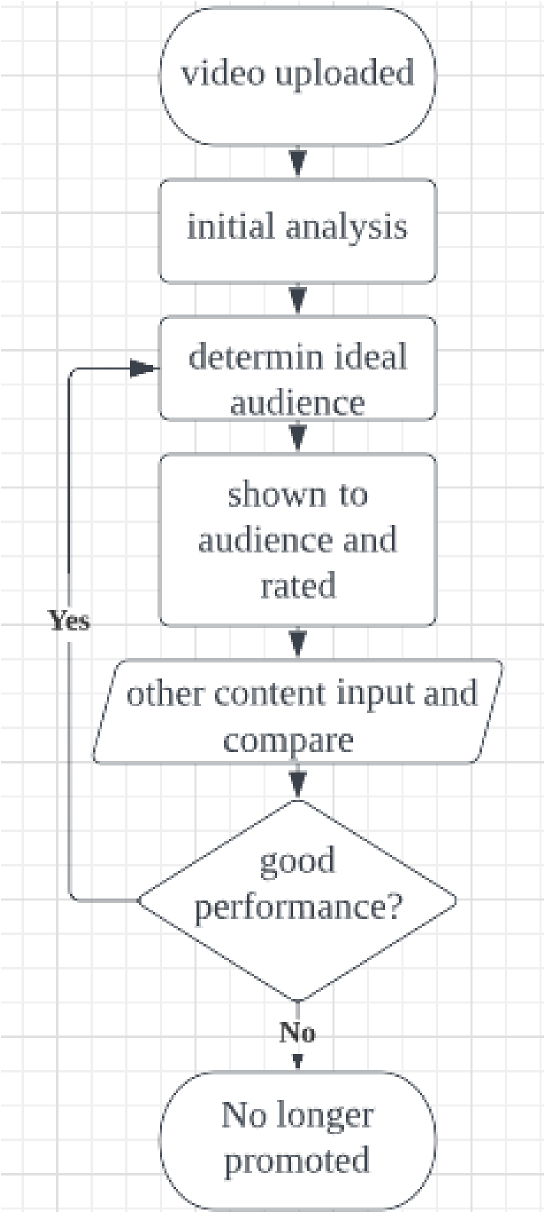


Based on the description:

- The start of the algorithm are represented by the terminal symbol, starting with the video uploading.

- Processes such as analysis, determining the audience, and showing to the audience are represented by a process symbol.
- There is an input of other content to compare performance, this process is represented by an input symbol.
- The result is based on a good or bad performance, the decision symbol is used to lead to different results.
- The algorithm ends when the performance is bad and no longer promoted.

These symbols are connected by arrows based on the logic sequence, a flowchart is formed as follows.



1.5.3. Problem

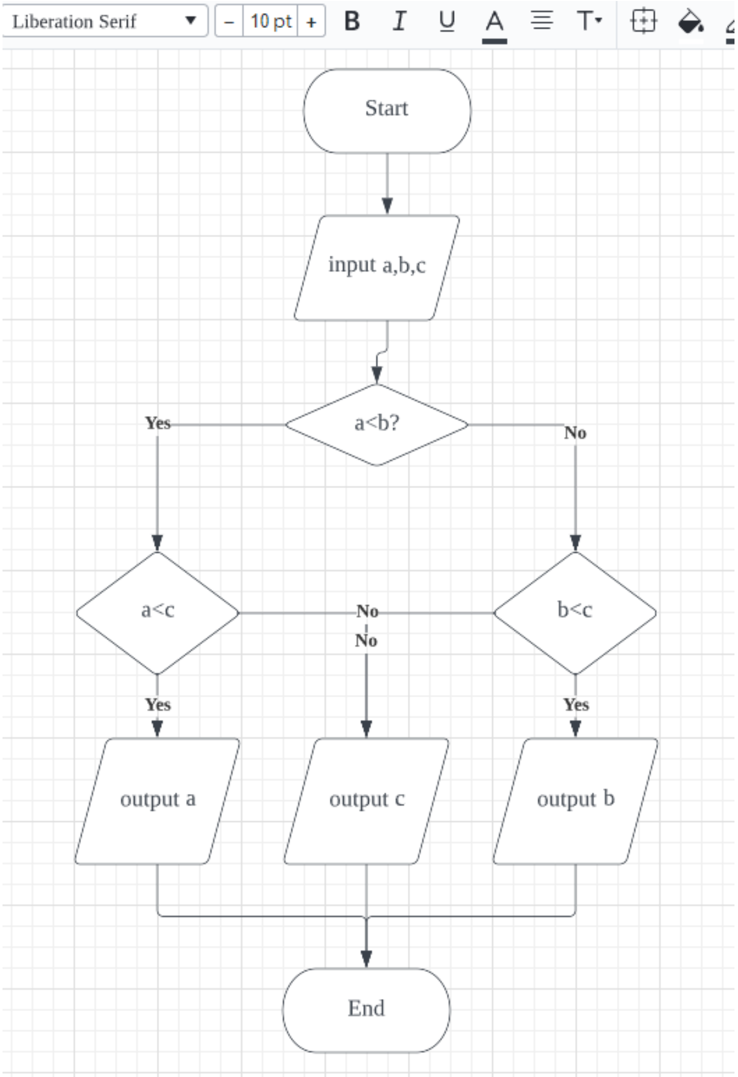
Create an algorithm to get the minimum numbers in three different numbers and draw its flowchart, for example, input four numbers a = 1, b =4, c=-1, the output of the algorithm should output -1, which is the smallest number among a, b, c.

1.5.4. Solution

The comparison of three different numbers can be achieved by following steps, the main idea is to compare two of them, and the use the smaller number compare with the rest one and output the smaller number.

```
Step 1: input numbers a, b ,c.
Step 2: compare a and b,
Step 3:
   Option1: if a <b, compare a and c
   Option2: if a >b, compare b and c
Step 4: if a < c, compare a and c
   Option1:
      if a < c, output a and end
      if a > c, output c and end
   Option2:
      if b < c, output b and end
      if b > c, output c and end
```

Based on the above steps, the flow chart can be drawn as follows.


Liberation Serif
10 pt
Start
input a,b,c
a<b?
Yes
No
a<c
No
No
b<c
Yes
Yes
output a
output c
output b
End

2. Chapter 2 : Variables, Arithmetic Operations and Conditionals

2.1. What is a variable in Python, what are the variable types, and what is happening in memory storage when a variable is created?

Python variables are locations of memory where values can be kept, as a reference or pointer to an object.

A variable need to have its name, so that if we want to call the variable out, we can use "=" to assign a value to a variable, such as:

```
a = 100
b = 100.0
c = "100.0"
d = True
```

a,b,c,d are variable names and here we assign different kinds of values for them.

2.1.1. Variable types

The value that a variable store dictates the type of that variable. Four typical kinds of variables are int, float, string, and Boolean.

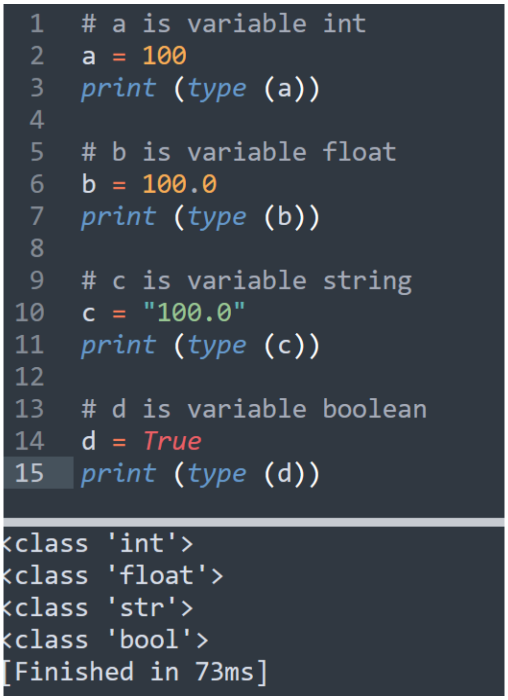
- int

  Int is one kind of numerical data, a short form of integers. For example, whole numbers in decimals such as 100, 0, -1 are ints.
- float

  Float is one kind of numerical data. A float is a number with a decimal place since it is a floating-point number. For example, 1.5, -10.34 are floats.
- String

  Short form of strings of characters, generally any group of characters can be interpreted literally can be called string. For example, "hello", "100", "asd346sadf" are strings.
- Boolean

  Boolean is a logical data type, the only two possible values for a Boolean data type are "True" and "False."

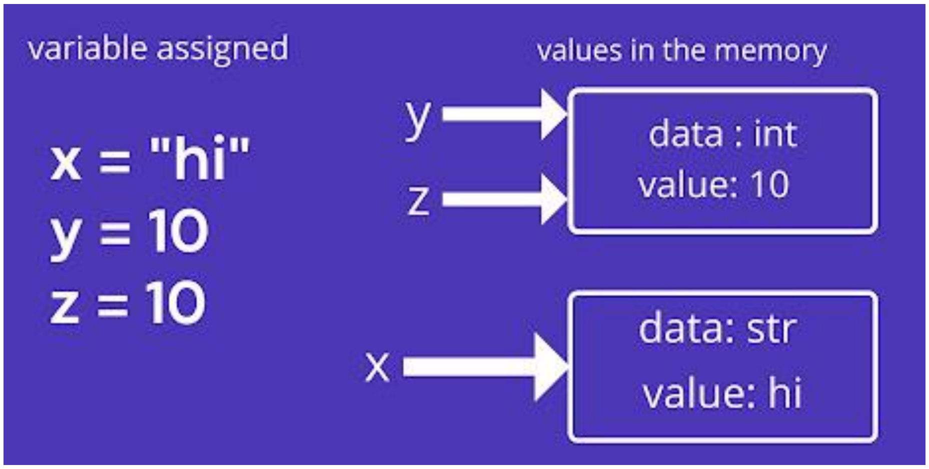


```
# a is variable int
a = 100
print (type (a))

# b is variable float
b = 100.0
print (type (b))

# c is variable string
c = "100.0"
print (type (c))

# d is variable boolean
d = True
print (type (d))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
[Finished in 73ms]
```

2.1.2. Memory storage of variable

When you create a variable, a memory location is assigned for the value of the variable, each variable is stored in a unique location memory.



```
x = "hi"
y = 10
z = 10
```

Source: https://www.pycodemates.com/2022/01/what-are-variables-in-python.html

As shown in the picture above, x, y, and z are variable names, after the assignment of the variables, the type of data is checked and their values are stored in the memory.

2.2. Explain arithmetic expressions in Python. Make sure to include all possibilities, syntax, order of operations, and mismatch issues.

2.2.1. Arithmetic operators

Addition, subtraction, multiplication, division, modulus, exponentiation, and floor division are the fundamental arithmetic operations in Python.

Operator	Example expression	Operation
+	a+b	Addition
-	a-b	Subtraction
*	a*b	multiplication
/	a/b	Division
%	a%b	Modulus
**	a**b	Exponentiation
//	a//b	Floor division

2.2.2. Assignment operators

Variables can be assigned to a certain value by using assignment operators. In the following examples, after different kinds of arithmetic operations, the result is assigned to a.

