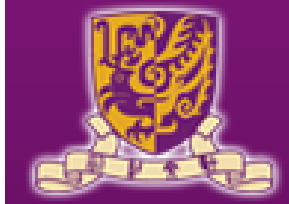


香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Introduction to AI Programming

Lecture 2 Python Basics

Prof. Junjie Hu

School of Artificial Intelligence

Outline

- Compiler vs Interpreter
- Interactive and Script Mode
- Variable, Constant, Reserved words
- Data Types: Integer, Float, String
- Functions: print(), int(), float(), str(), type()
- Call Expression and Nested Call Expression

Interpreter

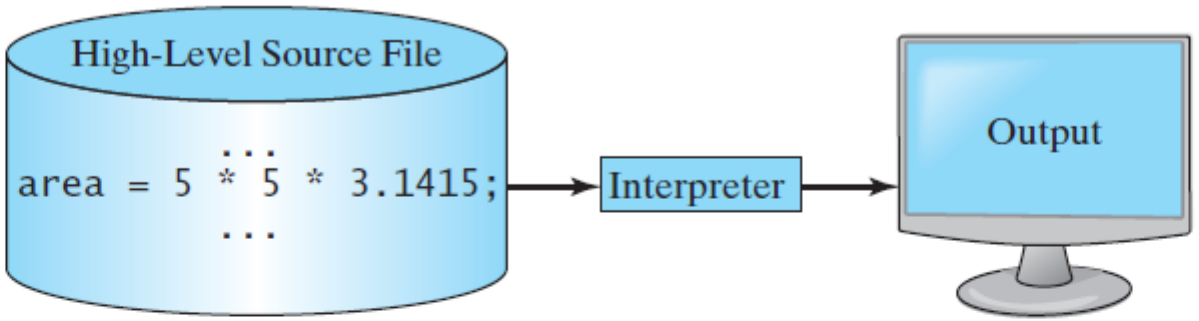
- Interpreter (解释器) is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without **previously** compiling them into a machine language program

Interpreter v.s. compiler

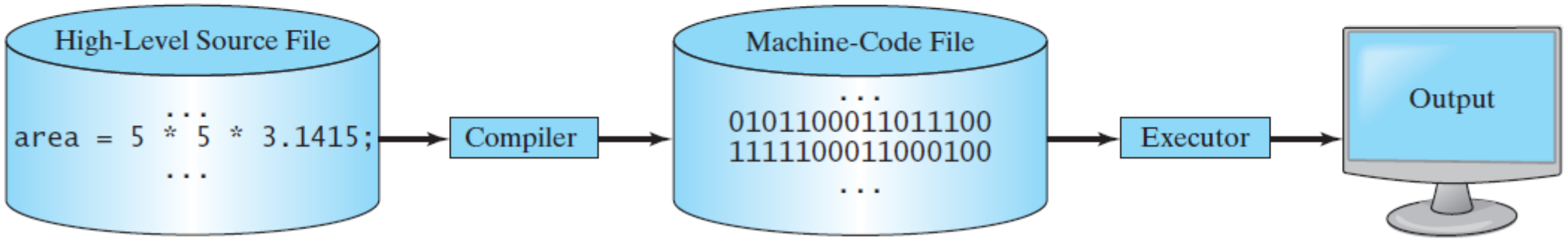
- Interpreter (解释器) is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without **previously** compiling them into a machine language program

- A compiler (编译器) is a computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code

Interpreter v.s. compiler



(a)

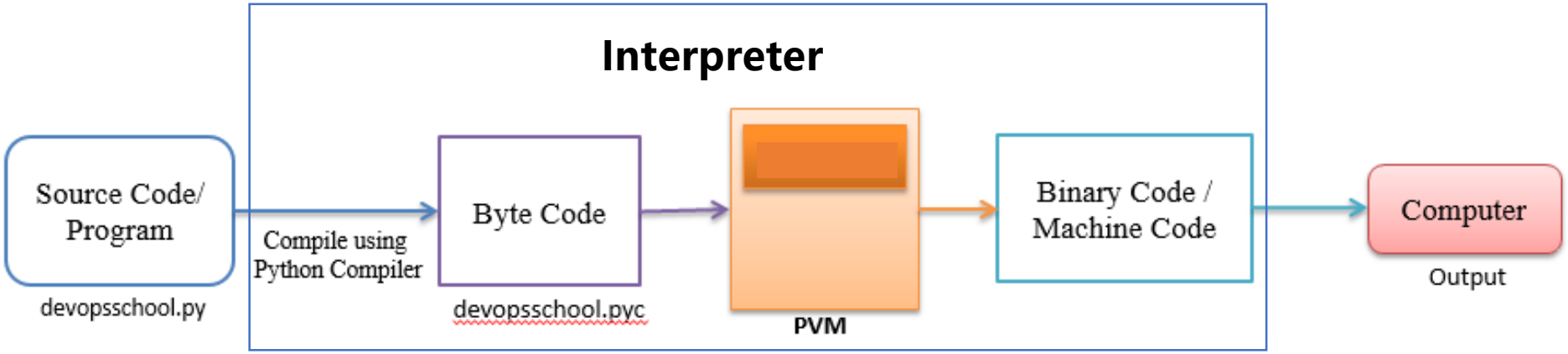


(b)

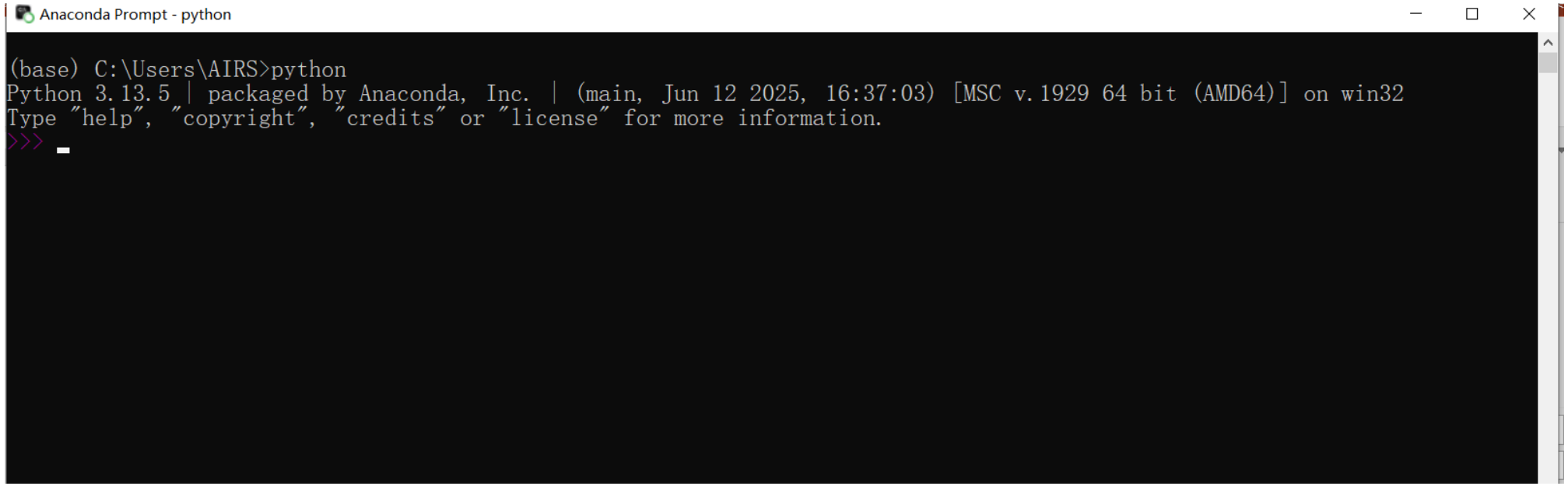
Is Python interpreted or compiled?

https://nedbatchelder.com/blog/201803/is_python_interpreted_or_compiled_yes.html

Python Interpreter



Python Shell



What is next?

