Python Tutorial Study Notes

This blog post was translated by Mistral

Through previous learning, we have gained some understanding of Python. Now, based on the documentation on the official website, we will supplement some knowledge of Python.

Control Flow of Code

[Control of code execution order] class int

Note: The given Chinese text does not contain any meaningful information related to the Python code snippet provided. The code snippet is written in Python and does not contain any Chinese characters. Therefore, the Chinese text and its translation are not directly related. `type` function is useful, to print the type of an object. [Here is the English translation of the Chinese text:]

The Chinese text does not contain any readable content related to the Python code you provided. The Python code `print(type('a'))` is written in English and does not need to be translated. It will output `<class 'str'>` when run in a Python environment. `range` function is very useful.

For i in range(5): print(i, end='') two four

The Chinese text provided does not contain any readable text as it is just a sequence of numbers represented in Chinese characters. The Python code snippet is written in English and does not need to be translated. Two. Four.

Look at the definition of the `range` function.

```
class range(Sequence[int]):
    start: int
``` This is a class.


```python
print(list(range(5)))
```

The Chinese text does not provide any meaningful code or translation to English. The provided Python code is simply printing out the first five integers using the `range()` function and converting it to a list for printing.

However, based on the context, it seems like the Chinese text is describing that the text following it is a class definition. But without the actual class definition, it' s impossible to provide an accurate English translation. The Chinese text does not provide any Chinese characters related to the translation you' re

asking for. The text only shows Python code for generating a range from 0 to 5. The English translation of the code is:

```
range(0, 5)
```

This can be understood as a function call in Python that generates a sequence of numbers from 0 up to (but not including) 5. The result is a range object, which can be iterated or converted to a list.

Therefore, there is no English translation for the given Chinese text as it only shows Python code. continue.

Translation: Continue.

There is no Chinese text provided for translation in the given input. The Python and shell commands are unrelated to the translation request. Why. Look at the definition of `list`.

```
class list(MutableSequence[_T], Generic[_T]):
```

The definition of `list` is `list(MutableSequence[_T], Generic[_T]):`. And the definition of `range` is `class range(Sequence[int])`. `list` inherits from `MutableSequence`. `range` inherits from `Sequence`. We cannot directly translate the given Chinese text to English as it does not contain any meaningful sentences or phrases. The text seems to be related to Python code snippets and their explanations.

The first two lines of code define aliases for Python's `collections.abc.Sequence` and `collections.abc.MutableSequence` types using the `_alias()` function. These aliases are then assigned the integer value `1`. The purpose of these aliases is not clear in the given context.

The second part of the text mentions that these aliases might be related to why we can write `list(range(5))` in Python. However, without more context, it is difficult to make a definitive statement about this.

Therefore, there is no English translation to provide for the given Chinese text. Here's some additional knowledge about functions.

```
def fn(a = 3):
 print(a)


fn()
```

Function definition: A function is a block of organized, reusable code that is used to perform a single, related action. In this example, we define a function named "fn" with a default argument "a" set to 3. If no argument is passed when calling the function, it uses the default value. When the function is called without any arguments, it will print the number 3. This function `fn` takes one required integer argument `end` and one optional integer argument `start` with a default value of 1.

```
def fn(end, start=1):
  i = start
```

```
  s = 0
```

The given Chinese text does not contain any text to be translated. The provided Chinese characters represent an empty string in this context. The Python code snippet is in English and does not need translation. While i is less than end: Add i to s, then increase i by 1. Return s.

Print the result of calling this function with the argument 10.

Sum of numbers from 1 to 10.

```
This function calculates the sum of numbers from 1 to a given number (end). When the function is called
```

```
In Python:
```

```
```python
def fn(end: int, start=1):
```

In Shell script:

```
fn() {
  end=$1
  start=1
  # function body
}
```

Note: In the provided Python code snippet, the order of the parameters is changed to follow the requirement in the Chinese text, but the default value of `start` is still at the end. In the Shell script snippet, the parameters are passed to the function using positional arguments, so the order of the arguments in the function definition does not matter. “Warning: non-default argument ‘b’ follows default argument ‘/’ ”

“The given function definition is incorrect. The non-default argument ‘b’ should come before any default arguments. In this case, ‘/’ is a default argument, meaning if not passed, it already has a default value.” This is where `/` is used to separate argument types. There are two ways to pass arguments. One way is to pass them based on position, and the other way is to pass them based on specified keywords.