Operator	Example expression	Another expression	Operation
+	a+=b	a=a+b	Addition
-	a-=b	a=a-b	Subtraction
*	a*=b	a=a*b	multiplication
/	a/=b	a=a/b	Division
%	a%=b	a=a%b	Modulus
**	a**=b	a=a**b	Exponentiation
//	a//=b	a=a//b	Floor division

2.2.3. Arithmetic expressions

The syntax of arithmetic operations is about combining numbers, operators, and variables in the right order.

Input:

```
a = 10
b = 3
print (a + b)
print (a - b)
print (a * b)
print (a / b)
print (a % b)
print (a ** b)
print (a // b)
```

Output:

```
13
7
30
3.3333333333333335
1
1000
3
```

2.2.4. Order of arithmetic operations

Python will always evaluate the arithmetic operators first, in the following order

- Exponentiation ** (high)
- Multiplication/division *, /, //, %
- addition/subtraction. +, -
- relational operators >=, ==, <=, >, <
- logical operators not, and, or(low)

2.2.5. Mismatch issues

Arithmetic operations are generally between the same type of variables, however, the operations may be not suitable for different types of variables.

```
a = 10
b = 3
c = "hola231"
d = "3"
e = True
f = False
g = 3.5

print(a+c)
print (str(a) + c)
print(b*c)
print(g*c)
print(a+int(d))
print(int(c)+f)
print(a+e+f)
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
10hola231
hola231hola231hola231
TypeError: can't multiply sequence by non-int of type 'float'
13
ValueError: invalid literal for int() with base 10: 'hola231'
11
```

print(a+c): a is an int and c is a string, these two kinds of variables cannot be added directly, and this mismatch will cause an error.

print (str(a) + c): if we cast a into a string, str(a) and c are both strings, they can be added by a '+' operator, the result will be a string combined by the two strings.

print(b*c): b is an int and c is a string, in python int * str means to print the string multiple times according to the int value.

print(g*c): g is a float and c is a string, a string can not print in the times of a float value, and this will cause an error.

print(a+int(d)): a is an int and string d is cast to an int, two ints can add together and will give a result of the value of the sum of two ints.

print(int(c)+f): c is a string but it can not be cast to an int since it contains letters, and this would cause an error.

print(a+e+f): a is an int and e,f are booleans, boolean can be used as an int in arithmetic operations, in which True is 1 and False is 0, the result will be the sum of three int.

2.3. Explain the selection syntax within python including single and multiple pathways.

2.3.1. If statement

In Python, the if statement is used to test a condition and determine whether or not to execute statements based on the outcome of the test.

For example:

If the condition is met, implement statements (statment1, statment2…), the syntax is as follows.

```
if condition:
        statment1
        statment2
        …
```

2.3.2. if -else statement

In Python, the if-else statement is used to test a condition and determine which of two blocks of statements should be executed based on the outcome.

For example:

If the condition is met, implement statements (statment1, statment2…)

If the condition is not met, implement other statements (statment3, statment4…)

the syntax is as follows.

```
if condition:
        statment1
        statment2
        …
else:
        statment3
        statment4
        …
        …
```

2.3.3. If-elif statement

Python's elif statement is used to test several conditions.

For example:

If condition1 is met, implement statements (statment1, statment2…)

If condition1 is not met, condition2 is met, implement statements (statment3, statment4…)

If condition1,2 are not met, condition3 is met, implement statements (statment5, statment6…)

If conditions 1,2,3 are not met, implement other statements (statment7, statment8…)

the syntax is as follow.

```
if condition1:
        statment1
        statment2
        …
elif condition2:
        statment3
        statment4
        …
elif condition3:
        statment5
        statment6
        …
else:
        statment7
        statment8
        …
```

2.4. Boolean expressions are likely a new concept to a lot of you. Explain what these are, how they work, when they would be used, syntax, and anything else that someone new to boolean variables would need to know.

Boolean is a logical data type, the only two possible values for a Boolean data type are "True" and "False."

2.4.1. Comparing two values

Python gives a Boolean response after comparing two values:

For Example:

Input:

```
print(10 < 3)
print(3 == 3)
```

Result:

```
False
True
```

2.4.2. Checking empty value

Python gives a Boolean response after checking the value of a variable is empty:

For Example:

Input:

```
print(bool(a))
print(bool(""))
print(bool([]))
print(bool(0))
print(bool(None))
print(bool(False))
```

Result:

```
True
False
False
False
False
False
```

2.4.3. Conditional statement

A Boolean can be used in an if statement as a condition, if the Boolean is True, means the condition is met.

For Example:

Input:

```
flag = True
if flag:
        print(“hi”)

flag = False
if not flag:
        print("hello")
```

Output:

```
hi
hello
```

2.4.4. Logical operations

Booleans can be used in logical operations, connected by logical operators.

For Example:

Input:

```
flag1 = True
flag2 = False
print (flag1 and flag2)
print (flag1 or flag2)
print (not flag2)
```

Output:

```
False
True
True
```

2.4.5. Arithmetic operations

Booleans can be used as numbers and participate in arithmetic operations, where True = 1 and False = 0

For Example:

Input:

```
flag1 = True
flag2 = False
print (flag1 + flag2)
print (flag2 - flag1)
print (int(not flag2))
```

Output:

```
1
-1
1
```

2.5. Practical

Create a program assignment for someone who is just learning about the concepts in this chapter. Give them learning goals, objectives, etc. Then, create a solution for that assignment. Make sure you are providing them a way to exercise all of the concepts.

2.5.1. Learning goals

- Arithmetic Operations and Correct usage of variables
- Boolean usage
- Selection syntax usage
- Catch user input
- Create a simple menu-driven program

2.5.2. Objectives

Practice problem 1: Four arithmetic calculator

Let the user input two numbers and an arithmetic operator from +.-.*.and /, and print the result for the user

```
Enter first number:5.2
Enter symbol(+,-,*,/):+
Enter second number:6.7
11.9
```

Practice problem 2: Tan to Sin calculator

Create a Tan value of an angle and transfer it to Sin value of this angle and print it out.

For example:

```
Enter tan value:1
The sin is 0.7071067811865475.
```

Practice problem 3: Weather and plan

Use boolean and selection syntax to create a menu-driven program, if the user input is ‘sunny’, print ‘I will go out and play.’, if the user input is ‘rainy’, print ‘I will stay at home.’, quit if user input ‘q’, print ‘I will stay at home.’ for other input.

Here is an example:

```
Enter Weather(sunny/rainy), enter q to quit:sunny
I will go out and play.
Enter Weather(sunny/rainy), enter q to quit:rainy
I will stay at home.
Enter Weather(sunny/rainy), enter q to quit:cloudy
I have not decided yet.
Enter Weather(sunny/rainy), enter q to quit:q
```

Practice problem 4 Text combiner:

Let the user enter a series of texts multiple times, and combine the text together, when quit the program, print out the combined message.

Here is an example:

```
Enter your message, enter 'q' to print the message:hello
Enter your message, enter 'q' to print the message:world
Enter your message, enter 'q' to print the message:q
hello world
```

2.5.3. Solution

Practice problem 1 solution solutions:

```
text1 = input("Enter first number:")
text2 = input("Enter symbol(+,-,*,/):")
text3 = input("Enter second number:")
if (text2 == "+"):
    print(float(text1) + float(text3))
elif (text2 == "-"):
    print(float(text1) - float(text3))
elif (text2 == "*"):
    print(float(text1) * float(text3))
elif (text2 == "/"):
    print(float(text1) / float(text3))
else:
    print("wrong input")
```

Practice problem 2 solutions:

```
tan = input("Enter tan value:")
sin = float(tan) / ((float(tan) ** 2 + float(1) ** 2) ** 0.5)
print(f"The sin is {sin}.")
```

Practice problem 3 solutions:

```
flag = True
while flag == True:
    text = input("Enter Weather(sunny/rainy), enter q to quit:")
    if (text == "sunny"):
        print("I will go out and play.")
    elif (text == "rainy"):
        print("I will stay at home.")
    elif (text == "q"):
        flag = False
    else:
        print("I have not decided yet.")
```

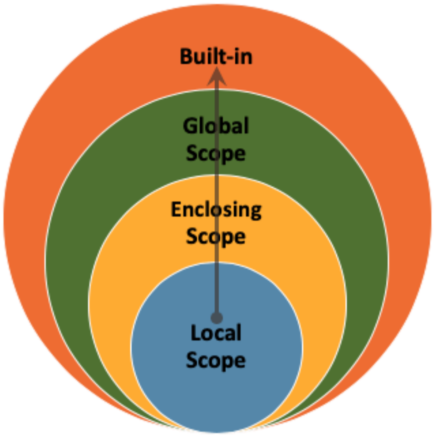
Practice problem 4 Text combiner:

```
flag = True
text = "
while flag == True:
    text_entered = input("Enter your message, enter 'q' to print the message:")
    if text_entered == 'q':
        flag = False
    else:
        text += text_entered
print (text)
```

3. Chapter 3 : Functions and Testing:

3.1. Describe the concept of scope and code blocks in Python. Examples are useful. Include all possible scopes.

In Python, the term scope of a variable refers to range of the script where it can be referenced.

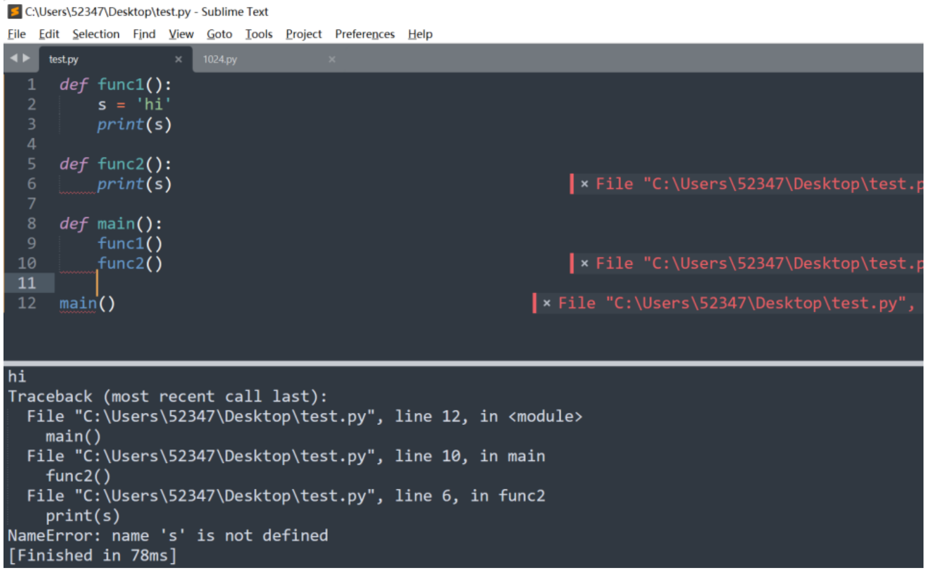


Source: https://www.datacamp.com/tutorial/scope-of-variables-python

Code block is made up of numerous statements that are meant to run when a particular condition is satisfied.