Syntax Error

```
(base) C:\Users\AIRS>python
Python 3.13.5 | packaged by Anaconda, Inc. | (main, Jun 12 2025, 16:37:03) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> tell me who you are?
  File "<python-input-0>", line 1
    tell me who you are?
         ^^

SyntaxError: invalid syntax
>>>
```

Early learner: syntax error

- We need to learn the Python language so we can communicate our instructions to Python. In the beginning we will make lots of mistakes and speak gibberish like small children

- When you make a mistake, the computer does not think you are “cute”. It says “syntax error” – given that it “knows” the language and you are just learning it. It seems like Python is cruel and unfeeling

- You must remember that you are intelligent and can learn, while the computer is simple and very fast – but cannot learn

- It is easier for you to learn Python than for the computer to learn human language

The more you practice, the better you’ll be!

Hello, world!

```
(base) C:\Users\AIRS>python
Python 3.13.5 | packaged by Anaconda, Inc. | (main, Jun 12 2025, 16:37:03) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> tell me who you are?
  File "<python-input-0>", line 1
    tell me who you are?
         ^^

SyntaxError: invalid syntax
>>> print("tell me who you are?")
tell me who you are?
>>> print("hello world")
hello world
>>> print('hello world')
hello world
>>>
```

- You must say something that Python interpreter can understand!!

- Print() is a function in Python

Exit()

```
(base) C:\Users\AIRS>python
Python 3.13.5 | packaged by Anaconda, Inc. | (main, Jun 12 2025, 16:37:03) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> tell me who you are?
  File "<python-input-0>", line 1
    tell me who you are?
         ^^

SyntaxError: invalid syntax
>>> print("tell me who you are?")
tell me who you are?
>>> print("hello world")
hello world
>>> print('hello world')
hello world
>>> exit()

(base) C:\Users\AIRS>
```

Programming scripts

- Interactive Python is good for experiments and programs of 3-4 lines long
- Most programs are much longer, so we have to type them in a file and execute them all together
- In this sense, we are giving Python a script
- As convention, ".py" is added as the suffix on the end of these files

Interactive v.s. script

- Interactive

✓ You type directly to Python one line at a time and it responds

- Script

✓ You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the file

What should we say to Python ?

Elements of Python Language

- Vocabulary/words – Variables and Reserved words
- Sentence structure – valid syntax patterns
- Story structure – constructing a meaningful program for some purposes

Variable

- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name
- Variable names are determined by programmers
- The value of a variable can be changed later in a program

https://www.youtube.com/watch?v=p3q5zWCw8J4

Rules for defining variables in Python

- Must start with a letter or underscore _

  courseName _courseName

- Can only contain letters, numbers and underscore

  course~Name

- Case sensitive

  courseName coursename

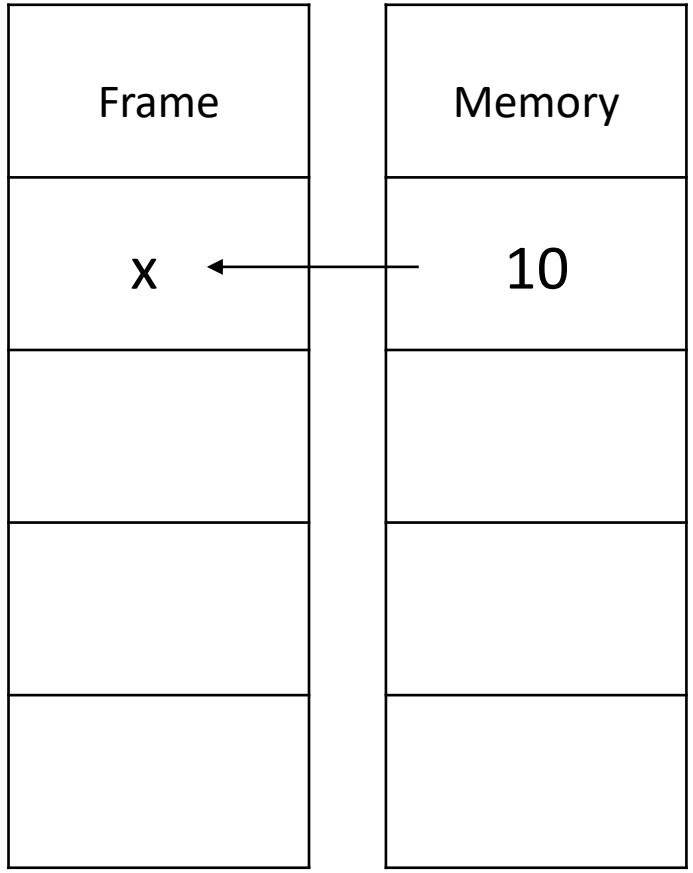
- Good: apple, car, myNumber123, _light

- Bad: 456aaa, #ab, var.12

- Different: apple, Apple, APPLE

Variable

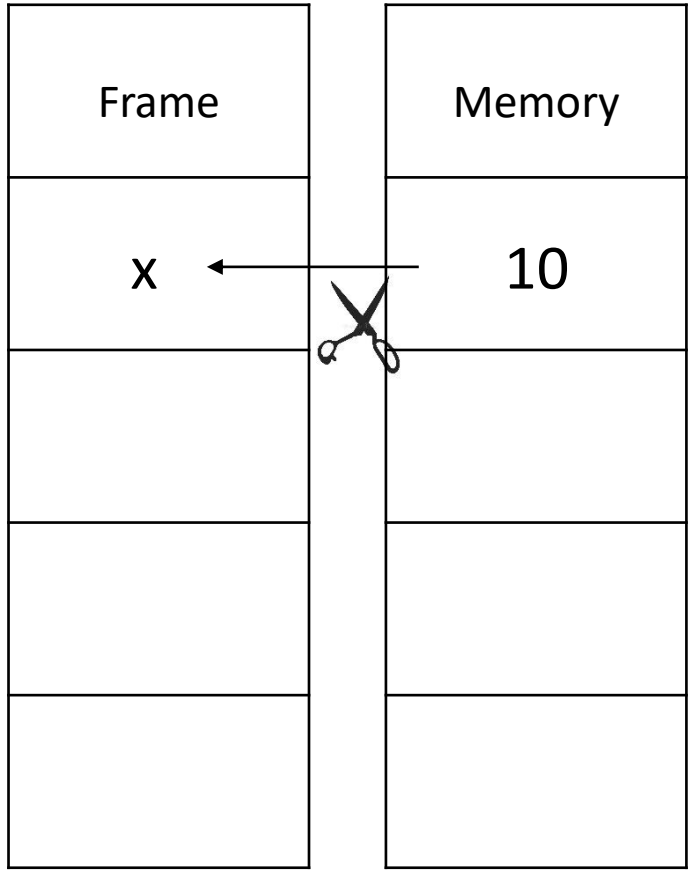
- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name



