



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

Introduction to Computer Science: Programming Methodology

Lecture 2 Python Basics

Prof. Yunming XIAO
School of Data Science

Parselmouth

Parseltongue is the language of serpents and those who can converse with them. An individual who can speak Parseltongue is known as a **Parselmouth**. It is very uncommon skill, and may be hereditary. Nearly all known Parselmouths are descended from Salazar Slytherin.

<https://harrypotter.wikia.com/wiki/Parseltongue>



Pythonista

Python is the language of Python interpreter and those who can converse with them. An individual who can speak Python is known as a **Pythonista**. It is very uncommon skill, and may be hereditary. Nearly all known Pythonistas use software initially developed by Guido van Rossum



Most popular programming language

Aug 2025	Aug 2024	Change	Programming Language	Ratings	Change
1	1		 Python	26.14%	+8.10%
2	2		 C++	9.18%	-0.86%
3	3		 C	9.03%	-0.15%
4	4		 Java	8.59%	-0.58%
5	5		 C#	5.52%	-0.87%
6	6		 JavaScript	3.15%	-0.76%
7	8		 Visual Basic	2.33%	+0.15%
8	9		 Go	2.11%	+0.08%
9	25		 Perl	2.08%	+1.17%
10	12		 Delphi/Object Pascal	1.82%	+0.19%

<https://www.tiobe.com/tiobe-index/>

Early learners: 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** (without AI)
- It is **easier** for you to learn Python than for the computer to learn human language

Installing Python

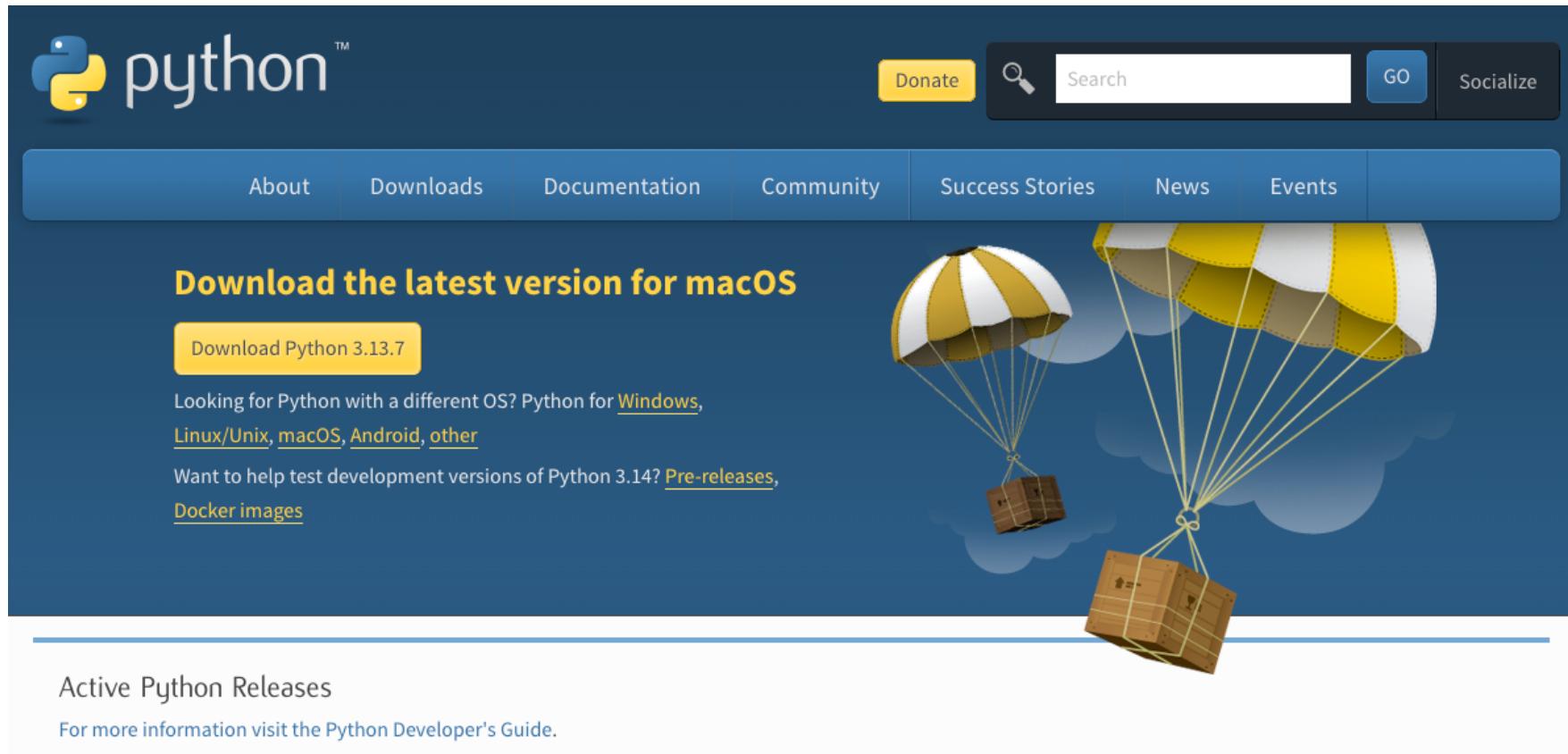
The screenshot shows the official Python website at <https://www.python.org>. The top navigation bar includes links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is the Python logo and a search bar with a magnifying glass icon. A secondary navigation bar below the header includes links for About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area features a code snippet in a terminal window:

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
    >>>     a, b = 0, 1
    >>>     while a < n:
    >>>         print(a, end=' ')
    >>>         a, b = b, a+b
    >>>     print()
    >>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

To the right of the code, a section titled "Functions Defined" explains the core of extensible programming. It states that Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. A link "More about defining functions in Python 3" is provided. At the bottom of the page, a call-to-action reads: "Python is a programming language that lets you work quickly and integrate systems more effectively. [» Learn More](#)".