3.1.1. Local Scope

Local scope refers to local variables defined in the body of a function


C:\Users\52347\Desktop\test.py - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
test.py
1024.py

```
def func1():
    s = 'hi'
    print(s)

def func2():
    print(s)                                  × File "C:\Users\52347\Desktop\test.p

def main():
    func1()
    func2()                                   × File "C:\Users\52347\Desktop\test.p

main()                                       × File "C:\Users\52347\Desktop\test.py",
```

```
hi
Traceback (most recent call last):
  File "C:\Users\52347\Desktop\test.py", line 12, in <module>
    main()
  File "C:\Users\52347\Desktop\test.py", line 10, in main
    func2()
  File "C:\Users\52347\Desktop\test.py", line 6, in func2
    print(s)
NameError: name 's' is not defined
[Finished in 78ms]
```

As can be seen in picture above, s is a variable in the body of function func1(), and this local variable can not be accessed outside the function.

In func2(), variable s can not be accessed and will cause an error, saying:

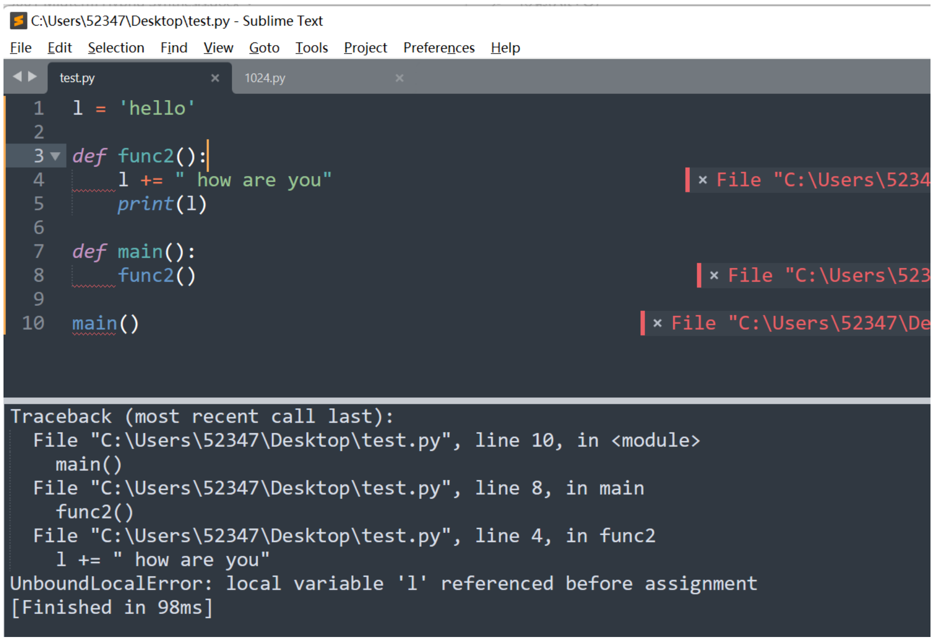
```
NameError: name 's' is not defined
```

3.1.2. Enclosing Scope

Enclosing scope refers to variables defined outside of the function

In the picture below, l is a variable in enclosing scope, however in func2(), the function tried to create another variable l, but l = ‘hello’ already and an error would be reported.

```
UnboundLocalError: local variable 'l' referenced before assignment
```


C:\Users\52347\Desktop\test.py - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
test.py
1024.py
× File "C:\Users\5234
× File "C:\Users\523
× File "C:\Users\52347\De

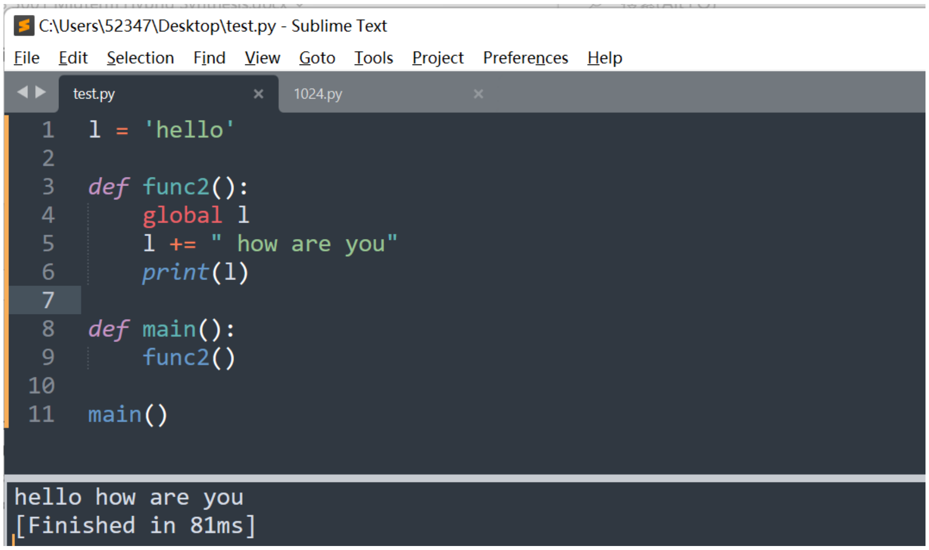
```
l = 'hello'

def func2():
    l += " how are you"
    print(l)

def main():
    func2()

main()
```

```
Traceback (most recent call last):
  File "C:\Users\52347\Desktop\test.py", line 10, in <module>
    main()
  File "C:\Users\52347\Desktop\test.py", line 8, in main
    func2()
  File "C:\Users\52347\Desktop\test.py", line 4, in func2
    l += " how are you"
UnboundLocalError: local variable 'l' referenced before assignment
[Finished in 98ms]
```

3.1.3. Global Scope

Global scope refers to variables that can be seen and reached anywhere in the program.

As can be seen in picture below, global keyword is used to set scope of variable l, and l can be referenced anywhere in the program, force the use of the l = ‘hello’ instead of creating another variable l.


C:\Users\52347\Desktop\test.py - Sublime Text
File Edit Selection Find View Goto Tools Project Preferences Help
test.py
1024.py

```python
l = 'hello'

def func2():
    global l
    l += " how are you"
    print(l)

def main():
    func2()

main()
```

```
hello how are you
[Finished in 81ms]
```

3.1.4. Built-in Scope

Built-in Scope is the widest scope, refer to built-in names included in Python, which can use directly.

For example, print(), True/False can be used anywhere and anytime in a python program.

3.2. Discuss functions in Python. Include why they are needed, the syntax, calling, organization, and anything else your reader would need to know to understand functions.

3.2.1. Definition of function

A function is a code block that only executes when called, the code is related to achieving a specific task.

3.2.2. Syntax of function

Function definition syntax

- Begins with the keyword def in the function header,
- Followed by a unique function name, parameters inside parentheses
- The function header ends with a colon
- The function body includes a series of statements
- A return statement is optional

```
def function_name(parameter1…):
        statment1
        statment2
        …
        return …
```

Call a function syntax

```
function_name(parameter1…)
```

In python, the function should be called only after it is defined.

3.2.3. Benefits of function

Functions can separate complex codes into small parts, each function executes a specific task, which could make code clear, modular, and manageable.

Functions can be reused, and this could reduce repetitive codes and increase teamwork efficiency.

3.3. Discuss, with examples, sending and receiving information from a function.

3.3.1. Send information from a function

In python, except for information inside the function body, Multiple arguments can be sent to a function, including objects, variables, and functions(which are also objects).

In the following example, msg ‘hello’ is sent to the function by parameter x, when executing the function, the input information is printed.

Example:

```
msg = ‘hello’
def greetings(x):
         print(x)
greetings(‘hello’)
```

Output:

```
hello
```

3.3.2. Receiving information from a function

Users receive information from a function by the return statement.

```
x = 6
def plus_five(x):
    y =  x + 5
    return y
print(plus_five(x))
```

```
11
```

In the code above, we give x=5 into the function, and the function gives us information by a return statement, in this function return value is int 11, and when we print the return value out, the answer is 11.

3.4. How do we and why do we separate our code into separate files?

3.4.1. How to separate code into separate files

When your code grows too large, you may separate the code into separate files, each file concentrating on a certain aspect of the whole program.

Suppose that a program is separated into two files, main.py and library.py, in order to make the separate files have the same function as the original file. We need to import modules in one file to another and access components of the imported module using import syntax.

There are 4 common ways of importing modules (the statement after the module is omitted).

(1) Import library.py to main.py, need to mention the syntax as follows.

```
import library
```

If calling a certain module of library in main.py, need to mention the imported module name, and use the syntax:

```
library.module…
```

(2) Import library.py to main.py, give the imported module a different name as lib, syntax as follow.

```
import library as lib
```

If calling a certain module of library in main.py, need to mention the imported module name, use the syntax:

```
lib. module…
```

(3) Import module from library.py

```
from library import module
```

If calling a certain module of library in main.py, do not need to mention the imported module name, use the syntax:

```
module…
```

(4) Import all from library.py

```
from library import *
```

If calling a certain module of library in main.py, do not need to mention the imported module name, use the syntax:

```
module…
```

3.4.2. Reason to separate code into separate files

- Splitting code into different files and each file concentrate on a certain aspect of the whole program, a clear structure is formed and makes it easier to read, and understand.
- Smaller and more focused code also makes it easier to reuse in other programs.
- Easier to maintain, update and find problems.

In conclusion, separate files could improve **readability** and **maintainability**, which could significantly improve the efficiency of real-world programming.

3.5. Practical

Provide your reader with a completed set of functions that could benefit from testing. Require them to create a main function that will test at least 10 functions you have created that demonstrate different concepts covered so far. Provide the solution to this problem. Make sure you test outlier cases and test at the borders.