```
def fn(a, /, b):
 print(a + b)

fn(a=1, b=3)
```

In the English translation, the function definition remains the same, but the function call uses keyword
```

arguments instead of positional arguments. This way is incorrect. `a=1` means this is passed as a keyword argument. It is treated as a keyword argument here. While `b` is a positional argument.

Therefore, the correct way to write it would be:

```
fn(1, 3)
```

or

```
fn(3, a=1)
```

depending on the function definition. I. Function definition with positional and keyword arguments:

def f(pos	1, pos2,	pos_or_kwd, *kw	d1, **kwd2):	
| | | P	ositional	or keyword arg	ument | Keyword-onl	y argument |
|–|–|–	|———	—————————	———-|————	————|
| | | Pos	itional o	nly argument |	|	

Note that the use of `/` and `*` in function definition implicitly specifies the passing type for each argument. Therefore, arguments should be passed accordingly. The function definition below does not raise an error:

```
def fn(a, b=None):
  print(a + b)

fn(1, 3)
``` The function definition should be:

def fn(a, b, c):
  print(a + b + c)

And the function call should be:

fn(1, 2, 3) `fn` can only accept two positional arguments, but three were given.

```python
def fn(a, b, c=None):
  print(a + b + c)

fn(a=1, b=3) # Error: too many arguments for call
```

In the provided Python code, the function `fn` is defined to accept two positional arguments `a` and `b`, but an optional keyword argument `c`. However, when calling the function, three arguments `a`, `b`, and `c` are provided, which results in an error.

To fix this error, you can either remove the extra argument `c` when calling the function or make `c` a required keyword argument by removing the `=None` default value.

Here' s the corrected version:

```
def fn(a, b, c):
  print(a + b + c)


fn(a=1, b=3, c=4) # Corrected: all arguments are provided
```

Or, if you want to keep `c` as an optional keyword argument:

```
def fn(a, b, c=None):
  if c is not None:
    print(a + b + c)

fn(a=1, b=3) # Corrected: no value provided for c
fn(a=1, b=3, c=4) # Optional: c has a value
``` The function "fn" received some positional arguments passed as keyword arguments instead: 'a'. "mapp

# Python function definition:
# takes keyword arguments
def fn(**kwds):
  print(kwds)

# Function call with a dictionary argument:
fn(**{'a': 1})
```

The Chinese text translates to "mapping type of parameters" in English. The provided Python code demonstrates a function definition that accepts keyword arguments and prints them out, followed by a function call passing a dictionary as keyword arguments. The function `fn` takes in keyword arguments and prints the value associated with the key 'a'. In this specific case, the dictionary `d` contains the key 'a' with the value 1. Therefore, when `fn` is called with `**d` as an argument, the output will be `1`.

Here' s the English translation of the Chinese text without any Chinese characters or punctuation:

Function definition:
```

```
def fn(**kwds):
  print(kwds['a'])
```

Code snippet:

```
d = {'a': 1}
fn(**d)
```

Translation: Function definition:

```
def function_name(keyword_arguments):
  print(keyword_arguments['a'])
```

Code snippet:

```
d = {'a': 1}
function_name(**d)
``` The `**` symbol is used to unpack arguments.

```python
def fn(a, **kwds):
  print(kwds['a'])
```

This function `fn` takes one mandatory argument `a` and one keyword argument `**kwds`. The `**kwds` argument is used to unpack any keyword arguments passed to the function, and the `kwds['a']` expression is used to access the value associated with the key `'a'` in the keyword arguments dictionary. The function simply prints the value of this key-value pair.: The function call `fn(1, **d)` translates to `fn(1, a=1)` when unpacked, resulting in an error as both `a` arguments have the same value.

Error: TypeError: fn() got multiple values for argument ‘a’. Def function accepts keyword arguments, print value of key ‘a’:

Dict object d contains key ‘a’ mapped to value 1, Function call passes this dictionary as argument.

Function call: fn(d)

Function execution: Print value of key ‘a’ in dictionary d.