<https://www.python.org>

Installing Python



The screenshot shows the Python.org website's download page. At the top, there's a dark blue header with the Python logo and the word "python" in white. To the right are buttons for "Donate", a search bar with a magnifying glass icon, and "GO". Below the header is a navigation menu with links for "About", "Downloads", "Documentation", "Community", "Success Stories", "News", and "Events". The main content area has a dark blue background. In the center, the text "Download the latest version for macOS" is displayed in yellow. Below it is a yellow button labeled "Download Python 3.13.7". Further down, there's text for other operating systems and links for pre-releases and Docker images. To the right of the text is a cartoon illustration of two boxes descending from the sky on parachutes. At the bottom of the page, there's a section titled "Active Python Releases" with a link to the "Python Developer's Guide".

python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

Download the latest version for macOS

[Download Python 3.13.7](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/Unix](#), [macOS](#), [Android](#), [other](#)

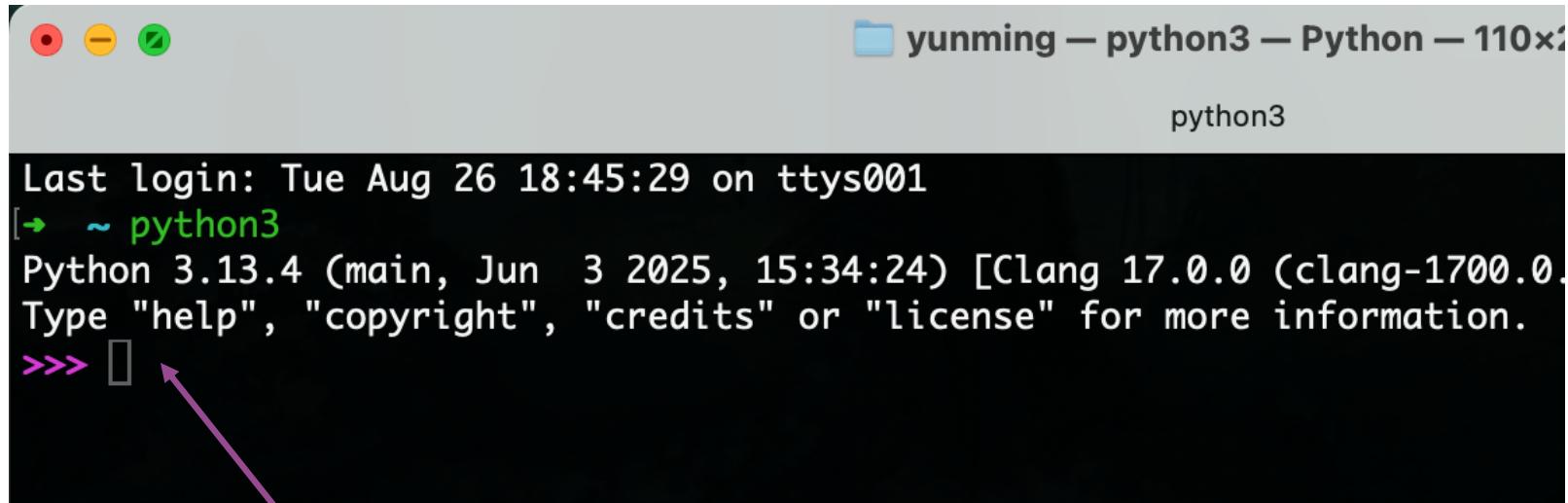
Want to help test development versions of Python 3.14? [Pre-releases](#), [Docker images](#)

Active Python Releases

For more information visit the [Python Developer's Guide](#).