3.5.1. Original functions

10 functions are written as follows, we currently do not know if there are significant bugs in there functions.

```
#combine_text is to combine two strings into one, separated by a space
def combine_text(a, b):
    return (a + b)

#check_equal: check if two numbers are equal
def check_equal(a, b):
    return False
    if (a - b == 0):
        return True

#get_sum: get sum of three numbers
def get_sum(a, b, c):
    return a + b

#check_time: check if the format of the time is correct, including the hour an time
def check_time(hour,minute):
    if hour <= 0 or minute <= 0 or minute >= 60:
        return False
    else:
        return True

#check_positive: check if num a is positive
def check_positive(a):
    if a > 0:
        return True
    elif a < 0:
        return False

#division: get dividend divide divisor
def division(dividend, divisor):
    return dividend / divisor
```

```
#get_min: get minimum number from a,b,c
def get_min(a, b, c):
    if a < b and a < c:
        return a
    elif b < a and b < c:
        return b
    elif c < a and c < b:
        return c

#guess_number_21: given a num, see if it is equal to 21
def guess_number_21(num):
    if num > 21:
        return ('too large')
    else:
        return ('too small')

#first_character: given a string word, get its first character
def first_character(word):
    return word[1]

#floor_div: get dividend floor divide divisor
def floor_div(dividend, divisor):
    if divisor != 0:
        return dividend / divisor
```

3.5.2. Test main function

After the 10 functions are written, we need to create a main function to test if they can pass all the tests, including the edging cases.

```
#main function
def main():
    #test combine_text function
    if combine_text('i','f') == 'i f':
        print ('combine_text test passed!')
    else:
        print ('Error found')
    #test check_equal function
    if check_equal(1, 1) == True:
        print ('check_equal test passed!')
    else:
        print ('Error found')
    #test get_sum function
    if get_sum(2, 5, 1) == 8:
        print ('get_sum test passed!')
    else:
        print ('Error found')
    #test check_time function
    if check_time(0, 0) == True:
        print ('check_time test passed!')
    else:
        print ('Error found')
    #test check_positive function
    if check_positive(0) == False:
        print ('check_positive test passed!')
    else:
        print ('Error found')
    #test division function
    if division(4, 1) == None:
        print ('division test passed!')
    else:
        print ('Error found')
    #test get_min function
    if get_min(5, 5, 5) == 5:
        print ('get_min test passed!')
    else:
        print ('Error found')
    #test guess_number_21 function
    if guess_number_21(21) == 'you are right':
        print ('guess_number_21! test passed!')
    else:
        print ('Error found')
    #test first_character function
    if first_character('word') == 'w':
        print ('first_character test passed!')
    else:
        print ('Error found')
    #test floor_div function
    if floor_div(7, 2) == 3:
        print ('floor_div test passed!')
    else:
        print ('Error found')

main()
```

3.5.3. Initial test result

After the first round of testing, except the division function reported ZeroDivisionError, other functions are not giving us correct output, and the mianfunction printed ‘error found’.

```
Error found
Error found
Error found
Error found
Error found
ZeroDivisionError: division by zero
Error found
Error found
Error found
Error found
```

3.5.4. Functions revision

Based on the checking cases, the reasons of errors are as follows:

combine_text(a, b): there need to be a space between two strings

check_equal(a, b): the function always resutn false, need to put the if (a-b==0)condition at front

get_sum(a, b, c):c is not added in the result

check_time(hour,minute):hour and minute are able to equals to 0

check_positive(a):the case a =0 is forgotten

division(dividend, divisor):the divisor=0 condition is not considered

get_min(a, b, c):the equal of two numbers is not considered

guess_number_21(num):num=21 condition is not considered

first_character(word):the return character is the second character

floor_div(dividend, divisor): result is actual division, not floor division

After getting clear about the reasons of bugs, the functions are revised accordingly.

```
#combine_text is to combine two strings into one, separated by a space
def combine_text(a, b):
    return (a + ' ' + b)

#check_equal: check if two numbers are equal
def check_equal(a, b):
    if (a - b == 0):
        return True
    return False

#get_sum: get sum of three numbers
def get_sum(a, b, c):
    return a + b + c

#check_time: check if the format of the time is correct, including the hour an time
def check_time(hour,minute):
    if hour < 0 or minute < 0 or minute > 60:
        return False
    else:
        return True

#check_positive: check if num a is positive
def check_positive(a):
    if a > 0:
        return True
    elif a <= 0:
        return False
```

```
#division: get dividend divide divisor
def division(dividend, divisor):
    if divisor != 0:
    return dividend / divisor

#get_min: get minimum number from a,b,c
def get_min(a, b, c):
    min = a
    if b <= min :
        min = b
    if c <= min:
        min = c
    return min

#guess_number_21: given a num, see if it is equal to 21
def guess_number_21(num):
    if num == 21:
        return 'you are right'
    elif num > 21:
        return 'too large'
    else:
        return 'too small'

#first_character: given a string word, get its first character
def first_character(word):
    return word[0]

#floor_div: get dividend floor divide divisor
def floor_div(dividend, divisor):
    if divisor != 0:
        return dividend // divisor
```

3.5.5. Final test result

After finding reasons and debug, using test main function totest above revised functions, we will get the final result and all tests are passed.

```
combine_text test passed!
check_equal test passed!
get_sum test passed!
check_time test passed!
check_positive test passed!
division test passed!
get_min test passed!
guess_number_21! test passed!
first_character test passed!
floor_div test passed!
```

4. Chapter 4: While Loops

4.1. Explain the difference in and the different uses of a counting while loop, a sentinel while loop, and an infinite while loop.

A while loop contains a loop condition and code block, while the condition is True, the code block will be executed again and again until the condition is False.

while loop syntax:

```
while condition:
        statments
```

4.1.1. Counting while loop

Counting while loops have a counter, and this counter is referenced as the while loop condition, in each loop, the counter will be updated in the code block. when the counter meets the condition of a while loop, the loop is executed. In the loops counter may change and could not meet the while loop condition, the loop will stop.

The following example is a counting loop, in which integer i is a counter, and this counting loop will print “hi” five times.

Code:

```
i = 0
while i < 5:
        i += 1
        print ("hi")
```

Output:

```
hi
hi
hi
hi
hi
```

4.1.2. Sentinel while loop

Sentinel while loops are also called conditional loops, instead of using a counter, A sentinel loop keeps processing statements until it encounters a certain value that indicates the end. The sentinel is the name of the exceptional value.

The following example is a sentinel loop, in which integer i= 5 is the condition to end the loop, and value 5 is the sentinel.

Code:

```
i = 0
while i  != 5:
        i = input('i = ?')
print ("end")
```

4.1.3. infinite while loop

As suggested by the name, an infinite while loop is a while loop in which the while condition is permanently true. However, there can be an exit or break in the code block of the while loop to make the loop break or exit in a certain condition.

Code (with break):

```
i = 0
while True:
        i = input('i = ?')
```

Code (without break):

```
i = 0
while True:
        i = input('i = ?')
        if i = 10:
                break
```

4.2. Explain, with examples, break, exit, and continue.

4.2.1. Continue statement

The continue statement will jump to the start of the while loop, the rest part of the while loop body will not be implemented, and the while loop is not terminated.

For example, in the code below, when i > 1, the loop will jump to the start of the while loop, and the rest part of the while loop body, which is print("hi") will not be implemented. "hi" will only be printed for 1 time. print the statement after the loop, and "hello" will be printed.

Code:

```
i = 0
while i < 5:
    i += 1
    if i > 1:
        continue
    print ("hi")
print("hello")
```

Output:

```
hi
hello
```

4.2.2. Break statement

The break statement will terminate the loop, but the program will not be terminated, and statements after the while loop will be implemented.

For example, in the code below, when i = 4, the loop will terminate because of the break statement, and ‘hi’ would be printed 4 times, and print the statement after the loop, and “hello” will be printed.

Code:

```
i = 0
while i < 5:
    i += 1
    print ("hi")
    if i > 3:
        break
print("hello")
```

Output:

```
hi
hi
hi
hi
hello
```

4.2.3. Exit statement

The exit statement will terminate the program, and statements after the while loop will no longer be implemented.

For example, in the code below, when i = 4, the program will terminate because of the exit statement, and ‘hi’ would be printed 4 times, and print the statement after the loop, which is “hello” will not be printed.

Code:

```
i = 0
while i < 5:
        i += 1
        print ("hi")
        if i > 3:
                exit()
print("hello")
```

Output:

```
hi
hi
hi
hi
```

4.3. What are sequence, selection, and iteration? Why are they the backbone of algorithms?

4.3.1. Sequence

Sequence is the order of step-by-step processes to be executed

4.3.2. Selection

Selection is used to choose operations to carry out based on a Boolean statement.

4.3.3. Iteration

Iteration is to repetitively carry out actions in sequence until a certain condition is met.

4.3.4. Relation to algorithm

Since the program is formed by multiple lines of code and an algorithm is a step-by-step process, the code is executed line by line and all the steps are implemented in a certain sequence, sequence is a significant trait of the algorithm.

In an algorithm, usually, we need some statements to be implemented in a certain condition, skip some lines of code and execute certain lines of code. For this reason, selection is quite an essential element of an algorithm.

Most programs execute repetitive tasks for us to achieve a certain goal under the hood, this requires the algorithm of iteration, which is to repetitively carry out actions in sequence until a certain condition is met.

For the above reasons, sequence, selection, and iteration are the backbone of algorithms.

4.4. Describe, and provide an example of the complexity of debugging while loops.

While loops are usually used to do repetitive tasks, even a small flaw would make a large effect on the program, such as infinite loops, or loops not executed. Due to the complexity of loop structures, fixing bugs in them is more difficult than fixing bugs in other areas.

For example, in the buggy code below, the output will be “hi” infinitely, and the result is expected to be five times “hi” and one time “hello”.

Buggy Code:

```
i = 0
while i < 5:
        print ("hi")
print("hello")
```

Output**:**

```
hi
hi
hi
hi
…
```

As the while loop condition is about counter i, a statement “print("i = " + str(i))” is added in the while loop body to see what happened to i in the loop.

Try to debug buggy code:

```
i = 0
while i < 5:
    print ("hi")
    print("i = " + str(i))
print("hello")
```

Output**:**

```
hi
i = 0
hi
i = 0
hi
…
```

According to the output, we find the counter i is always 0 and not updated during loops, so the bug is that i is not updated and the while loop condition is always True.

After fixing the bug, the code will run as expected.

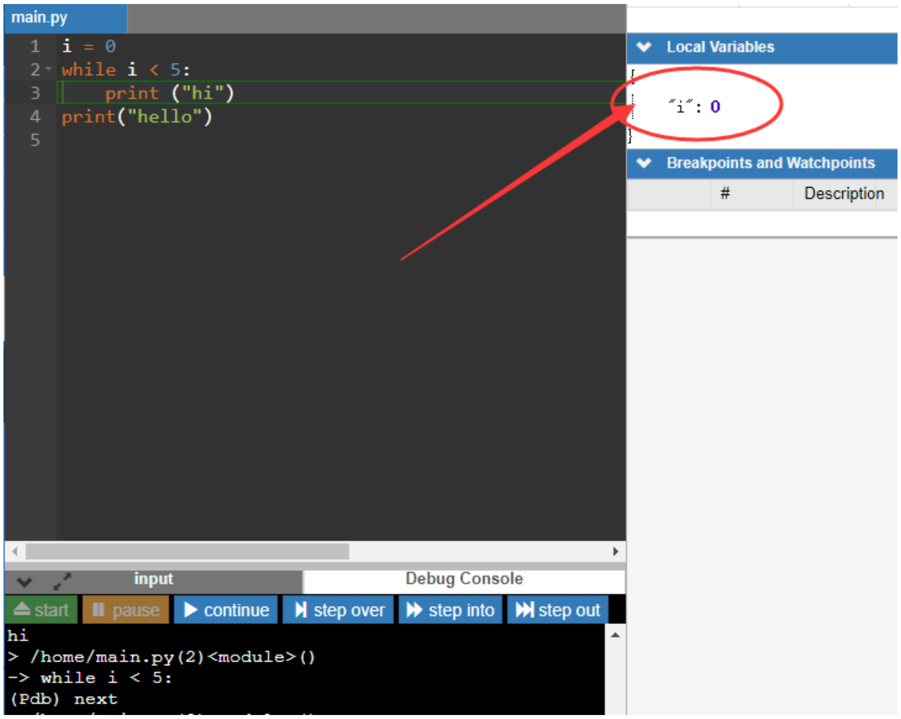
Corrected code:

```
i = 0
while i < 5:
    print ("hi")
    i += 1
print("hello")
```

Output**:**

```
hi
hi
hi
hi
hi
hello
```

Also, it is convenient to use a debugger, not hard to find i = 0 in every step, and just update the counter inside the while loop body to fix the bug.