Output: 1

English Translation:

Function `fn` accepts keyword arguments. It prints the value of key 'a'.

```
Dictionary `d` contains key 'a' mapped to value 1.

Function call `fn(d)` passes this dictionary as an argument.

Function execution:
Print the value of key 'a' in dictionary `d`.

Output: 1 TypeError: fn() takes no positional arguments but 1 was given

In the given Python code, the function `fn` is defined as a variable-length function with a default argu

```python
fn(a=1)  # Correct way to call the function
``` This is valid. Shows that positional arguments and keyword arguments with the same name can coexist

```python
def fn(a, a):
  print(a)

d = {'a': 1}
fn(1, **d)

# Output: 1
```

Translation:

This works. Demonstrates that positional arguments and keyword arguments with the same name can exist together.

```python
def fn(a, a):
  print(a)

d = {'a': 1}
fn(1, **d)

# Output: 1
```

```
``` In this case, an error occurred. Be aware of the subtle differences between the following situations
```

```
fn(1, **d)

In English, this Python code attempts to define a function named "fn" with one required argument "1" an

```python
def fn(a, /, **kwds):
  print(kwds['a'])

fn(1, **{1: 2})
```

The error message in the Chinese text translates to the following English error message:

```
TypeError: fn() argument after ** must be a mapping, not list
```

This error occurs because the argument passed to the function after the `**` keyword argument is a list instead of a dictionary. In the Chinese text, the dictionary is represented as `[1,2]`, but in Python, dictionaries are created using curly braces `{}` or the dictionary constructor `dict()`. Therefore, the correct call to the function would be:

```
fn(1, **{1: 2})
``` `**` should be followed by a mapping (dictionary).

Iterable types as parameters

```python
def fn(*args):
  print(args)

fn(1, 2, 3, 4, 5, 6, 7, 8, 9, kwds={"a": 1, "b": 2})
```

In the provided Python code, the function `fn` accepts a variable number of arguments using the `*args` syntax. However, the text suggests that a mapping (dictionary) should follow the `**` syntax. This is not the case for the given code. Instead, the `**` syntax is used for keyword arguments.

To follow the text' s suggestion, the function should be modified to accept a single argument that is an iterable containing key-value pairs, like a list of tuples or a dictionary. Here' s an example:

```
def fn(mapping):
  print(mapping)

fn([("a", 1), ("b", 2)])
```
```

However, it' s important to note that this is not a common way to define functions in Python, and using the ** syntax for keyword arguments is more idiomatic and widely used. body:

def fn(kwds): print(kwds)

Input: (1, 2)

Output: (1, 2)

```
The Chinese text provided does not contain any meaningful code or instructions, as it is just a functio

In the context of the error message you provided, it seems that `fn` is a function that is being called

```python
def fn(a, *args):
    # function implementation here

fn(1, 2, 3, 4)
```

In this example, `args` is an iterable that can accept multiple arguments. When calling the function, we pass the first argument `1` as a regular argument, and the remaining arguments `2, 3, 4` as an iterable using the `*` symbol. The Chinese text provided does not contain any readable information as it is just a Python code snippet and its output. The code defines a function `fn` that takes a required argument `a` and a variable number of keyword arguments `kwds`. The function prints the type of `kwds`. The output shown is the type of `kwds` when the function is called with the argument `1` and a tuple containing the single element `1`. Therefore, the output is `<class 'tuple'>`.

Since the text itself is not in Chinese, there is no Chinese text to translate. Print the type. This is why the output above is `(1,)` instead of `[1]`.

```python
def fn(*kwds):
  print(kwds)

fn(1, *[1])
```

In this code, `fn` is a function that takes variable-length arguments (`*kwds`). In the function call `fn(1, *[1])`, `[1]` is a list with one element, but the `*` before it in the function call unpacks it into separate arguments, so `kwds` receives a tuple `(1,)` instead of a list `[1]`. Therefore, the `print` statement outputs `(1,)` instead of `[1]`. The given Chinese text does not contain any readable content as it is just a code snippet written in
```

Python with some comments. The comments explain that when calling the function `fn` with an argument of a tuple `(1, 1)`, it was first unpacked into separate arguments `(1, 1)` during the function call, and then `kwds` (a keyword argument list) in the function definition converted it back into a tuple `(1, 1)`.