<https://www.python.org/download>

Python shell



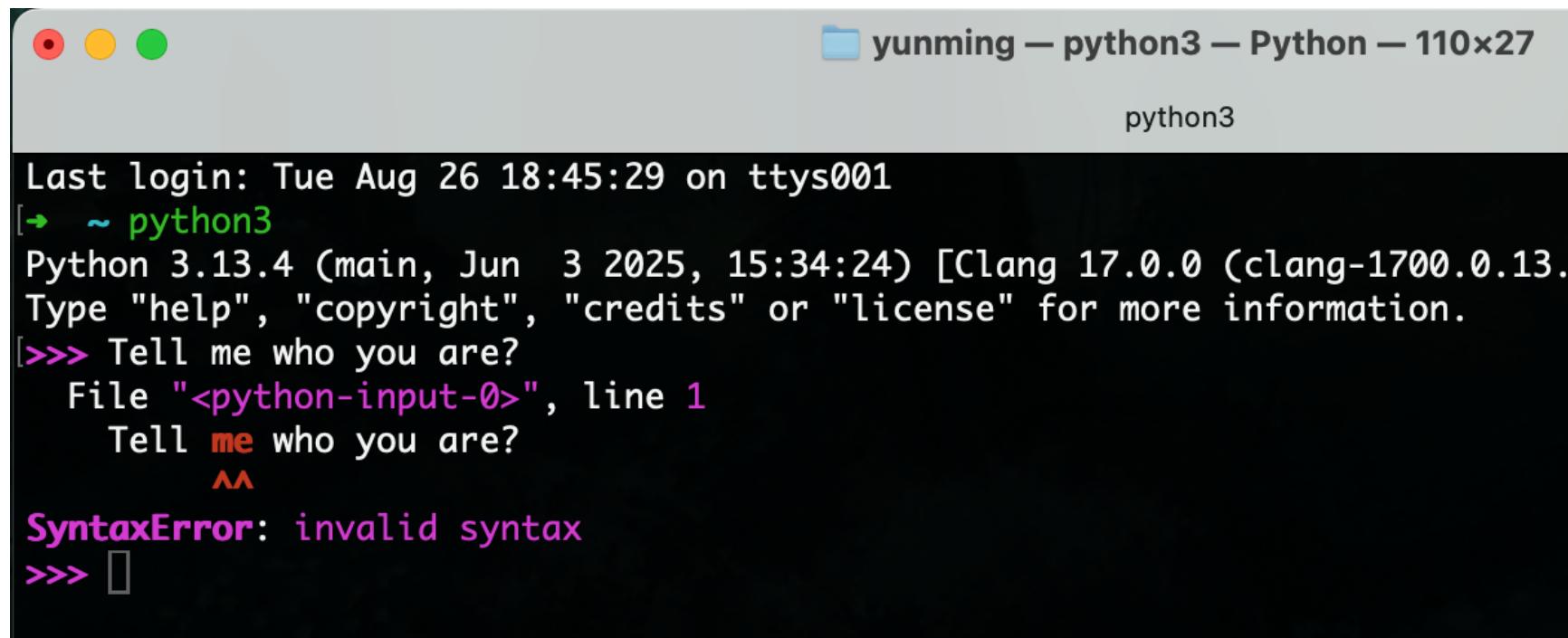
A screenshot of a macOS terminal window titled "yunming — python3 — Python — 110x24". The window shows a Python shell session:

```
Last login: Tue Aug 26 18:45:29 on ttys001
[→ ~ python3
Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.0.2.30)]
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

A purple arrow points from the text "What is next?" at the bottom left towards the command line prompt in the terminal window.

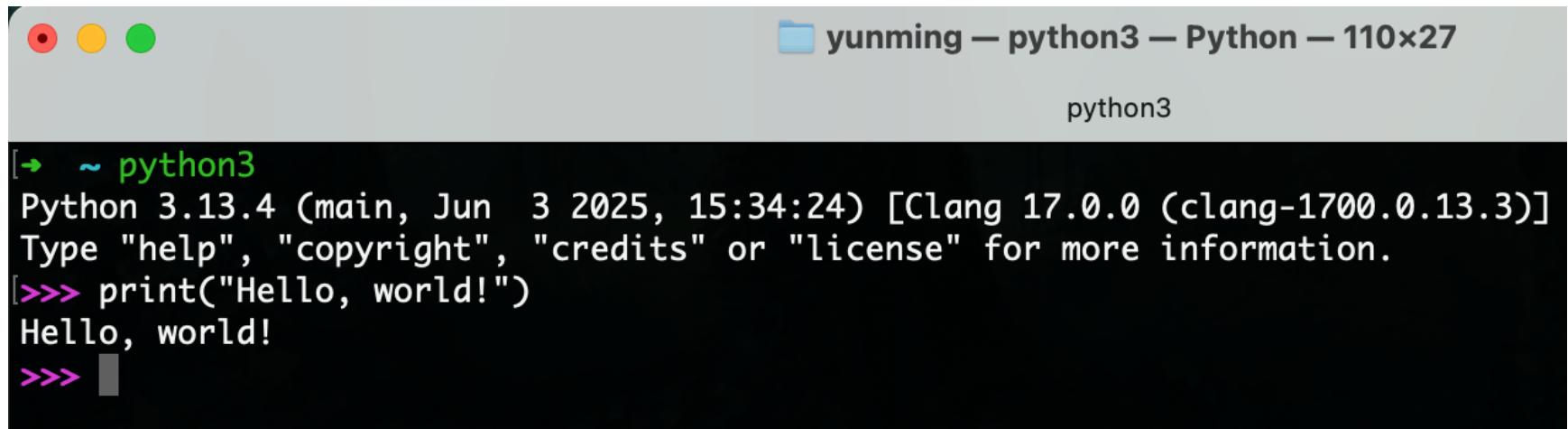
What is next?

Syntax error



Last login: Tue Aug 26 18:45:29 on ttys001
[→ ~ python3
Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.0.13.
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
[>>>]

Hello, world!

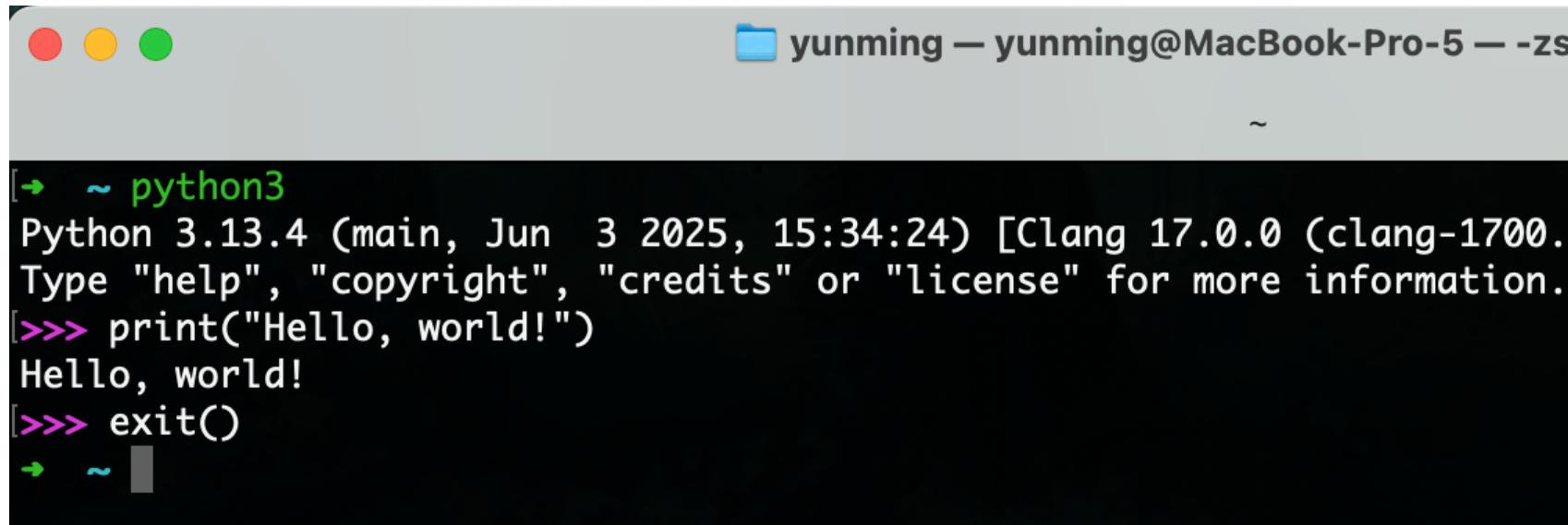


The screenshot shows a macOS terminal window with a dark theme. The title bar reads "yunming — python3 — Python — 110x27". The window contains the following text:

```
[→ ~ python3
Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.0.13.3)]
Type "help", "copyright", "credits" or "license" for more information.
[>>> print("Hello, world!")
Hello, world!
>>> ]
```

- You must say something that Python interpreter can understand!!
- Print() is a **function** in Python

Exit()



A screenshot of a macOS terminal window. The title bar shows the user 'yunming' and the host 'yunming@MacBook-Pro-5'. The command line starts with a red arrow icon followed by the text '~ python3'. The Python interpreter then displays its version and build information: 'Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.0.4.4)]. Type "help", "copyright", "credits" or "license" for more information.' The user then types 'print("Hello, world!")' and presses Enter, resulting in the output 'Hello, world!'. Finally, the user types 'exit()' and presses Enter, which exits the Python interpreter back to the terminal prompt.

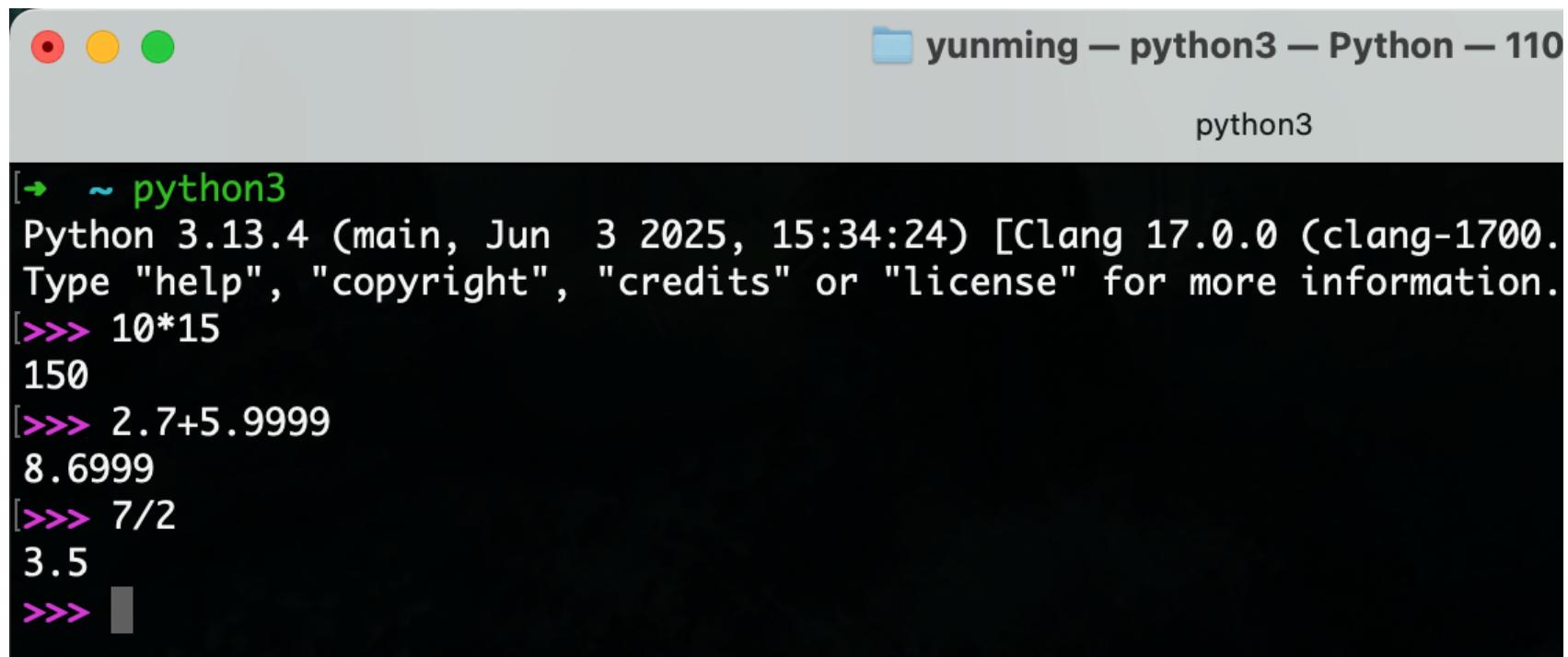
```
[→ ~ python3
Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.0.4.4)]
Type "help", "copyright", "credits" or "license" for more information.
[>>> print("Hello, world!")
Hello, world!
[>>> exit()
[→ ~
```

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

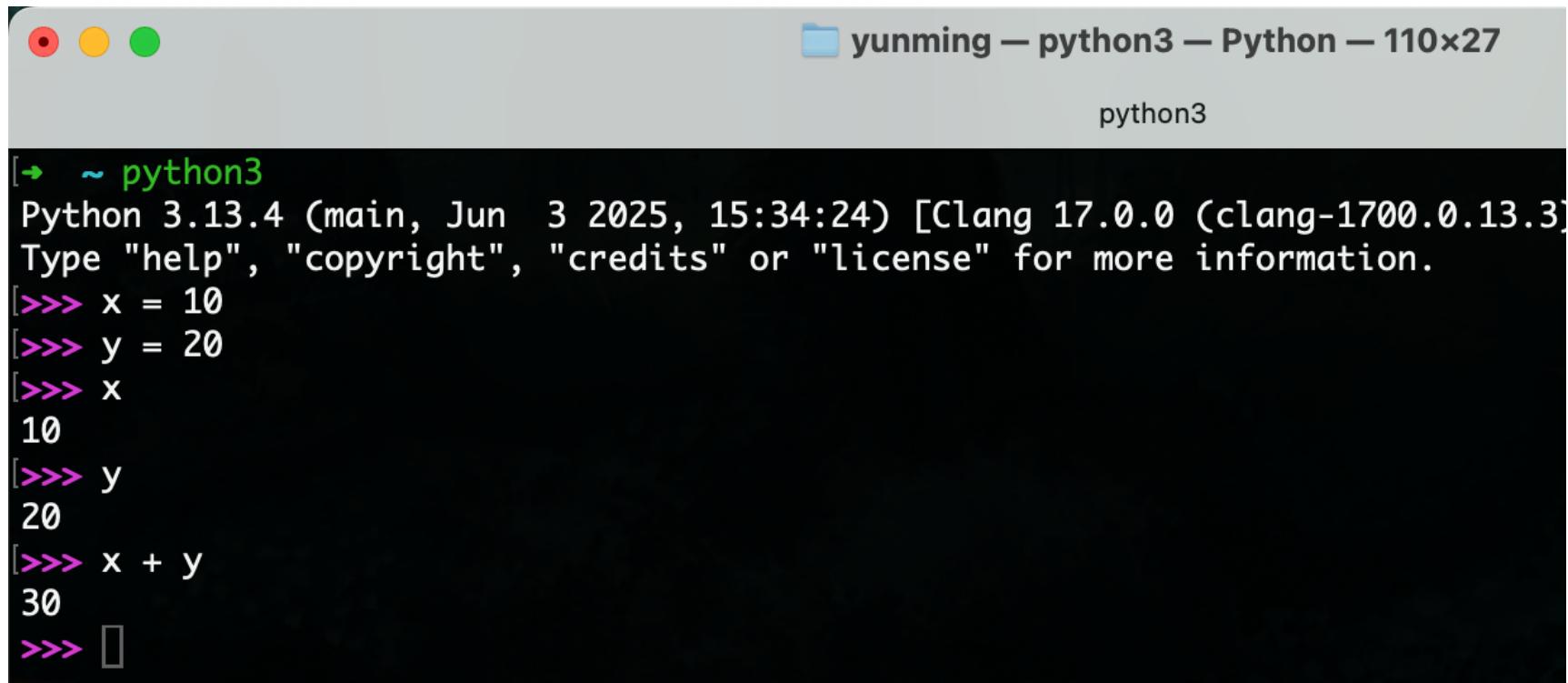
Use Python as a calculator



A screenshot of a macOS terminal window. The title bar shows the path: "yunming — python3 — Python — 110" and the application name "python3". The terminal window itself has a dark background and contains the following text:

```
[→ ~ python3
Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.
Type "help", "copyright", "credits" or "license" for more information.
[>>> 10*15
150
[>>> 2.7+5.9999
8.6999
[>>> 7/2
3.5
[>>> ]
```

Variables



The screenshot shows a macOS terminal window with the following details:

- Title Bar:** yunming — python3 — Python — 110x27
- Icon:** A blue folder icon.
- Text Area:** Shows a Python 3 session:

```
[→ ~ python3
Python 3.13.4 (main, Jun 3 2025, 15:34:24) [Clang 17.0.0 (clang-1700.0.13.3)]
Type "help", "copyright", "credits" or "license" for more information.
[>>> x = 10
[>>> y = 20
[>>> x
10
[>>> y
20
[>>> x + y
30
>>> ]]
```

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		

Sentences or lines

```
>>> x = 2           ← Assignment statement  
>>> x = x + 2       ← Assignment with expressions  
>>> print(x)        ← Print statement (output statement)  
4  
>>> █
```

Programming scripts

- Interactive Python is good for experiments and programs of 3-4 lines long
- Most programs are much longer, so we need 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

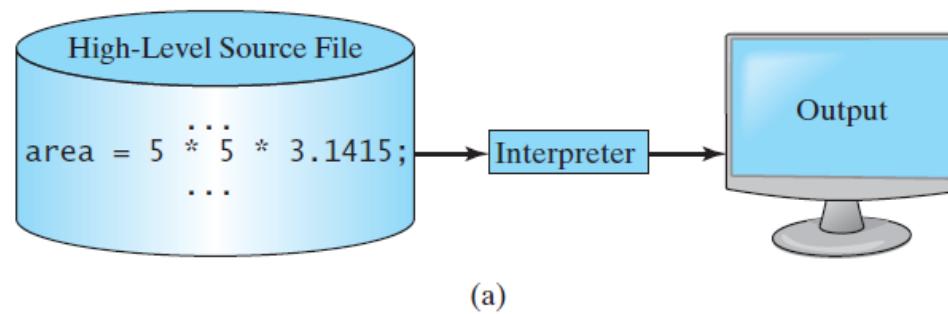
Interactive vs. script

- **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

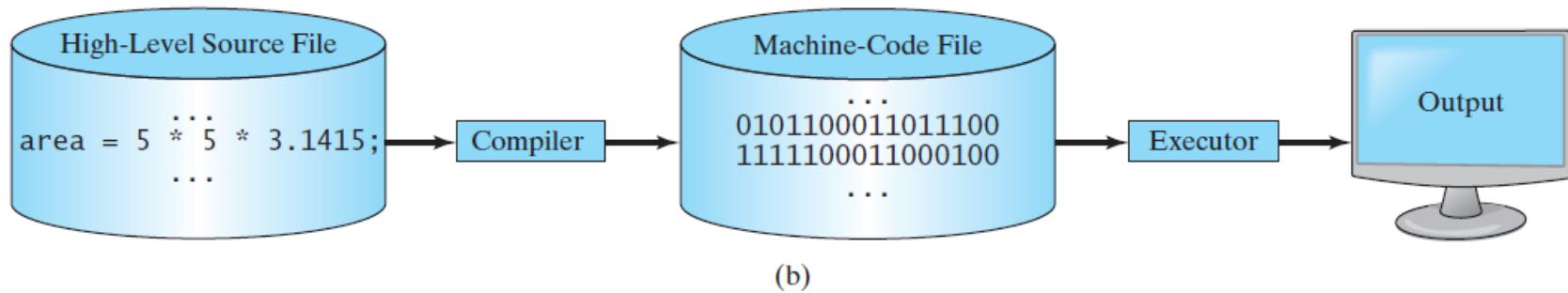
Interpreter vs. compiler

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



(a)



(b)

Example of compiler

- For C programs, they need to be compiled first using `gcc` before execution

```
[→ code ls lecture2-compiler.c
lecture2-compiler.c
[→ code gcc -o lecture2-compiler lecture2-compiler.c
[→ code ls
lecture2-compiler
lecture2-compiler.c
lecture2-compiler.cpp      lecture2-conditional-flow.py
                           lecture2-divmod.py
                           lecture2-floor-division.py
[→ code ./lecture2-compiler
Hello, World!
[→ code █
```

Example of interpreter

- For Python programs, they **don't** need to be compiled first before execution
- By calling `python3`, it will execute the program with an interpreter