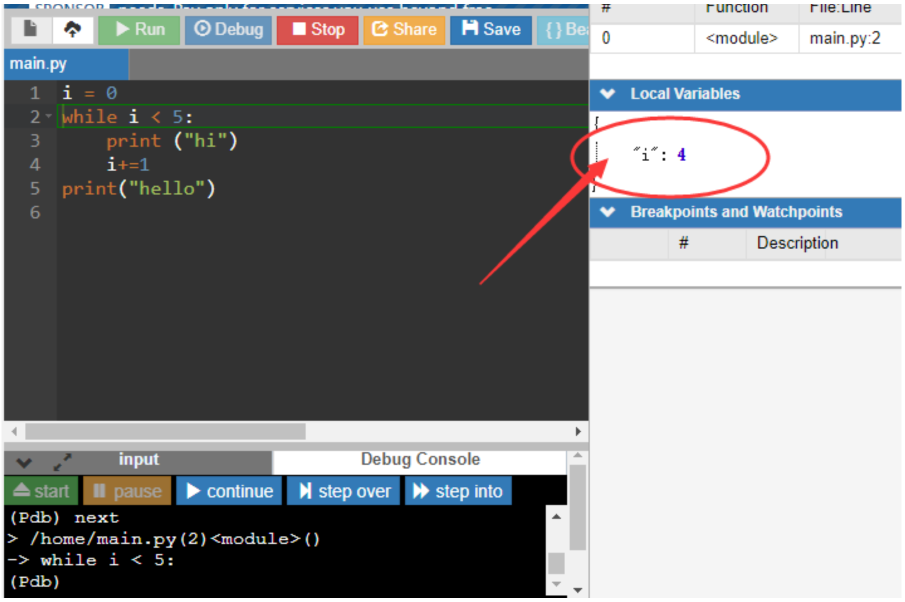


```
i = 0
while i < 5:
    print ("hi")
print("hello")
```

```
{
  "i": 0
}
```

```
hi
> /home/main.py(2)<module>()
-> while i < 5:
(Pdb) next
```

Buggy code



#	Function	File:Line
0	<module>	main.py:2

```
i = 0
while i < 5:
    print ("hi")
    i+=1
print("hello")
```

```
{
  "i": 4
}
```

```
(Pdb) next
> /home/main.py(2)<module>()
-> while i < 5:
(Pdb)
```

Correct code

4.5. Practical

Create a lab assignment that will help explain how nested while loops work. Then, provide the solution to that assignment.

4.5.1. Problem

Use nested while loop to create a reverse triangle pattern as follow.

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27
28 29 30 31 32 33 34
35 36 37 38 39 40
41 42 43 44 45
46 47 48 49
50 51 52
53 54
55
```

Hints:

Use the outer while loop to traverse all rows of the pattern;

Use the inner while loop to print all the numbers in the same row;

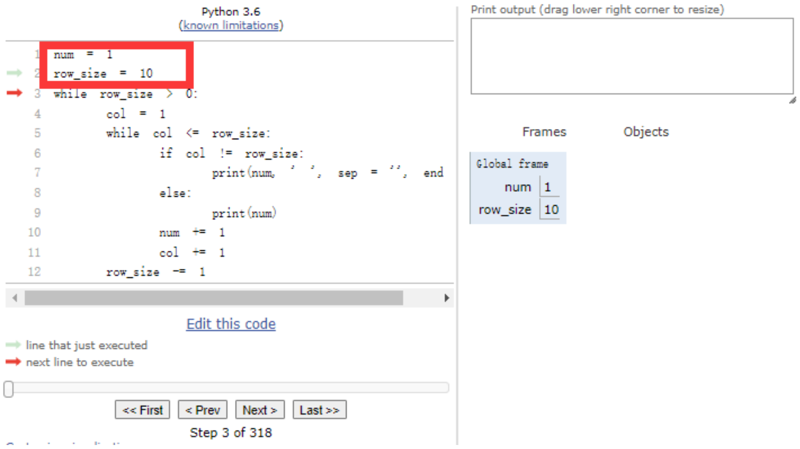
Printing of the last number in a row is a little different from other numbers.

4.5.2. Solution

```
#the value of number to be printed, start from 1
num = 1
#row_size: how many numbers in the same row, start from 10
row_size = 10
#The last row has one number, outer while loop ends at row_size = 0
while row_size > 0:
            #col: index of number in a row, start from 1 for each row
    col = 1
    #when row_size = col, the row is complete and jump to the next row
    while col <= row_size:
            #col != row, the row is not complete, print num and the space between numbers
        if col != row_size:
            print(num, ' ', sep = '', end = '')
        #col == row_size, the row is complete, print the last number in the row, to next line
        else:
            print(num)
        #next number is alway 1 larger than previous one
        num += 1
        #update col, jump to the next number in the row
        col += 1
    #number of nums minus 1 in the next row
    row_size -= 1
```

4.5.3. Steps

Initial condition:



```
num = 1
row_size = 10
while row_size > 0:
    col = 1
    while col <= row_size:
        if col != row_size:
            print(num, ' ', sep = '', end
        else:
            print(num)
        num += 1
        col += 1
    row_size -= 1
```

Step 3 of 318

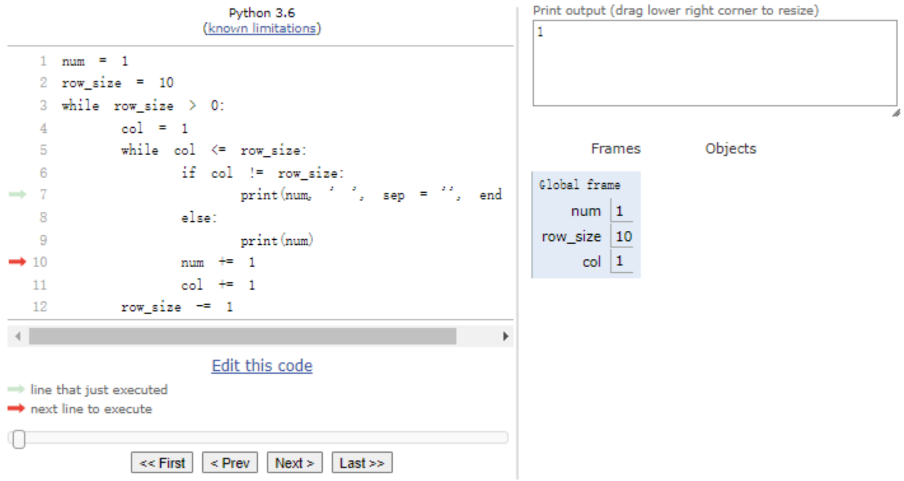
Global frame

num	1
row_size	10

The first outer loop:

row_size = 10, col = 1, num = 1

num 1 is printed



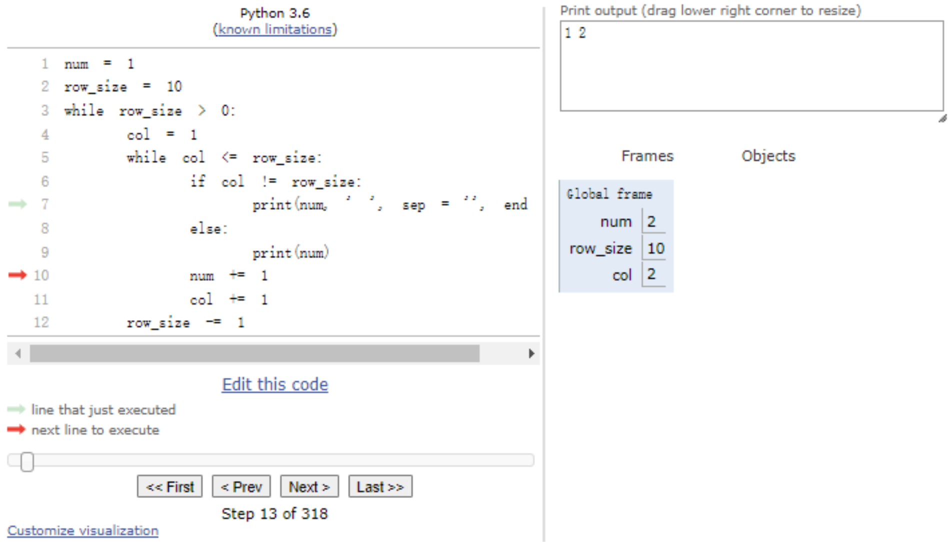
```
num = 1
row_size = 10
while row_size > 0:
    col = 1
    while col <= row_size:
        if col != row_size:
            print(num, ' ', sep = '', end
        else:
            print(num)
        num += 1
        col += 1
    row_size -= 1
```

```
1
```

Global frame	
num	1
row_size	10
col	1

row_size = 10, col = 2, num = 2

num 2 is printed



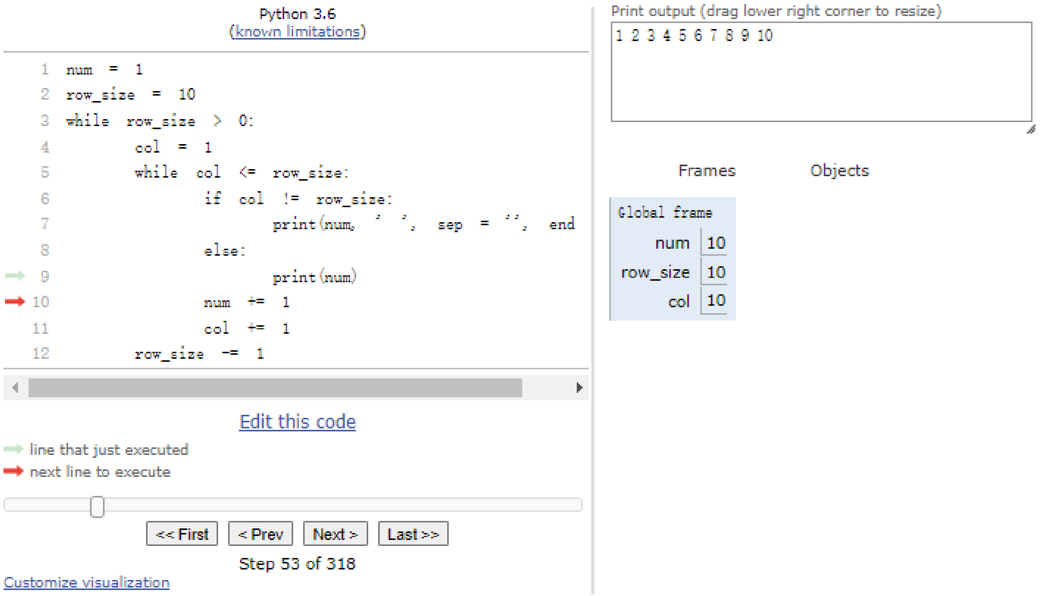
```
num = 1
row_size = 10
while row_size > 0:
    col = 1
    while col <= row_size:
        if col != row_size:
            print(num, ' ', sep = '', end
        else:
            print(num)
        num += 1
        col += 1
    row_size -= 1
```