However, there is no Chinese text to translate in the given code snippet. If you have any Chinese text related to this code that you would like translated, please provide it. `print(','.join(['a', 'b', 'c']))`

or

`print('a,b,c')` in Python 3.6 and above (using f-strings)

The Chinese text seems to be describing a Python function call, specifically a print statement with a list being concatenated and separated by commas. The provided code snippet in Chinese translates to the English code above.

The lambda expression in the Chinese text is not present in the provided code snippet. If there was a lambda expression, it would have been something like `print(lambda x: ','.join(x) if x else '', [['a'], ['b'], ['c']])`. But since it' s not there, I didn' t include it in the translation. `lambda` is to save functions as variables. Do you remember the explanation in the "Mystery of Computer Science" article?

Python code:

```
def incrementor(n):
  return lambda x: x + n


f = incrementor(2)
print(f(3))
```

Translation:

`lambda` is to store functions as variables. Do you remember the explanation in the "Mystery of Computer Science" article?

Python code:

```
def incrementor(n):
  return lambda x: x + n


f = incrementor(2)
print(f(3))
```

English Translation:

Lambda is to save functions as variables. Do you remember the explanation in the "Mystery of Computer Science" article?

Python code:

```
def incrementor(n):
  return lambda x: x + n

f = incrementor(2)
print(f(3))
``` Five.

Example:

pairs = [(1, 4), (2, 1), (0, 3)]1. The given code snippet is written in Python. It sorts a list of pairs

2. The `pairs.sort()` function sorts the list `pairs` in-place. The `key` argument is a function that is

3. After the sorting is done, the sorted list `pairs` is printed out.

4. The output of the code is:

```python
[(0, 3), (1, 4), (2, 1)]
```

So, the English translation of the code without any Chinese characters or punctuation is:

```
pairs.sort(key=lambda pair: pair[1])
print(pairs)
```

Output:

```
[...]
[(0, 3), (1, 4), (2, 1)]
``` In English, the Chinese text does not provide any content to be translated as it is Python code writ

```python
[(0, 3), (1, 4), (2, 1)]
``` def add():
 # add something

pair = [] # empty list to store pairs

```
def sort_pairs(pairs):
 pair[0].sort() # sort first elements of pairs
 pairs.sort(key=lambda x: x[1]) # sort second elements of pairs
```

In English:

At the time of `pair[0]`, sort the first elements of the pairs. At the time of `pair[1]`, sort the second elements of the pairs.

```
def add():
 """Add something"""

 pair = [] # Initialize an empty list to store pairs

 def sort_pairs(pairs):
 # Sort the first elements of pairs
 pair[0].sort()
 # Sort the second elements of pairs using a key function
 pairs.sort(key=lambda x: x[1])
``` add: Add two numbers.

This is a comment in Python. It does not provide any output or perform any action. The next line prints

The last line is a command in Bash shell to run a script or command named 'add' with an argument 'somet

```python
def add(a: int, b: int) -> int:
 print(add.__annotations__)
 return a + b

add(1, 2)
[Function type with name 'add' and signature: (a: int, b: int) -> int]
``` { 'a': int, 'b': int, 'return': int }

## Translation:

This is a Python dictionary with three keys: 'a', 'b', and 'return'. The values of 'a' and 'b' are of t

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

a.append(5)
print(a) # [1, 2, 3, 4, 5]

a[len(a):] = [6]
print(a) # [1, 2, 3, 4, 5, 6]
```

List

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

a.append(5)
print(a) # [1, 2, 3, 4, 5]

a[len(a):] = [6]
print(a) # [1, 2, 3, 4, 5, 6] a[3] = [6]
print(a) # [1, 2, 3, 6]

a.insert(0, -1)
print(a) # [-1, 1, 2, 3, 6]

a.remove(1)
print(a) # [-1, 2, 3, 6] a.pop()
print(a) # [2, 3, -1]

a.clear()
print(a) # []

a[:] = [1, 2]
print(a.count(1)) # 1

a.reverse()
print(a) # [2, 1] (Note: The reverse method modifies the list in-place, so there is no need to assign t
```

```
b = a.copy() # make a copy of list a
a[0] = 10 # change the first element of list a to 10

print(b) # 2, 1 (since b is a copy, it hasn't been changed)
print(a) # 10, 1 (list a has been changed)

b = a # assign list b the reference of list a
a[0] = 3 # change the first element of list a to 3

print(b) # 3, 1 (since b has the same reference as a, it gets changed) print(a) # [3, 1]

### List construction

print(3 ** 2) # 9
print(3 ** 3) # 27 First, let's learn an operation, `**`. It represents `exponentiation`.