```
[→ code ls lecture2-hello-world.py
lecture2-hello-world.py
[→ code cat lecture2-hello-world.py
print("Hello, World!")
[→ code python3 lecture2-hello-world.py
Hello, World!
→ code ]]
```

Does Python use interpreter or compiler?

- By default, Python programs are executed by an interpreter
- But it is possible to compile Python programs
 - (numba, cython, etc)

What's the difference?

- Compiling first allows the program to run more efficiently, i.e., faster

```
(venv) ➔ code python3 lecture2-hardwork.py
Computing sum of squares for N=3000000
Result: 8999995500000500000
Time elapsed: 0.160 seconds
```

```
(venv) ➔ code gcc -o lecture2-hardwork-c lecture2-hardwork-c.c
(venv) ➔ code ./lecture2-hardwork-c
C result: 8999995500000500000
Time: 0.010057 s
(venv) ➔ code
```

Program steps or program flow

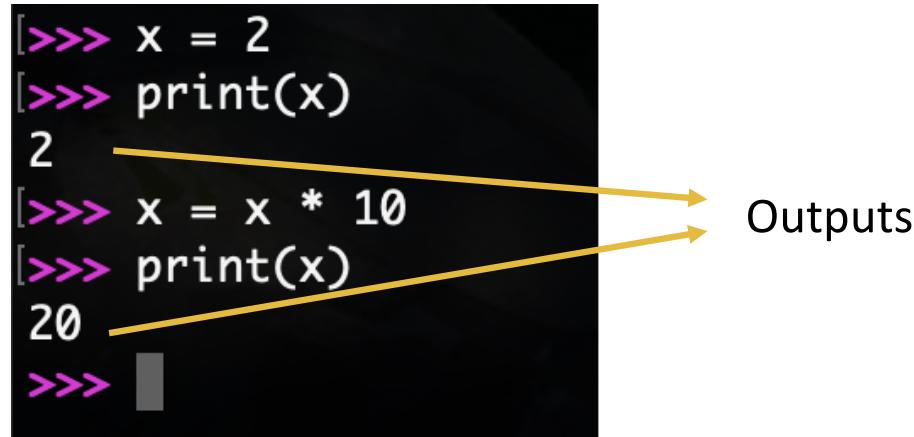
- Like a recipe, a program is **a sequence of steps** to be done in **pre-determined order**
- Some steps are **conditional**, i.e. they may be skipped
- Sometimes, we will **repeat** some steps
- Sometimes, we **store** a set of steps to be used over and over again in future as needed

Sequential flow

Execute sequentially



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



- 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

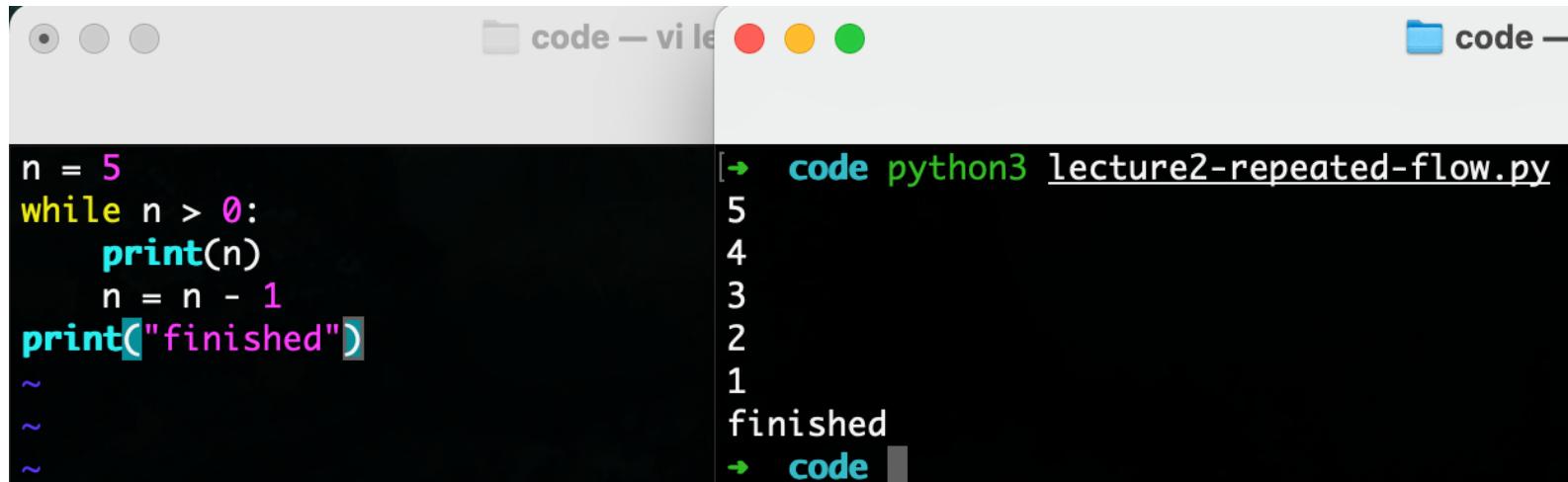
```
x = 5
if x < 10:
    print("smaller")
elif x > 10:
    print("bigger")
else:
    print("equal")
print("finished")
```

Outputs