>>> x = 10

Variable

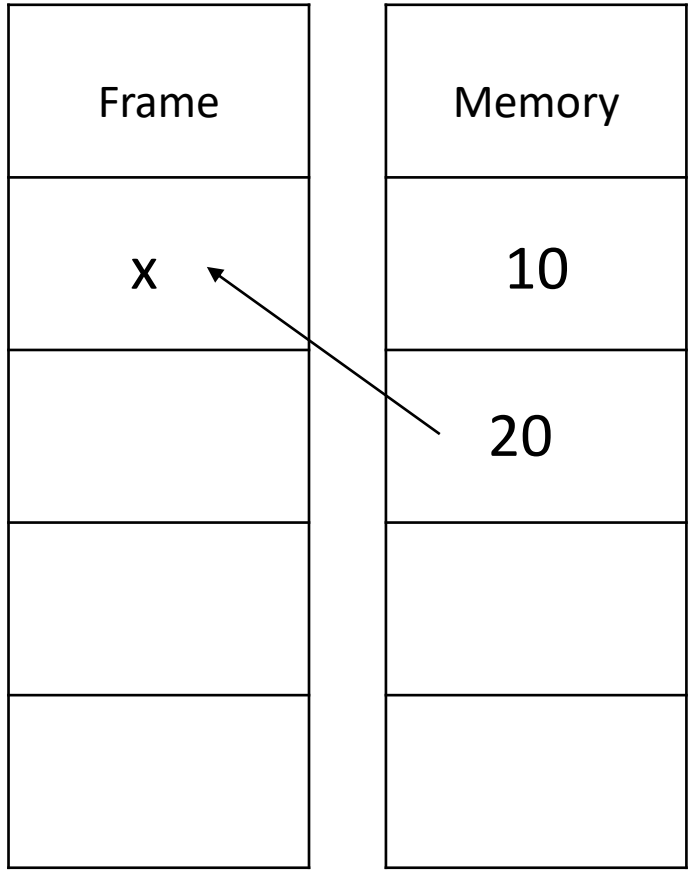
- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name




```
>>> x = 10
>>> x = 20
```

Variable

- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name

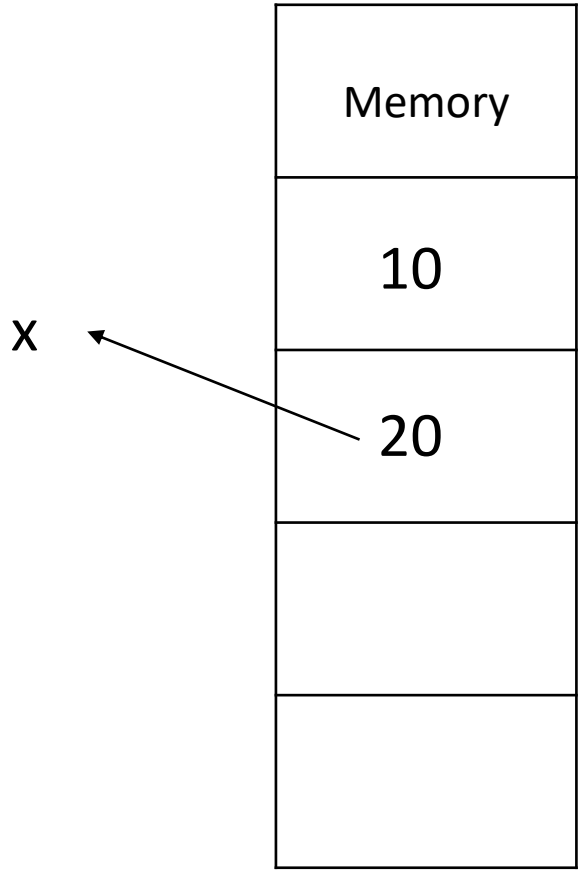



```
>>> x = 10
>>> x = 20
```

Variable

- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name

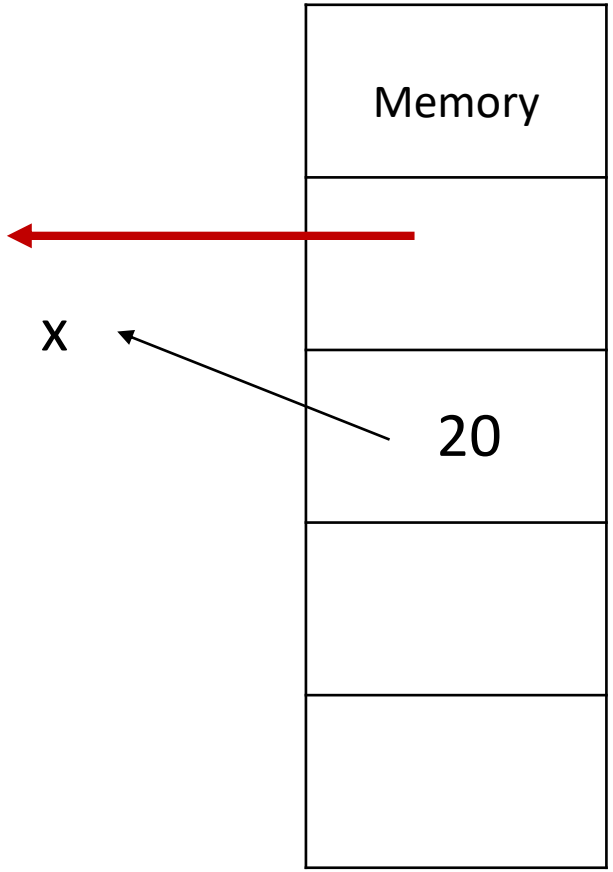
An integer variable is immutable




```
>>> x = 10
>>> x = 20
>>> print (x)
20
```

Variable

- A variable is a named space in the memory where a programmer can store data and later retrieve the data using the variable name



This memory address will be released by Python when no part of your code references it anymore.

```
>>> x = 10
>>> x = 20
>>> print (x)
20
```

Personal tips

- Use meaningful words as variable names
- Start with a lower letter
- Capitalize the first letter of each word
- Example: myBankAccountID, numOfCards, salaryAtYear1995…

What is this code doing?

```
xlq3z0ocd = 35.0
xlq3z0afd = 12.5
xlq3p0ocd = xlq3z0ocd * xlq3z0afd
print(xlq3p0ocd)
```

Reserved words

- You cannot use the following words as variables

```
False    None     True     and      as       assert   break
class    continue def      del      elif     else     except
finally  for      from     global   if       import   in
is       lambda   nonlocal not      or       pass     raise
return   try      while    with     yield
```

Constants

- Fixed values such as numbers and letters are called constants, since their values won't change

- String constants use single-quotes (') or double-quotes (")

Constants

- We cannot declare a variable or value as constant in Python.