```python
sq = []
for x in range(10):
  sq.append(x ** 2)

print(sq)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
``` Try using `map` again.

```python
a = map(lambda x:x, range(10))
print(a)
# <map object at 0x...>
print(list(a))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

This is an English translation of the given Python code, which calculates the square of each number from 0 to 9 using the `map` function and prints the result as a list. The output of the code is the given list of squares.[i for i in range(5)] # [0, 1, 2, 3, 4]

sq = [x ** 2 for x in range(10)] print(sq) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

```
This Python code uses a list comprehension to create two lists. The first list `a` is created using a li

a = [i+j for i in range(3) for j in range(3)] # list comprehension to generate a list of sums of i and j

print(a) # print the list

# [0, 1, 2, 1, 2, 3, 2, 3, 4]

a = [i for i in range(5) if i % 2 == 0] # list comprehension to generate a list of even numbers in the r

print(a) # print the list

# [0, 2, 4] a = [(i, i) for i in range(3)]
# a = [(0, 0), (1, 1), (2, 2)]
```makes a list called matrix, where each inner list contains the numbers from 0 to 3 with an offset i =

creates an empty list called t. For each value j in the range 0 to 2, appends to t a new list that is a

[ [0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11] ] [list(map(lambda i: list(map(lambda j: i+j*4, range(4)))

# or

[[i+j*4 for j in range(3)] for i in range(4)]
```

Both ways produce the same result: a list of lists, where the outer list has 3 elements, and each inner list has 4 elements, calculated by adding `i` to `j` multiplied by 4 for each `i` in the range of 0 to 3, and for each `j` in the range of 0 to 2. This translates to:

for j in range(3): for i in range(4): print(i + j * 4)

```
In English, the Chinese text can be translated to:

For j in the range of 3:
For i in the range of 4:
[i plus j times 4]
```
```

This translates to Python code as:

```python
for j in range(3):
  for i in range(4):
    print(i + j * 4)
```

So, the English translation of the Chinese text without any Chinese characters or punctuation would be:

for j in range 3: for i in range 4: print i plus j times 4. This makes it convenient for matrix transposition.

```python
matrix = [[i+j*4 for i in range(4)] for j in range(3)]
print(matrix)
#  [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

mt = [[row[j] for row in matrix] for j in range(4)]
```

Transposition of matrix:

```
# [[0, 4, 8], [1, 5, 9], [2, 6, 10], [3, 7, 11]]
``` `print(mt) # [ [0, 4, 8], [1, 5, 9], [2, 6, 10], [3, 7, 11] ]

print(list(map(list, zip(*[list(t)[::-1] for t in matrix]))))
[[8, 4, 0], [9, 5, 1], [10, 6, 2], [11, 7, 3]]
```

The Chinese text seems to be describing the code for transposing a matrix in Python. The English translation of the code is provided above. The `zip(*matrix)` function is used to transpose the matrix, and `list(map(list, ...))` is used to convert the resulting tuples into lists. The `del` keyword is not used in the code. I. Initialization of list: a = [1, 2, 3, 4]

II. Deletion of element at index 1: del a[1]

III. Printing the modified list: [1, 3, 4]

IV. Deletion of elements at indices 0 to 1: del a[0:2]

V. Printing the modified list: [4] dictionary:

ages = { ‘li’ : 19, ‘wang’ : 28, ‘he’ : 7} print(ages[ ‘li’ ]) # Output: 19 print(ages[ ‘wang’ ]) # Output: 28 print(ages[ ‘he’ ]) # Output: 7

### 列表
```

```python
foods = ['apple', 'banana', 'cherry']
print(foods[0]) # Output: 'apple'
print(foods[1]) # Output: 'banana'
print(foods[2]) # Output: 'cherry'
```