```
[→ code python3 lecture2-conditional-flow.py
smaller
finished
→ code ]
```

Repeated flow

Program



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

The terminal window shows the command `code python3 lecture2-repeated-flow.py` followed by the output:
5
4
3
2
1
finished

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 is the largest number?

25	1	114	117	150	152	120	46	19	126
191	121	104	116	160	105	89	125	40	14
31	139	113	94	97	193	154	140	195	122
112	163	177	48	78	101	130	83	35	197
44	54	106	143	59	38	3	41	93	81
20	164	4	11	131	0	107	71	159	69
181	178	173	148	62	142	170	72	37	145
60	187	198	99	15	82	26	8	192	17
129	73	45	9	24	188	42	151	51	183
179	79	50	76	34	33	185	102	193	184

What is the largest number?

25	1	114	117	150	152	120	46	19	126
191	121	104	116	160	105	89	125	40	14
31	139	113	94	97	193	154	140	195	122
112	163	177	48	78	101	130	83	35	197
44	54	106	143	59	38	3	41	93	81
20	164	4	11	131	0	107	71	159	69
181	178	173	148	62	142	170	72	37	145
60	187	198	99	15	82	26	8	192	17
129	73	45	9	24	188	42	151	51	183
179	79	50	76	34	33	185	102	193	184

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 (")

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

Rules for naming variables in Python

- Must start with a letter or underscore _
- Can **only** contain letters, numbers and underscore
- Case **sensitive**
- **Good:** apple, car, myNumber123, _light
- **Bad:** 456aaa, #ab, var.12
- **Different:** apple, Apple, APPLE

Most common naming methods

- Use **meaningful words** as variable names
- Camel case
 - Start with a **lower-case letter**
 - Capitalize the **first letter** of each word
 - **Example:** bankAccountID, numOfCards, ...
- Snake case
 - Use **lower-case letters** in most cases
 - Use **underscores** to connect the words
 - **Example:** bank_account_id/bank_account_ID, num_cards, ...

Bad example: what is this code doing?

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print x1q3p9afd
```

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		

Identify different elements in the code

```
>>> x = 2           ← Assignment statement  
>>> x = x + 2     ← Assignment with expressions  
>>> print(x)      ← Print statement (output statement)  
4  
>>> █
```

Variable	Operator	Constant	Reserved words
x	= +	2	print

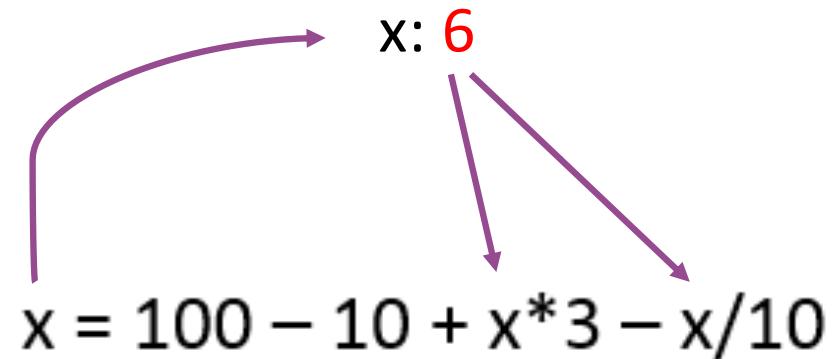
Assignment statement

- 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



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)
>>> |
```

Simultaneous assignment

- The values of two variables can be exchanged using simultaneous assignment
(not applicable in many other languages)

Example

```
[>>> current_pwd = "deep_secret"      1# Set current password
[>>> old_pwd = "you'll never know"   2# Set old password
[>>> current_pwd, old_pwd           3
('deep_secret', "you'll never know")
[>>> current_pwd, old_pwd = old_pwd, 4current_pwd # Exchange the passwords
[>>> current_pwd, old_pwd           5
("you'll never know", 'deep_secret')finished
```

Practice

- Write a program to exchange the values of two variables
without using simultaneous assignment

Bad use of simultaneous assignment

```
# A bad use of simultaneous assignment
x, y = (45 + 34) / (21 - 4), 56 * 57 * 58 * 59
print(x, y)

# A better way of setting the values of x and y
x = (45 + 34) / (21 - 4)
y = 56 * 57 * 58 * 59
print(x, y)
```

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`

Numerical 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
- Multiplication, division and remainder
- Addition and subtraction
- Left to right



Operator precedence rules

Example: $x = 1 + 2 * 3 - 4 / 5 ^\star 6$

Floor division

```
code — vi lecture2-floor-division.py — vi lecture2-floor-d
vi

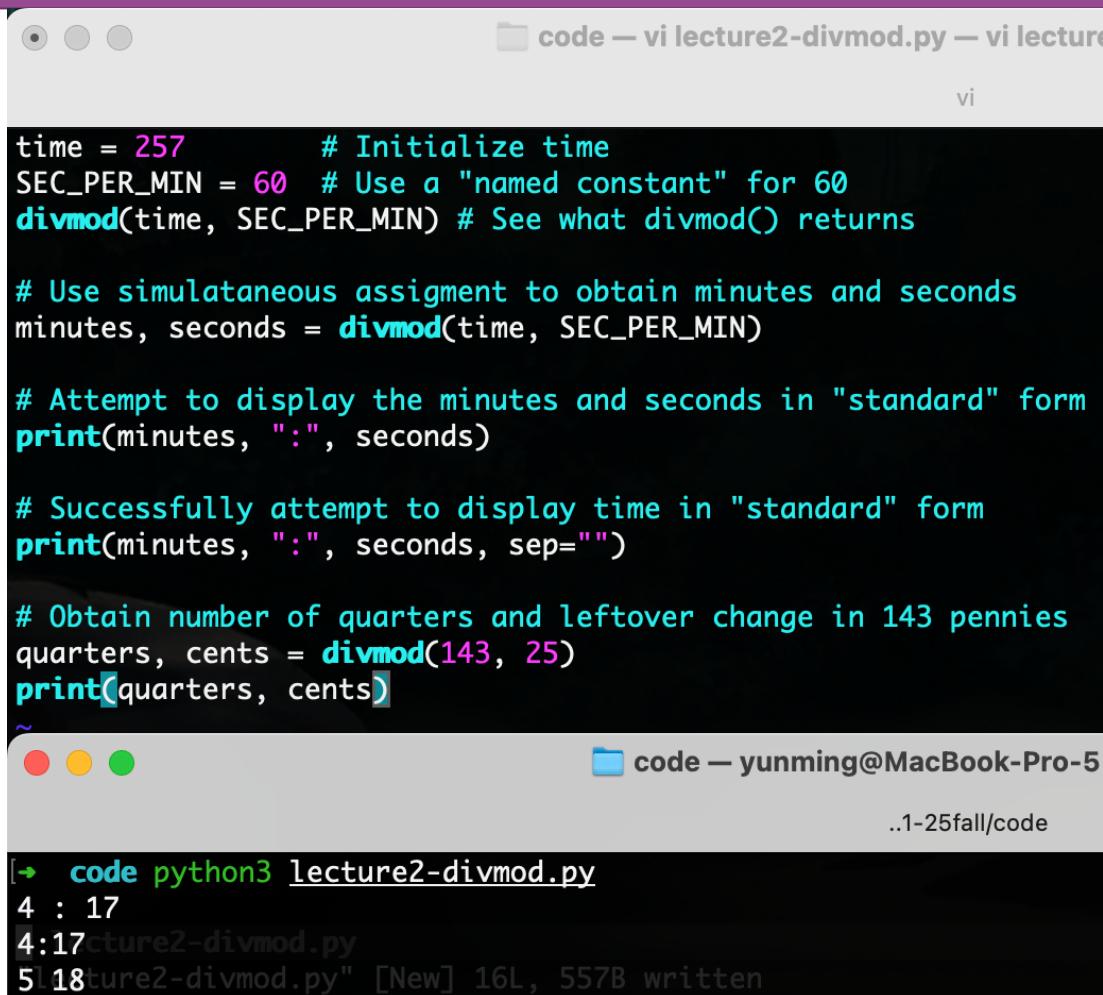
time = 257      # Time in seconds
minutes = time // 60  # Number of complete minutes in time
print("There are ", minutes, "complete minutes in", time, "seconds.")