- By convention, we use **uppercase letters** for constant names to remind ourselves (and others) that these values should not be changed.

```python
# Defining constants (by convention, in uppercase)
PI = 3.14159
GRAVITY = 9.8

# Using constants in calculations
radius = 5
area = PI * (radius ** 2)  # Area of a circle

print("Radius:", radius)
print("Area of the circle:", area)
print("Gravity on Earth:", GRAVITY)
```

Data Type

- In Python, variables and constants have an associated “type”

- Python knows the difference between a number and a string

- Example:

```
>>> a = 100 + 200
>>> print(a)

>>> b = "100" + "200"
>>> print(b)
```

Type matters

- Python knows what type everything is
- Some operations are prohibited on certain types
- You cannot “add 1” to a string
- We can check the type of something using function type()

Types of numbers

- Numbers in Python generally have two types:

✓Integers: 1, 2, 100, -20394209

✓Floating point numbers: 2.5, 3.7, 11.32309, -30.999

- There are other number types, which are variations on float and integer (e.g., complex numbers)

Type can change

- The type of a variable can be dynamically changed
- A variable's type is determined by the value that is last assigned to the variable

```
>>> x = 7 * 3 * 2
>>> y = "is the answer to the ultimate question of life"
>>> print(x, y)          # Check what x and y are.
42 is the answer to the ultimate question of life
>>> x, y                 # Quicker way to check x and y.
(42, 'is the answer to the ultimate question of life')
>>> type(x), type(y)     # Check types of x and y.
(<class 'int'>, <class 'str'>)
>>> # Set x and y to new values.
>>> x = x + 3.14159
>>> y = 1232121321312312312312 * 98734237892374382 97
>>> print(x, y)          # Check what x and y are.
45.14159 121652559650716498712086836307354934126 64
>>> type(x), type(y)     # Check types of x and y.
(<class 'float'>, <class 'int'>)
```

Type conversion

- When an expression contains both integer and float, integers will be converted into float implicitly

- You can control this using functions int() and float()

- Example:

```
>>> print(float(99)/100)

>>> i=42
>>> type(i)

>>> f=float(i)
>>> print(f)

>>> type(f)

>>> print(1+2*float(3)/4-5)
```

String conversions

- You can also use int() and float() to convert strings into numbers

- You will get an error if the string contains characters other than numbers

Converting numbers into string

- We can convert numbers into string using function str()

```
>>> str(5)                  # Convert int to a string.
'5'
>>> str(1 + 10 + 100)       # Convert int expression to a string.
'111'
>>> str(-12.34)             # Convert float to a string.
'-12.34'
>>> str("Hello World!")     # str() accepts string arguments.
'Hello World!'
>>> str(divmod(14, 9))      # Convert tuple to a string.
'(1, 5)'
>>> x = 42
>>> str(x)                  # Convert int variable to a string.
'42'
```

Sentences or lines

```
>>> x=2
>>> x=x+2
>>> print(x)
4
>>>
```

Assignment statement

Assignment with expressions

Print statement (output statement)

Expressions

Types of Expressions

An expression describes a computation and evaluates to a value

$18 + 45$

$f(x)$

$\frac{6}{23}$

$\binom{45}{18}$

$\sqrt{2323478}$

2^{100}

$\sin \pi$

$\sum_{i=1}^{100} i$

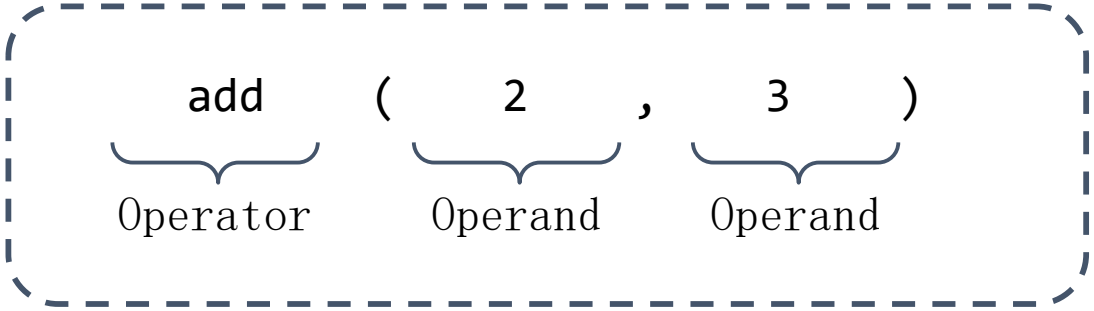
$|-1253|$

$\lim_{x \to \infty} \frac{1}{x}$

$7 \bmod 2$

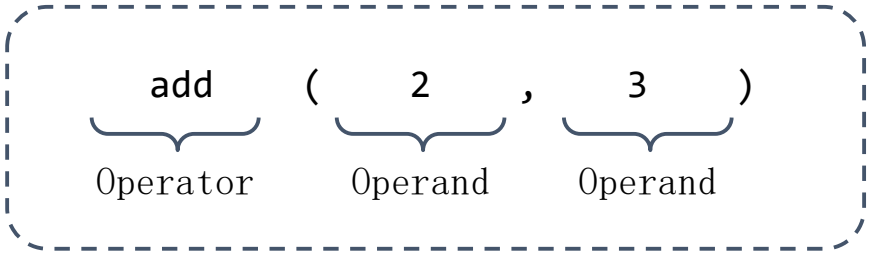
$\log_2 1024$

Anatomy of a Call Expression



Operators and operands are also expressions

Evaluation of a Call Expression



1. Evaluate
    a. Evaluate the operator subexpression
    b. Evaluate each operand subexpression
2. Apply
    a. Apply the value of the operator subexpression to the values of the operand subexpression

```
add(add(6, mul(4, 6)), mul(3, 5))
```