元组

```python
fruits = ('apple', 'banana', 'cherry')
print(fruits[0]) # Output: 'apple'
print(fruits[1]) # Output: 'banana'
print(fruits[2]) # Output: 'cherry'

# Note: Tuples are similar to lists, but they are immutable, meaning their elements cannot be changed o
```

列表和元组的比较

```python
# Lists are mutable, meaning their elements can be changed.
my_list = [1, 2, 3]
my_list[0] = 5
print(my_list) # Output: [5, 2, 3]

# Tuples are immutable, meaning their elements cannot be changed.
my_tuple = (1, 2, 3)
# my_tuple[0] = 5 # This will result in a TypeError

# However, you can create a new tuple with the modified elements and assign it to a new variable.
new_tuple = (1, 5, 3)
print(new_tuple) # Output: (1, 5, 3)
```

字符串

```python
message = 'Hello, World!'
print(message) # Output: 'Hello, World!'
print(message[0]) # Output: 'H'
print(message[7:13]) # Output: 'World'
```

控制流

```
# If statement
if age >= 18:
    print('You are an adult.')
else:
    print('You are not an adult.')

# For loop
for fruit in fruits:
    print(fruit)

# While loop
i = 0
while i < len(foods):
    print(foods[i])
    i += 1
```

函数

```
# Function definition
def greet(name):
    print('Hello, ' + name + '!')

# Function call
greet('Alice') # Output: 'Hello, Alice!'

# Function with return value
def add(x, y):
    return x + y

# Function call with return value
sum = add(3, 5)
print(sum) # Output: 8
```

模块

```
# Importing a module
import math
```

```
# Using functions from a module
print(math.sqrt(16)) # Output: 4.0

# Importing specific functions from a module
import math as m

# Using functions from a module with alias
print(m.sqrt(16)) # Output: 4.0
```

类和对象

```
# Class definition
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print('Hello, I am ' + self.name + ' and I am ' + str(self.age) + ' years old.')

# Creating an object
person = Person('Alice', 25)

# Using methods of an object
person.greet() # Output: 'Hello, I am Alice and I am 25 years old.'

# Class inheritance
class Student(Person):
    def __init__(self, name, age, grade):
        super().__init__(name, age)
        self.grade = grade

    def print_grade(self):
        print('My grade is ' + str(self.grade))

# Creating an object of the subclass
```

```
student = Student('Bob', 18, 85)

# Using methods of the object
student.greet() # Output: 'Hello, I am Bob and I am 18 years old.'
student.print_grade() # Output: 'My grade is 85'
```

错误处理

```
# Try-except block
try:
    x = int('hello')
except ValueError:
    print('Invalid input. Please enter a valid integer.')

# Raising an exception
def divide(x, y):
    if y == 0:
        raise ZeroDivisionError('Cannot divide by zero.')
    return x / y

try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print(e)
``` I. For name, age in ages.items():
II. print(name)
III. print(age)

I. For name in ages:
II. print(name) Li, Wang, He

For the Python code, the error is due to the incorrect usage of the `for` loop with the `ages` variable

To fix the code, either use separate variables for name and age, or use `enumerate` function to get both

```python
ages = [['Li', 25], ['Wang', 30], ['He', 35]]
```

```
for name, age in ages:
    print(name)
    print(age)

# or

for i, name in enumerate(ages):
    print(name[0])
    print(name[1])
``` 1 3 2
print(i, name)

# 0 li
# 1 wang
# 2 he

print(reversed([1, 2, 3]))
# reversed object: <reverse iterator at 0x10701ffd0>

print(list(reversed([1, 2, 3])))
# [2, 1, 0]: [Three, Two, One]

### Module

### Calling a module in scripting way
```

The Chinese text provided does not contain any meaningful content related to the given code snippet in square brackets “# [3, 2, 1]”. The Chinese text seems to be unrelated to the code snippet. Therefore, no English translation is required for the given text.

However, if you meant to ask for a translation of the Chinese words “模块”(módù) and “脚本方式”(jiǎoben fāngshì), they can be translated to “module” and “scripting way” respectively. I. import sys

II. def f(n):

III. if n < 2:

IV. return n V. else:
```

V. return f(n-1) + f(n-2)

VI. if **name** == “**main**” :

VII. [

VIII.