```
1 2
```

Global frame	
num	2
row_size	10
col	2

Complete all steps in the inner loop until the first row is printed

row_size = 10, col = 10, num = 10



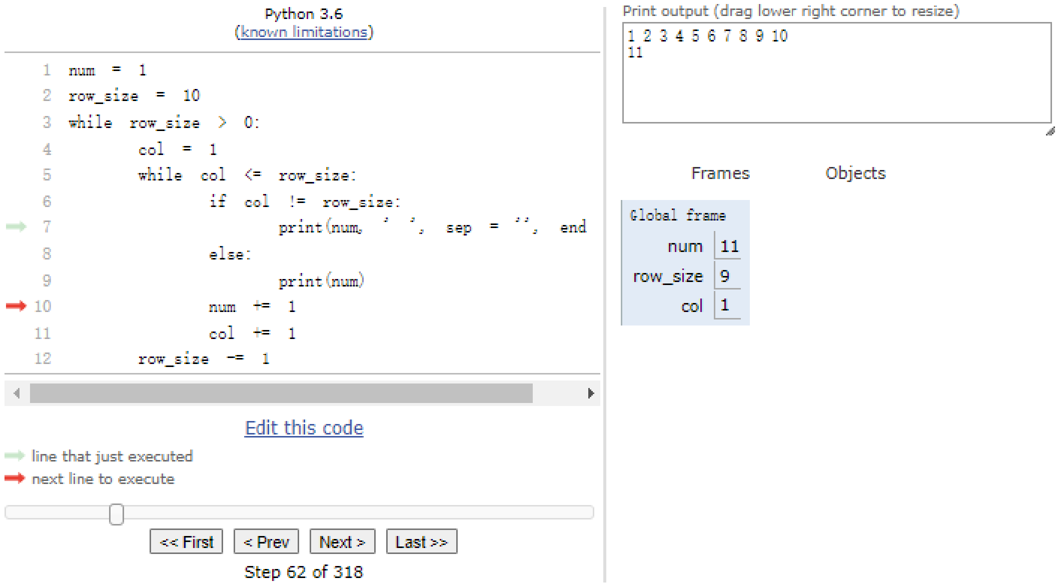
```
num = 1
row_size = 10
while row_size > 0:
        col = 1
        while col <= row_size:
                if col != row_size:
                        print(num, ' ', sep = '', end
                else:
                        print(num)
                num += 1
                col += 1
        row_size -= 1
```

```
1 2 3 4 5 6 7 8 9 10
```

Global frame

num	10
row_size	10
col	10

When the first row is complete, col = 11, row_size = 10, at this time the inner loop condition is not met, jump out to the outer loop, print the next line, row_size reduce to 9, and in the next row, reset col to 1.



```
num = 1
row_size = 10
while row_size > 0:
        col = 1
        while col <= row_size:
                if col != row_size:
                        print(num, ' ', sep = '', end
                else:
                        print(num)
                num += 1
                col += 1
        row_size -= 1
```

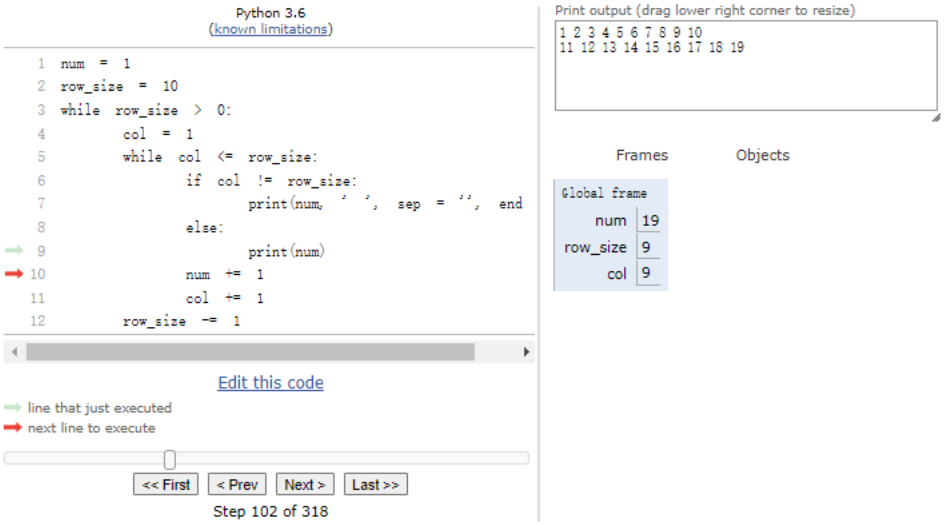
```
1 2 3 4 5 6 7 8 9 10
11
```

Global frame

num	11
row_size	9
col	1

The second time finish all inner loops, we will get the next row printed.

Num = 19, row_size = 9, col = 9



```
num = 1
row_size = 10
while row_size > 0:
    col = 1
    while col <= row_size:
        if col != row_size:
            print(num, ' ', sep = '', end
        else:
            print(num)
        num += 1
        col += 1
    row_size -= 1
```

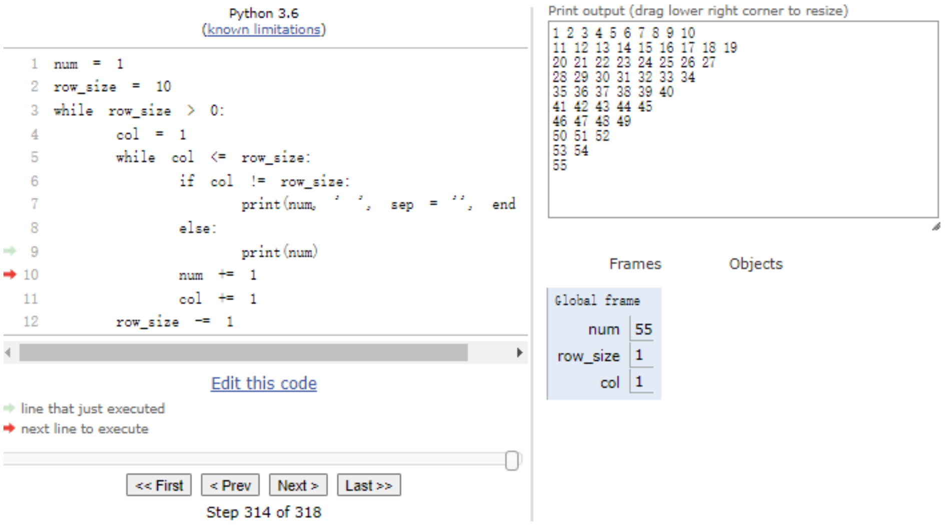
Step 102 of 318

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19
```

Global frame	
num	19
row_size	9
col	9

As steps above, rows 3~10 will be printed as well.

Before the ending of outer while loop, row_size = 1, and after row_size -= 1, row_size will become 0, this does not meet the outer while loop condition, row_size > 0, terminate outer loop at this time.



```
num = 1
row_size = 10
while row_size > 0:
    col = 1
    while col <= row_size:
        if col != row_size:
            print(num, ' ', sep = '', end
        else:
            print(num)
        num += 1
        col += 1
    row_size -= 1
```

Step 314 of 318

```
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27
28 29 30 31 32 33 34
35 36 37 38 39 40
41 42 43 44 45
46 47 48 49
50 51 52
53 54
55
```

Global frame	
num	55
row_size	1
col	1

Now there is no other code after the outer loop, and the program terminates.

5. Chapter 5: Strings and Lists

5.1. What are strings, how are they stored in memory, and what are the various operations you can use on strings.

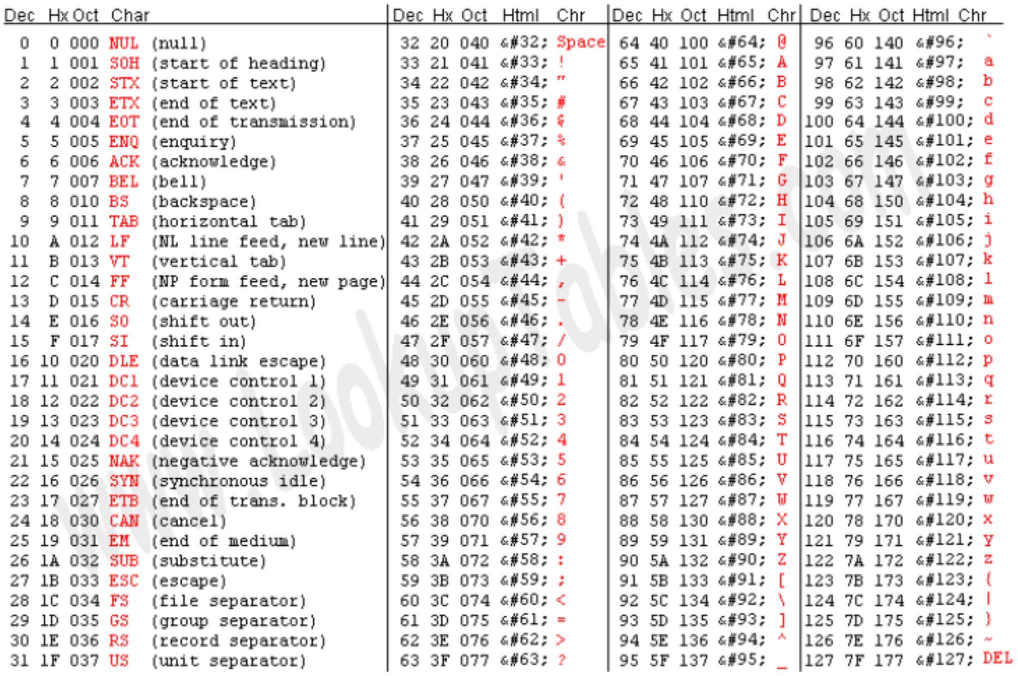
5.1.1. Definition of string

Strings are the short form of strings of characters, generally, any group of characters that can be interpreted literally can be called a string. For example, “hello”, “100”, “asd346sadf” are strings.

5.1.2. String in memory

Each character in a string is encoded in binary, and the string is kept in a single, continuous memory location as individual characters.

The following picture is a table for character, hex, oct, and binary transfer.



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

5.1.3. String operation

upper()	String is changed to upper case
lower()	String is changed to lower case
count()	Returns how many times a given value appears in a string.
replace()	Replaces a provided string with a specified one in a string.
split()	returns a list after splitting the string at the specified separator.

join()	Converts the elements into a string

Example:

```
str = "hello"
print(str.upper())
print(str.lower())
print(str.count("l"))
print(str.replace("l","o"))
print(str.split("e"))
print(" ".join(str))
```

Output:

```
HELLO
hello
2
heooo
['h', 'llo']
h e l l o
```

5.2. What are mutable and immutable objects? Provide some examples and speculate on why both exist.

5.2.1. Mutable objects

Mutable objects can change themselves, including content and state.

Examples of mutable data types: list, dictionary, set, and customized classes

5.2.2. Immutable objects

An immutable object is an object which can not change its content and state after it is created. When an immutable object is created, a special id is given to it. The object's type is determined at runtime and cannot be altered later.