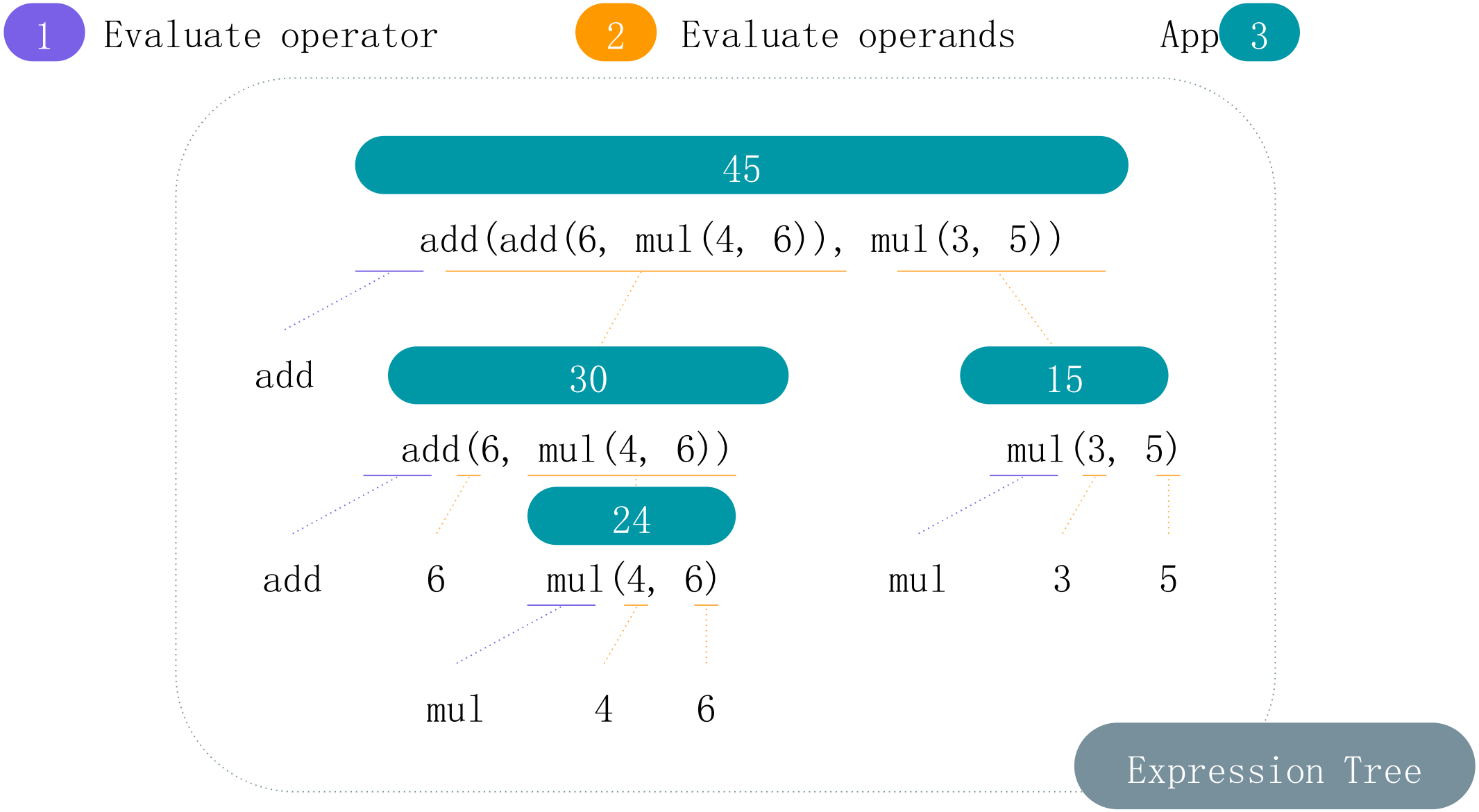
Humans

We like to inside inside-out

```
add(add(6, mul(4, 6)), mul(3, 5))
add(add(6,    24    ), mul(3, 5))
add(add(6,    24    ), mul(3, 5))
add(       30       , mul(3, 5))
add(       30       , mul(3, 5))
add(       30       ,     15   )
add(       30       ,     15   )
                  45
```

Python can’t jump around in the same way we do

Nested Call Expression

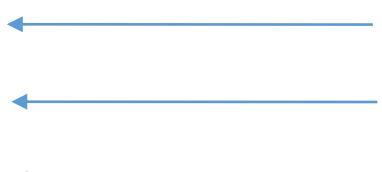


Outline

- Assignment statement
  - Assignment statement, Cascaded assignment, Simultaneous assignment
- Operators and Precedence
- Floor Division and Divmod()
- Augmented assignment
- Flow
  - Sequential flow, conditional flow, repeated flow
- Input function, Comments, *+* and “*” of string, Print()

Assignment statement

```
>>> x=2
>>> x=x+2
>>> print(x)
4
>>>
```



Assignment statement (赋值语句)

Assignment with expressions

Print statement (output statement)

In **Python**, = means *assign the value on the right to the variable on the left*.

Assignment statement

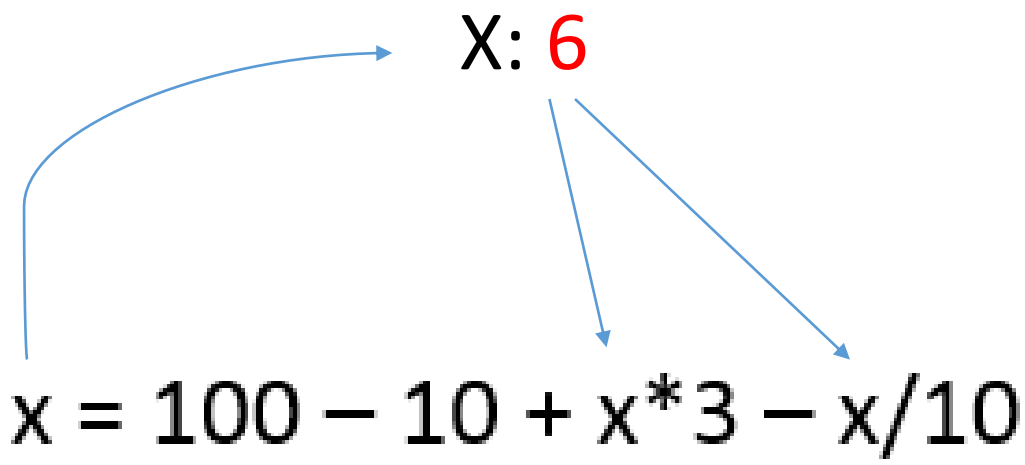
- We assign a value to a variable using the assignment operator (=)

- An assignment statement consists of an expression on the right hand side, and a variable to store the result

Example: x = 100 – 10 + x*3 – x/10

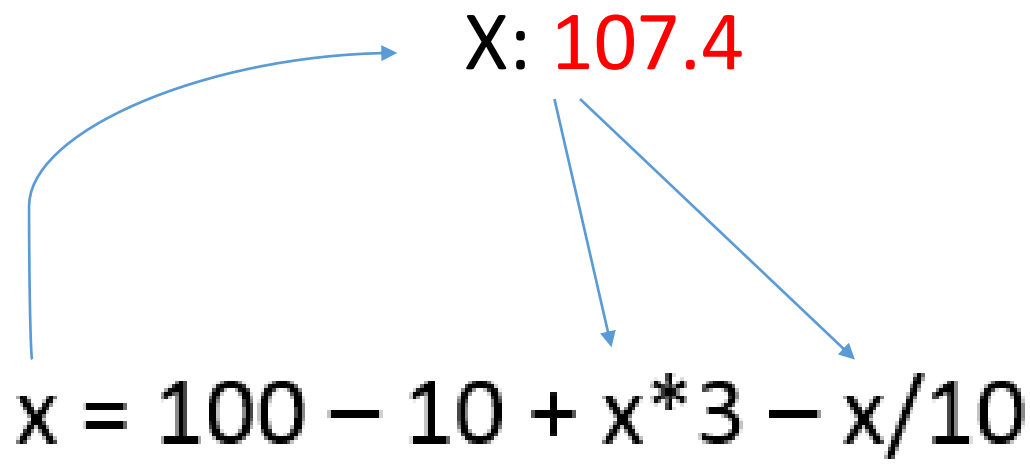
Assignment statement

- There is a location in the memory for x
- Whenever the value of x is needed, it can be retrieved from the memory
- After the expression is evaluated, the result will be put back into x