print(143 // 25)
print(143.4 // 25)
print(9 // 2.5)
~
```

```
code — yunming@MacBook-Pro-5 — -zsh — '
..1-25fall/code

→ code python3 lecture2-floor-division.py
There are 4 complete minutes in 257 seconds.
5
5.0
3.0
→ code
```

Floor division



```
time = 257      # Initialize time
SEC_PER_MIN = 60 # Use a "named constant" for 60
divmod(time, SEC_PER_MIN) # See what divmod() returns

# Use simultaneous 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)

# Successfully attempt to display time in "standard" form
print(minutes, ":", seconds, sep="")

# Obtain number of quarters and leftover change in 143 pennies
quarters, cents = divmod(143, 25)
print(quarters, cents)
```

```
[→ code python3 lecture2-divmod.py
4 : 17
4:17lecture2-divmod.py
5 18lecture2-divmod.py" [New] 16L, 557B written
```

Floor division

- The general form of augmented assignment looks like

<lvalue> <op>= <expression>

Example

```
[>>> x = 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

Integer division in Python

- In Python 2, Integer division **truncates**
- Integer division produces floating point numbers in Python 3
- The **conversion** between integer and floating-point numbers is done **automatically** in Python 3
- Things change between Python 2 and Python 3

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)
300
[>>> print(type(a))
<class 'int'>
[>>>
[>>> b = "100" + "200"
[>>> print(b)
100200
[>>> print(type(b))
<class 'str'>
>>> ]
```

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()`

Type 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

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)
42 is the answer to the ultimate question of life
[>>> x, y
(42, 'is the answer to the ultimate question of life')
[>>> type(x), type(y)
(<class 'int'>, <class 'str'>)
[>>>
[>>> # Set x and y to new values
[>>> x = x + 3.14159
[>>> y = 123123321 * 987654321
[>>> print(x, y)
45.14159 121603280001520041
[>>> type(x), type(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()`

```
[>>> print(float(99)/100)
0.99
[>>> i = 42
[>>> type(i)
<class 'int'>
[>>> f = float(i)
[>>> print(f)
42.0
[>>> type(f)
<class 'float'>
[>>> print(1+2*float(3)/4-5)
-2.5
[>>> print(int(1+2*float(3)/4-5))
-2
[>>> ]
```

String conversion

- 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 argument
'Hello World'
[>>> str(divmod(14, 9)) # Convert tuple to a string
'(1, 5)'
[>>> x = 42
[>>> str(x)               # Convert int variable to a string
'42'
>>> ]
```

User input

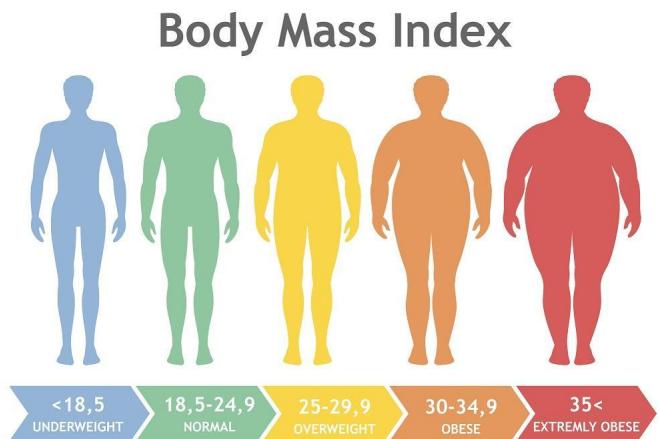
- We can instruct Python to stop and take user inputs using function `input()`
- The `input()` function returns a `string`

Type conversion

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

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

- 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()

- We can learn more about a function, such as print(), using function `help()`

```
[>>> help(print)

Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

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

Help on print line 1/13 (END) (press h for help or q to quit)]
```

Examples

```
[>>> print("I", "am", "Yunming Xiao")
I am Yunming Xiao
[>>> print("I", "am", "Yunming Xiao", sep="")
IamYunming Xiao
[>>> print("I", "am", "Yunming Xiao", sep=",")
I,am,Yunming Xiao
>>> ]
```

Examples

```
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")
```

```
[→ code python3 lecture2-print.py
```

```
Test line 1
Test line 2
Test line 1Test line 2
Test line 1---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!')") # We 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: 28
[>>> age
28
[>>> age = eval(input("Enter your age: "))
[Enter your age: 20 + 8 + 0.5
[>>> age
28.5
>>> ]
```

Example

```
[>>> eval("10, 32")    # String with comma-separated values
(10, 32)
[>>> x, y = eval("10, 20 + 12") # Use simultaneous assignment
[>>> x, y
(10, 32)
[>>> x, y = eval(input("Enter x and y: "))
[Enter x and y: 5 * 2, 32
[>>> x, y
(10, 32)
>>> ]
```

Thanks