Examples of immutable data types: int, float, bool, string, tuple

5.2.3. Comparations

When you need to make adjustments without creating a new object, mutable objects are convenient. But anytime you change the object, all references to that object will also change. It can provide flexibility when there is a need to change the object.

Immutable objects are quicker to access and can not change, which provides stability. Immutable objects can be useful in multi-threaded since no need to worry about other threads altering the data.

Since there are benefits in mutable objects and immutable objects in certain circumstances, they do have reasons to exist.

5.3. Explain the concept of an object in Python. Provide some examples and discuss what’s going on in memory.

5.3.1. Object

An object contains a collection of data (variables) and methods (functions) that can operate on that data.

Python is an Object-Oriented Programming language, and almost everything in Python is an object.

For example, int, float, string, bool are all objects.

```
a = 100
b = 100.0
c = "100.0"
d = True
```

5.3.2. Object in memory

When an object is created, python automatically allocates memory for the object, and gets stored in the Heap Memory, some objects can be passed by reference according to an unique id, which is the memory address where the object is stored.

5.4. What is a list in Python? Include a discussion of the memory, syntax, and their usage.

5.4.1. Definition of list

Lists are used to store multiple items in a single variable, declared with [].

For example, list a have 4 int, and list b contains int, string and list a.

Code:

```
a = [1,2,3,4]
b = [1,"a", a]
print(a)
print(b)
```

Output:

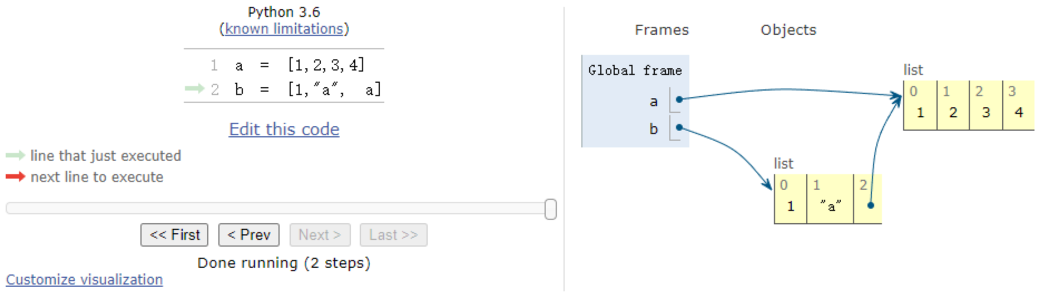
```
[1, 2, 3, 4]
[1, 'a', [1, 2, 3, 4]]
```

5.4.2. Memory of list

When a list is created, 64 bytes of memory is allocated for the list, and each object inside the list will add 8 bytes to the list size, and the list store references to objects inside the list.

For example:

```
a = [1,2,3,4]
b = [1,"a", a]
```



```
a = [1,2,3,4]
b = [1,"a",  a]
```

5.5. Practical

Create a useful application of your own design that uses lists and strings. Make sure it is fully realized, well documented, and complete.

5.5.1. Description

The program is based on CLI interface and aimed to select a lucky award winner among 20 candidates, which has functions such as adding a candidate, deleting a candidate, getting a lucky candidate, showing all candidate names, and showing the candidate amount.

5.5.2. Code

```
import random

#list candidates to store all the candidites joined the lottery
candidates = []

#function for adding a lottery candidate, string candidate is candidate's name
def add_candidate(candidate):
    #change the string candidate into lower case
    candidate = candidate.lower()
    #candidate are able to add to list if it is not in the list candidates
    if candidate not in candidates and len(candidate) > 0:
        #the size of list candidates is limited to 20
        if len(candidates) < 20:
            #add candidate name into the list candidates in lowercase
            candidates.append(candidate)
        else:
            #enough people to start lottery, print a message
            print("Ready to start lottery.\n")
    else:
        #the name has already been in the list, or name is empty, print a message
        print("Unable to add candidate.\n")

#function for deleting a lottery candidate, string candidate is candidate's name
def cancel_candidate(candidate):
    #change the string candidate into lower case
    candidate = candidate.lower()
    #only available when there are at least 1 person in the list candidates, and candidate to remove is in the list candidates
    if len(candidates) > 0 and candidate in candidates:
        #remove the candidate from the list
        candidates.remove(candidate)
    else:
        #unable to remove from the list, print a message
        print("Unable to cancel.\n")
```

```
#function to get a lucky candidate who won the lottery
def get_lucky_candidate():
    #set at least 15 persons to start the lottery
    if len(candidates) < 15:
        print("No enough candidates to start!\n")
    else:
        #get a random number as index of list candidates
        index = random.randint(0, len(candidates) - 1)
        #print the award message to the lucky candidate, name set to uppercase in title
        print(f"{candidates.pop(index).title()}, you won the award!\n")

#function to show all candidates who joined the lottery
def show_all_candidates():
    print('Here are all candidates:\n')
    #loop the list candidates
    #sort the list
    candidates.sort()
    for candidate in candidates:
        #name set to uppercase in title
        print(candidate.title())

#function to show candidates amount
def show_candidates_number():
    print(f'{len(candidates)} candidates joined the lottery.\n')

#function to load 20 example candidates for testing
def load_example_candidates():
    #string including 20 names, separated by ","
    candidates_str =
'Olivia,Emma,Charlotte,Amelia,Ava,Sophia,Isabella,Mia,Evelyn,Harper,Luna,Camila,Gian
na,Elizabeth,Eleanor,Ella,Abigail,Sofia,Avery,Scarlett'
    #transfer the string into list with 20 strings
    candidates_examples = candidates_str.split(',')
    #add 20 names to the list candidates
    for candidates_example in candidates_examples:
        add_candidate(candidates_example)
```

```
def main():
    #the menu message
    menu =  "1. Join the lottery\n"
    menu += "2. Cancel joining lottery\n"
    menu += "3. Get lucky candidate\n"
    menu += "4. Show all candidates names\n"
    menu += "5. Show candidate numbers\n"
    menu += "6. Quit\n"
    menu += "7. Load example 20 candidates\n"
    #boolean to decide whether program is running
    running = True
    while running:
        #get user selection, run different functions based on the selection
        selection = input(menu)
        if selection == "1":
            candidate = input("Please enter your full name")
            add_candidate(candidate)
        elif selection == "2":
            candidate = input("Please enter your full name")
            cancel_candidate(candidate)
        elif selection == "3":
            get_lucky_candidate()
        elif selection == "4":
            show_all_candidates()
        elif selection == "5":
            show_candidates_number()
        elif selection == "6":
            #terminate the loop
            running = False
        elif selection == "7":
            load_example_candidates()
        else:
            #handle wrong input
            print ("Please select an option from 1 ~ 7!")
#run main function
main()
```

6. Chapter 6: For Loops

6.1. Describe the syntax, and use of for loops.

A for loop can be used for iterating repeatedly over a range and iterator through a sequence, which allows us to run a series of instructions once for each element of a string, a list, a tuple, a set, or a range.

The syntax follows a series of rules.

- The keyword "for" denotes the start of the for loop, also the start of the header
- The iterator variable follows, iterating across the sequence and serving a variety of purposes within the loop.
- "in" keyword, which instructs the iterator variable to loop over the sequence's items
- the sequence might be a list, a string, an integer in a certain range, or any other sort of iterator
- the header ends with a colon
- statements are followed after the header

for loop syntax:

```
for iterator_variable in sequence:
        statments
```

Example of for loop a range:

```
for i in range(0, 5):
        print(i)
```

```
0
1
2
3
4
```

In the above example. i is the iterator_variable, the sequence is the number in range(0,5), which is 0,1,2,3,4, and these numbers are printed in sequence.

6.2. Compare and contrast while and for loops. Include when you would use each.

While loop and for loop run code repeatedly, they have different syntax and format.

Syntax of while loop:

```
while condition:
        statments
```

Syntax of for loop:

```
for iterator_variable in sequence:
        statments
```

For loops are counted loops, before the loop even begins, we know precisely how many times it will run. When we know the iteration times, such as an iterable object(list, string) or a certain range, for loop is appropriate to be used.

While loops are usually used based on a certain condition, and this condition could be based on the count or other conditions. If we don’t know the iteration times and the loop condition is known, a while loop is suitable.

While loops cover a larger range of use for iteration than for loops. For loops can always be replaced with while loops. For loop is usually faster than a while loop.

6.3. Describe, with examples the use of for loops to iterate of a string or list.

6.3.1. For loop a string

Strings hold characters in sequence and are iterable objects. In the following example. i is the iterator_variable, representing a character in the string. The sequence is the order in the string from the first character to the last character.

In the output, all the characters in the string “north” are printed, from the first character to the last.

Code:

```
for i in 'north':
        print(i)
```

Output:

```
n
o
r
t
h
```

6.3.2. For loop a list

Lists hold objects in sequence and are iterable objects. In the following example. i is the iterator_variable, representing an object in a list. The sequence is the order in the list from the first index to the last index.

In the output, all the objects in the list are printed, from index 0 to index 2.

Code:

```
list = ['north', 'eastern', 'university']
for i in list:
    print(i)
```

Output:

```
north
eastern
university
```

6.4. Explain how a nested for loop works and when you would use one.

The nested for loop in python is to use one loop inside another loop, the inner loop will be executed one time for each iteration of the outer loop. The syntax is as follows.

```
for iteratorr_variable1 in sequence1:
    statments
    for iteratorr_variable2 in sequence2:
        statments
```

Nested for loop is suitable for the condition in which each loop of the outer loop, and the inner loop is executed entirely.

A typical example is nested loop objects in two different dimensions, such as a matrix. If we need to traverse all numbers, we can use the outer loop to scan each row, and the inner loop to scan each number in different columns of this certain row.

The following example is to loop all possible combinations of objects in two lists.

Code:

```
for i in ['1', '2', '3']:
    for j in ['northeastern', 'university']:
        print(i+j)
```

Output:

```
1northeastern
1university
2northeastern
2university
3northeastern
3university
```

For each string in ['1', '2', '3'], all string in ['northeastern', 'university'] are looped, in the following order.

```
Outer loop i = '1':
    Inner loop j = 'northeastern' , j =  'university'
Outer loop i = '2':
    Inner loop j = 'northeastern',  j =  'university'
Outer loop i = '3':
    Inner loop j = 'northeastern',  j = 'university'
```

6.5. Practical

Nested for loops often “trip” up students new to programming. Create a tutorial, including code, that will help explain this concept to a learner new to programming. Walk them through it step by step. If you require them to solve anything, make sure to provide the solution.

6.5.1. Create a multiplication table

Create a multiplication calculation list of numbers 0 ~ 9 by a nested for loop, the table is represented by a list with nine lists inside, and each list inside contains nine integers, shown as follows.