Assignment statement

- There is a location in the memory for x
- Whenever the value of x is needed, it can be retrieved from the memory
- After the expression is evaluated, the result will be put back into x

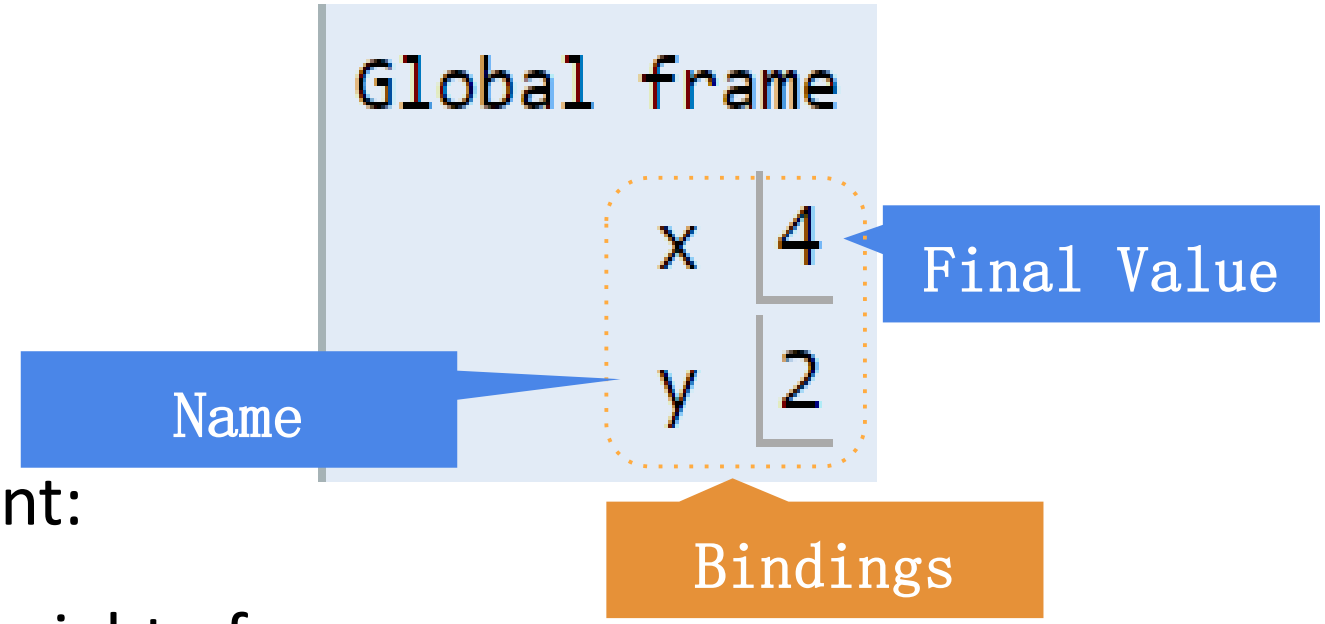


Visualizing Assignment

Demo

Names are bound to **values** in an environment

```
1  x = 1
2  y = 2
3  x = y * 2
```



To execute an assignment statement:

1. **Evaluate** the expression to the right of =.
2. **Bind** the value of the expression to the name to the left of = in the current environment.

Cascaded assignment

- We can set multiple variables into the same value using a single assignment statement

Example

```
>>> z = y = x = 2 + 7 + 2
>>> x, y, z
(11, 11, 11)
```

Practice

- Write a program to exchange the values of two variables.

Simultaneous assignment

- The values of two variables can be exchanged using simultaneous assignment

Example

```
>>> c = "deepSecret"           # Set current password.
>>> o = "you'll never guess"   # Set old password.
>>> c, o                       # See what passwords are.
('deepSecret', "you'll never guess")
>>> c, o = o, c                # Exchange the passwords.
```

Bad use of simultaneous assignment

```
>>> # A bad use of simultaneous assignment.
>>> x, y = (45 + 34) / (21 - 4), 56 * 57 * 58 * 59
>>> x, y
(4.647058823529412, 10923024)
>>> # A better way to set the values of x and y.
>>> x = (45 + 34) / (21 - 4)
>>> y = 56 * 57 * 58 * 59
>>> x, y
(4.647058823529412, 10923024)
```

Order evaluation

- When we put operators together, Python needs to know which one to do first

- This is called “operator precedence”

- Which operator “takes precedence” over the others

Example: X = 1 + 2*3 – 4 / 5 ** 6

Numeric expression and operators

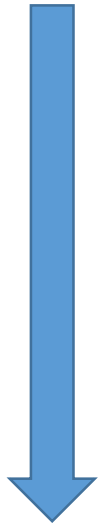
- We use some keys we have on the keyboard to denote the classic math operators
- Asterisk (*) is the multiplication operator
- Double asterisk (**) is used to denote Exponentiation (raise to a power)

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Operator precedence rules

- **Highest to lowest precedence rule**

✓ Parenthesis are always with highest priority
✓ Power/ Exponentiation
✓ Multiplication, division and remainder
✓ Addition and subtraction
✓ Left to right



Operator precedence

Example: x = 1+2**3/4*5

Floor division

```
>>> time = 257                # Time in seconds.
>>> minutes = time // 60 # Number of complete minutes in time.
>>> print("There are", minutes, "complete minutes in", time, "seconds.")
There are 4 complete minutes in 257 seconds.
>>> 143 // 25
5
>>> 143.4 // 25
5.0
>>> 9 // 2.5
3.0
```

Floor division

-2 // 3 = ?

divmod()

```
>>> time = 257          # Initialize time.
>>> SEC_PER_MIN = 60 # Use a "named constant" for 60.
>>> divmod(time, SEC_PER_MIN)   # See what divmod() returns.
(4, 17)
>>> # Use simulataneous assignment to obtain minutes and seconds.
>>> minutes, seconds = divmod(time, SEC_PER_MIN)
>>> # Attempt to display the minutes and seconds in "standard" form.
>>> print(minutes, ":", seconds)
4 : 17
>>> # Successful attempt to display time "standard" form.
>>> print(minutes, ":", seconds, sep="")
4:17
>>> # Obtain number of quarters and leftover change in 143 pennies.
>>> quarters, cents = divmod(143, 25)
>>> quarters, cents
(5, 18)
```

Augmented assignment

```
>>> x = 22
>>> x = x + 7
```

Augmented assignment

- The general form of augmented assignment looks like

```
<lvalue> <op>= <expression>
```

Example

```
>>> x = 22          # Initialize x to 22.
>>> x += 7          # Equivalent to: x = x + 7
>>> x
29
>>> x -= 2 * 7      # Equivalent to: x = x - (2 * 7)
>>> x
15
```

Personal tips

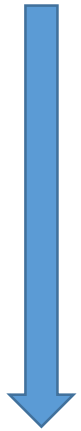
- Use parenthesis
- Keep mathematical expressions simple so that they are easy to understand
- Break up long series of math expressions to make them easy to understand

Program steps or program flow

- Like a recipe, a program is a sequence of steps to be done in pre-determined order
- Some steps are conditional, i.e. they may be skipped
- Sometimes, we will repeat some steps
- Sometimes, we store a set of steps to be used over and over again in future as needed

Sequential flow

Execute sequentially



```
>>> x=2
>>> print(x)
2
>>> x=x*10
>>> print(x)
20
>>>
```

Outputs

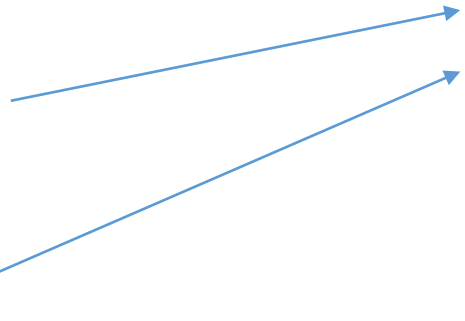
- When a program is running, it flows from one step to the next
- We as programmers, set up “paths” for the program to follow

Conditional flow

Program

```
x=5
if x<10:
    print("smaller")
if x>20:
    print("bigger")
print("finished")
```

Outputs

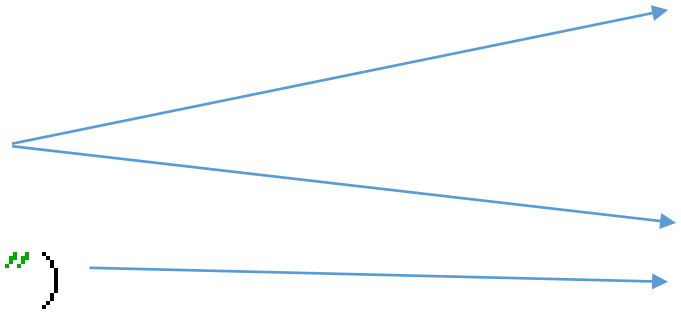


Repeated flow

Program

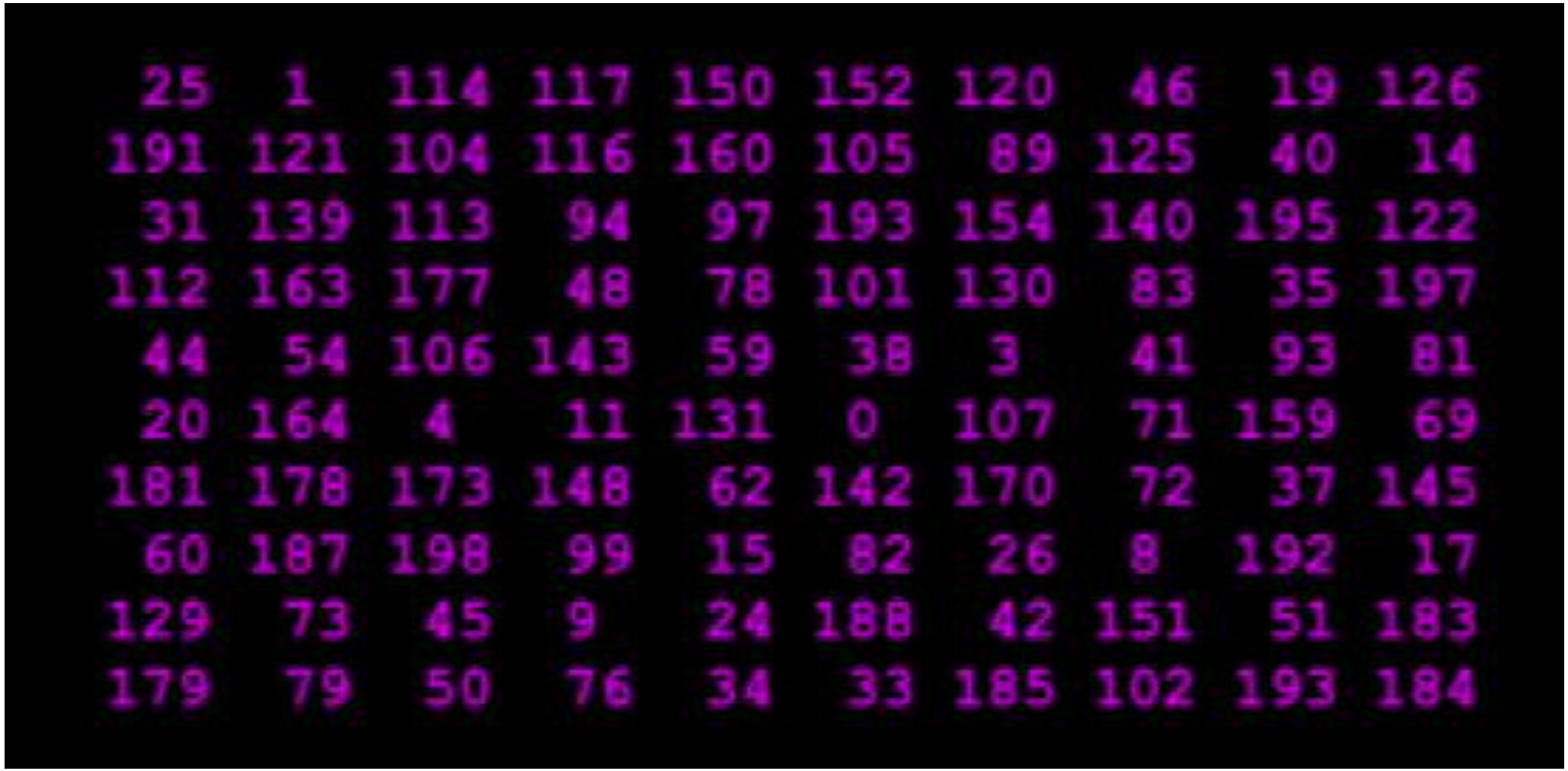
```
n=5
while n>0:
    print(n)
    n = n - 1
print("Finish")
```