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18],
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27],
[0, 4, 8, 12, 16, 20, 24, 28, 32, 36],
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45],
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54],
[0, 7, 14, 21, 28, 35, 42, 49, 56, 63],
[0, 8, 16, 24, 32, 40, 48, 56, 64, 72],
```

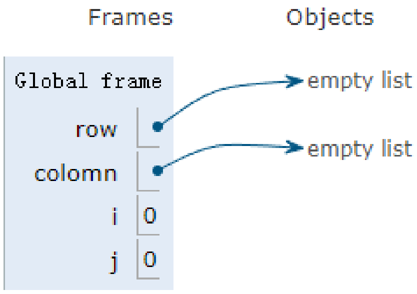
[0, 9, 18, 27, 36, 45, 54, 63, 72, 81]]

6.5.2. Code for reference

```
row = []
column = []
for i in range(0, 10):
        for j in range(0, 10):
                row.append(i * j)
        column.append(row)
        row = []
print (column)
```

6.5.3. Steps

Create two empty lists row and column, int i indicate the index in the list column and j indicates the index in the list row.



In the first outer loop, i = 0, the inner loop traverses j, which is all the numbers in 0 ~ 9, list row adds i * j.

According to the order of the inner loop, the following numbers are added to the empty list row. row = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] is created.

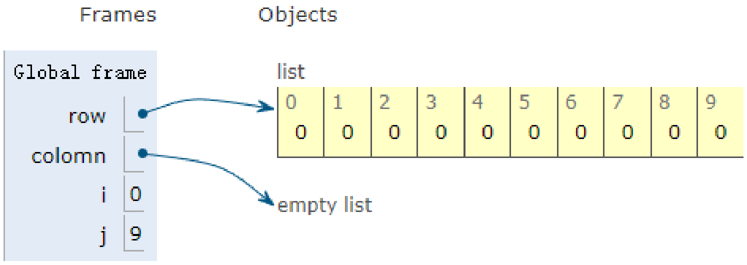
i*j=0*0

i*j=0*1

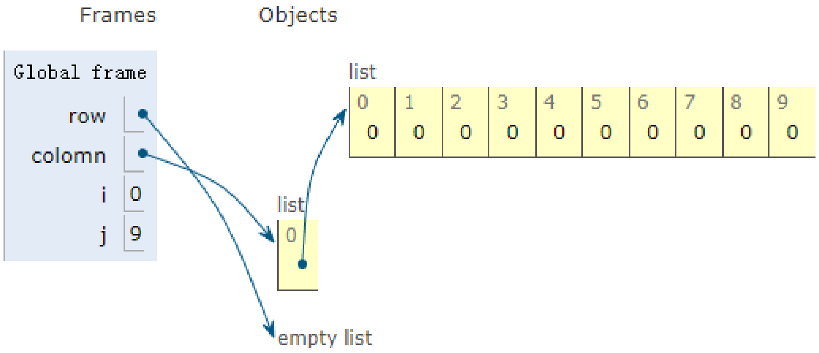
…

i*j=0*9

In the first traverse of inner loop, after row = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] is created, it is adde to the list colomn as the first object.



The next step is to create the second list in the column, the list row needs to reset to an empty list.



In the second outer loop, i = 1, the inner loop traverses j, which is all the numbers in 0 ~ 9, list row adds i * j.

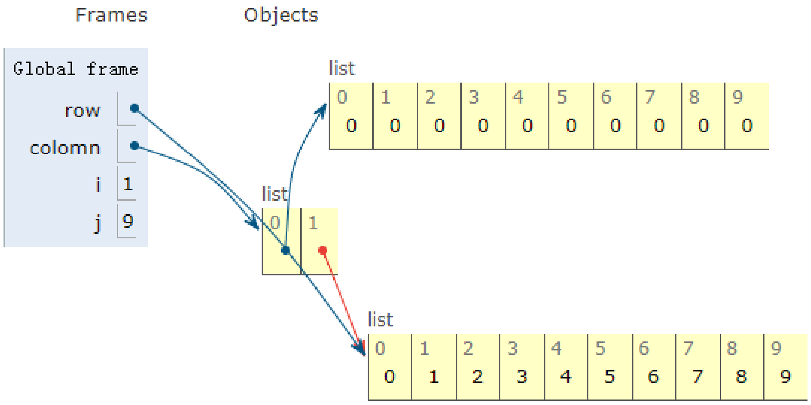
According to the order of the inner loop, the following numbers are added to the empty list row. row =[0, 1, 2, 3, 4, 5, 6, 7, 8, 9] is created.

i*j=1*0

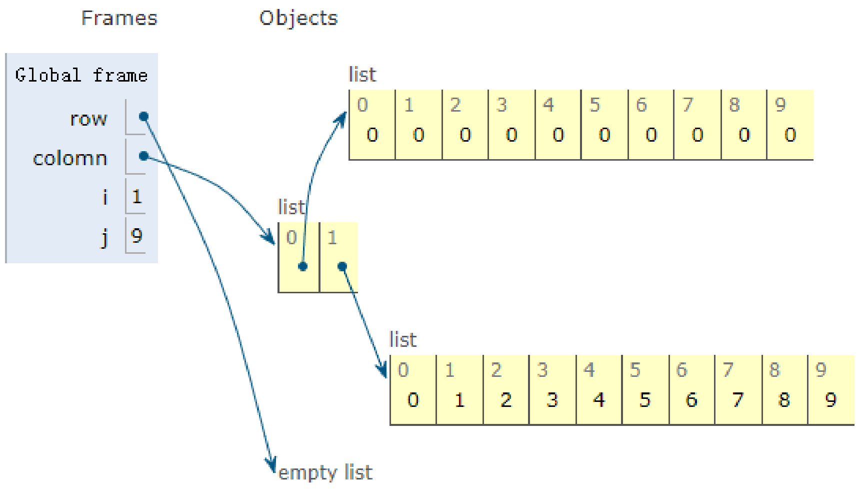
i*j=1*1

…

i*j=1*9



In the second traverse of the inner loop, after row = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] is created, it is added to the list column as the second object.



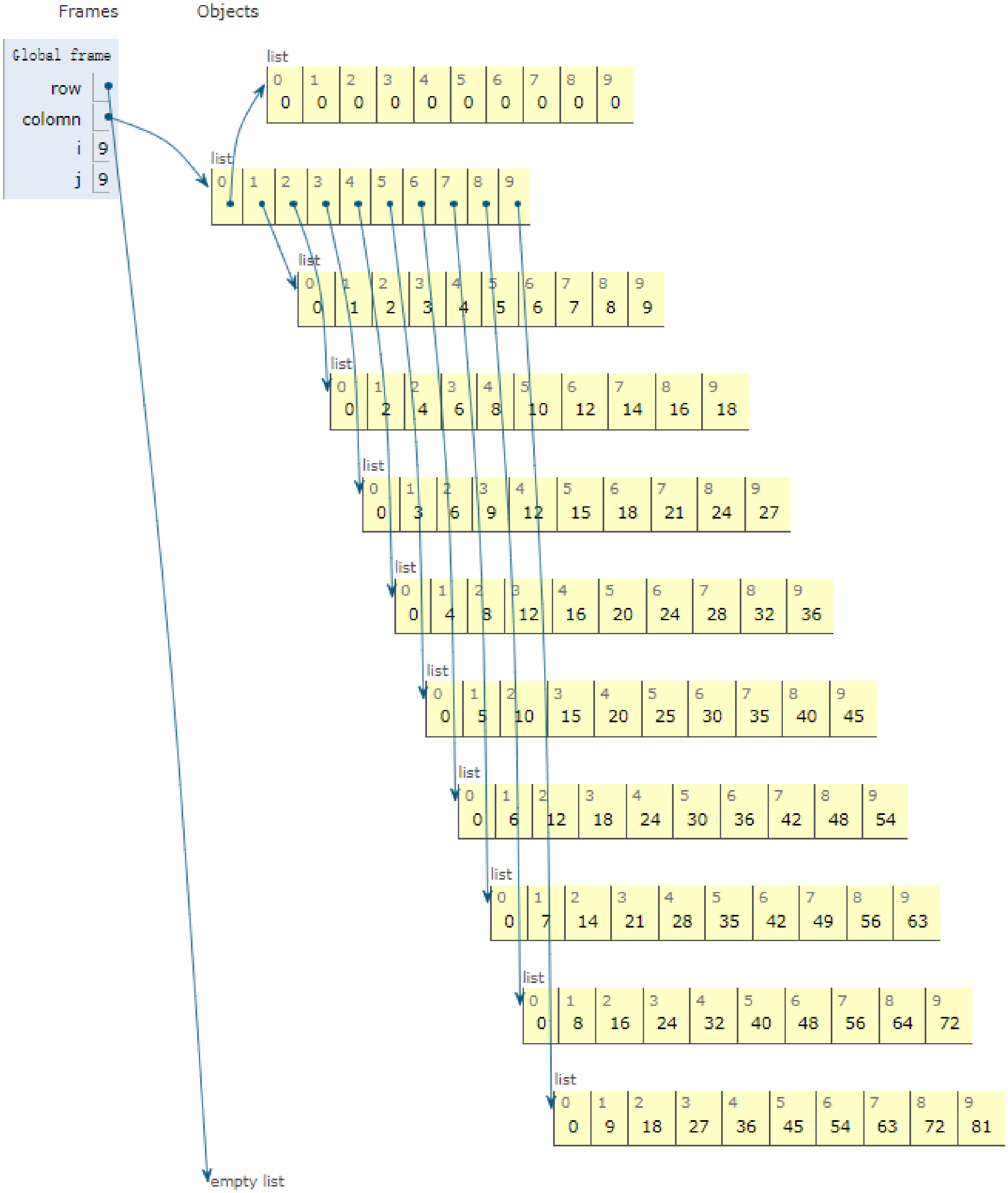
The next step is to create the third list in the column, the list row needs to reset to an empty list.

The list column contains nine lists, and the third list to the ninth list is created in the same way as the previous two row lists. As the result is saved in the column, when we print out the list column, will get the output as follows:

Output:

```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18], [0, 3, 6, 9, 12, 15,
18, 21, 24, 27], [0, 4, 8, 12, 16, 20, 24, 28, 32, 36],
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45], [0, 6, 12, 18, 24,
30, 36, 42, 48, 54], [0, 7, 14, 21, 28, 35, 42, 49, 56,
63], [0, 8, 16, 24, 32, 40, 48, 56, 64, 72], [0, 9, 18,
27, 36, 45, 54, 63, 72, 81]]
```

The final result after the nested for loops is shown in the following picture.



Global frame	
row	●
colomn	●
i	9
j	9

list

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

list

0	1	2	3	4	5	6	7	8	9
●	●	●	●	●	●	●	●	●	●

list

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

list

0	1	2	3	4	5	6	7	8	9
0	2	4	6	8	10	12	14	16	18

list

0	1	2	3	4	5	6	7	8	9
0	3	6	9	12	15	18	21	24	27

list

0	1	2	3	4	5	6	7	8	9
0	4	8	12	16	20	24	28	32	36

list

0	1	2	3	4	5	6	7	8	9
0	5	10	15	20	25	30	35	40	45

list

0	1	2	3	4	5	6	7	8	9
0	6	12	18	24	30	36	42	48	54

list

0	1	2	3	4	5	6	7	8	9
0	7	14	21	28	35	42	49	56	63

list

0	1	2	3	4	5	6	7	8	9
0	8	16	24	32	40	48	56	64	72

list

0	1	2	3	4	5	6	7	8	9
0	9	18	27	36	45	54	63	72	81