Outputs

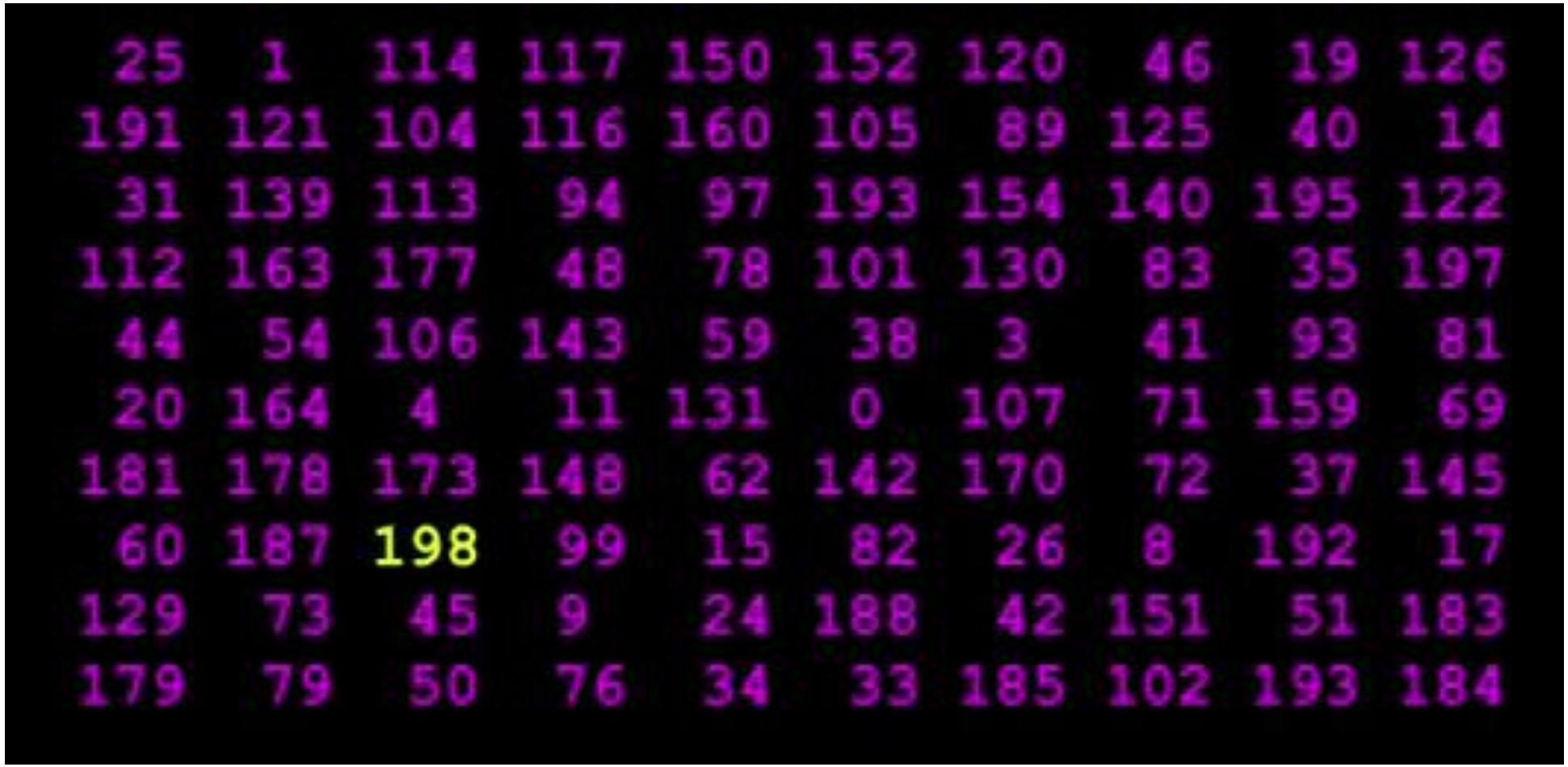


- Loops (repeated steps) have iterative variables that change each time through a loop
- Often these iterative variables go through a sequence of numbers

What the largest number is?



What the largest number is?



User input

- We can instruct Python to stop and take user inputs using function input()

- The input() function returns a string

Practice

- The BMI (body mass index) of a human can be calculated using the following equation:

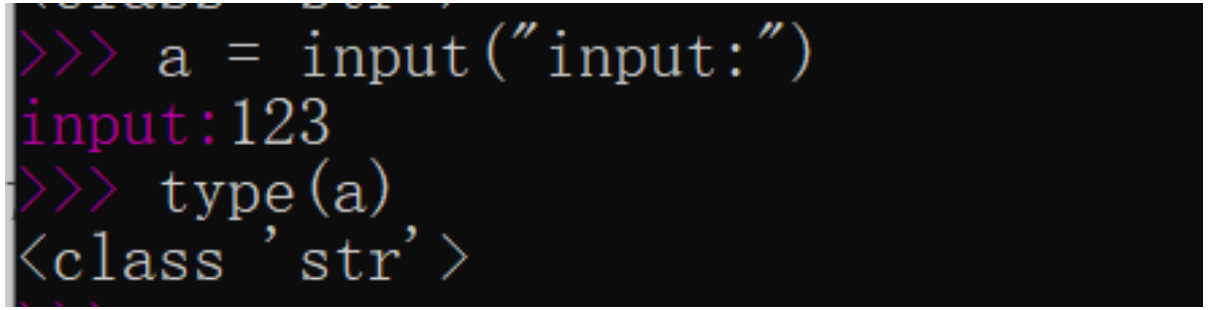
$$BMI = weight\ (kg) \div height\ \hat{}\ 2\ (m)$$

- Write a program to input a user's weight and height, and then output his BMI

Converting user input

- If we want to read a number using input(), we must then convert the input into a number using int() or float()

- Later we will deal with bad input data



Comments

- Anything after a “#” is ignored by Python

- Why comment?

✓Describe what is going to happen in a sequence of code

✓Document who wrote the code and other important information

✓Turn off a line of code – usually temporarily

String operations

- Some operators apply to strings

✓“+”: concatenation

✓“*”: multiple concatenation

- Python knows whether it is dealing with a number or a string

Practice

- Write a program to instruct the user to input two of his friends' names, and then output a sentence "I am the friend of XX and XX."

Output using Print()

```
>>> print(42, "42") # An int and a str that looks like an int.
42 42
>>> print('3.14')    # A str that looks like a float.
3.14
>>> print(3.14)      # A float.
3.14
```

More details on print()

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Examples

```
>>> print("I", "like", "AI")
I like AI
>>> print("I", "like", "AI", sep='')
IlikeAI
>>> print("I", "like", "AI", sep=',')
I,like,AI
```

Example

```
print("Test line 1")
print("Test line 2")

print("Test line 1", end = " ")
print("Test line 2")

print("Test line 1", end = "---")
print("Test line 2")
```

A powerful function - eval()

- The eval() function takes a string argument and evaluates that string as a Python expression, i.e., just as if the programmer had directly entered the expression as code
- The function returns the result of that expression
- Eval() gives the programmers the flexibility to determine what to execute at run-time
- one should be cautious about using it in situations where users could potentially cause problems with “inappropriate” input

Example

```
>>> string = "5 + 12" # Create a string.
>>> print(string)     # Print the string.
5 + 12
>>> eval(string)      # Evaluate the string.
17
>>> print(string, "=", eval(string))
5 + 12 = 17
>>> eval("print('Hello World!')") # Can call functions from eval().
Hello World!
>>> # Using eval() we can accept all kinds of input...
>>> age = eval(input("Enter your age: "))
Enter your age: 57.5
>>> age
57.5
>>> age = eval(input("Enter your age: "))
Enter your age: 57
>>> age
57
>>> age = eval(input("Enter your age: "))
Enter your age: 40 + 17 + 0.5
>>> age
57.5
```

Example

```
>>> eval("10, 32")          # String with comma-separated values.
(10, 32)
>>> x, y = eval("10, 20 + 12") # Use simultaneous assignment.
>>> x, y
(10, 32)
>>> # Prompt for multiple values.  Must separate values with a comma.
>>> x, y = eval(input("Enter x and y: "))
Enter x and y: 5 * 2, 32
>>> x, y
(10, 32)
```

Other functions

How to get a substring from a string?

Drawing environment diagrams

Python Tutor (tutor.cs61a.org)

- Useful for quick visualization or for environment diagram questions