Your code here

X.] r is assigned the value of the function f with the argument being the integer value of sys.argv[1]. The result is then printed.

In the given example, the Python script named “fib.py” is executed with the argument “3” . Therefore, the value of r will be assigned the result of the function f when called with the argument 3, which is 2. Hence, the output of the script will be “2” .

English translation: r = function(int(sys.argv[1])) print(r)

Python script execution example: % python fib.py 3 Result: 2 The given Chinese text does not contain any readable information as it only consists of a Python command and its argument. The command is to run the built-in Python module ‘fib’ with an argument of ‘5’ . In English, this can be written as:

python -m fib 5

In the provided Python code snippet, it seems to be importing a module named ‘fib’ . However, without knowing the content of the ‘fib’ module, it is impossible to provide a translation for it. Therefore, I will only provide the English translation for the Python command.[‘**builtins**’ , ‘**cached**’ , ‘**doc**’ , ‘**file**’ , ‘**loader**’ , ‘**name**’ , ‘**package**’ , ‘**spec**’ , ‘f’ , ‘sys’]

The given Chinese text is not provided, as it is empty. The output is the result of the provided Python code snippet when run in an interpreter. The `dir()` function returns a list of all the attributes and methods of an object in Python. In this case, it is being called on the `fib` function, which is assumed to be a previously defined Fibonacci sequence generator function. The output shows the various attributes and methods that are available for this function object in the Python interpreter.[ArithmeticError, AssertionError, AttributeError, BaseException, BlockingIOError, BrokenPipeError, BufferError, BytesWarning, ChildProcessError, ConnectionAbortedError, ConnectionError, ConnectionRefusedError, ConnectionResetError, DeprecationWarning, EOFError, Ellipsis, EnvironmentError, Exception, False, FileExistsError, FileNotFoundError, FloatingPointError, FutureWarning, GeneratorExit, IOError, ImportError, ImportWarning, IndentationError, IndexError, InterruptedError, IsADirectoryError, KeyError, KeyboardInterrupt, LookupError, MemoryError, ModuleNotFoundError, NameError, None, NotADirectoryError, NotImplemented, NotImplementedError, OSError, OverflowError, PendingDeprecationWarning, PermissionError, ProcessLookupEr-

ror, RecursionError, ReferenceError, ResourceWarning, RuntimeError, RuntimeWarning, StopAsyncIteration, StopIteration, SyntaxError, SyntaxWarning, SystemError, SystemExit, TabError, TimeoutError, True, TypeError, UnboundLocalError, UnicodeDecodeError, UnicodeEncodeError, UnicodeError, UnicodeTranslateError, UnicodeWarning, UserWarning, ValueError, Warning, ZeroDivisionError, **build_class**, **debug**, **doc**, **import**, **loader**, **name**, **package**, **spec**, abs, all, any, ascii, bin, bool, breakpoint, bytearray, bytes, callable, chr, classmethod, compile, complex, copyright, credits, delattr, dict, dir, divmod, enumerate, eval, exec, exit, filter, float, format, frozenset, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, quit, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip]

Classes and functions defined in the built-in module builtins

This code imports the built-in module `__builtins__` and prints out the list of its attributes (i.e., classes and functions). The list contains various error types, special values, and built-in functions that are part of the Python standard library.

Here' s a brief description of some of the items in the list:

- Error types: `ArithmeticError`, `AssertionError`, `AttributeError`, `BaseException`, `ConnectionAbortedError`, `ConnectionError`, `ConnectionRefusedError`, `ConnectionResetError`, `DeprecationWarning`, `EOFError`, `FileExistsError`, `FileNotFoundError`, `FloatingPointError`, `IOError`, `ImportError`, `ImportWarning`, `IndentationError`, `IndexError`, `InterruptedError`, `IsADirectoryError`, `KeyError`, `KeyboardInterrupt`, `LookupError`, `MemoryError`, `ModuleNotFoundError`, `NameError`, `NotADirectoryError`, `NotImplementedError`, `OSError`, `OverflowError`, `PendingDeprecationWarning`, `PermissionError`, `ProcessLookupError`, `RecursionError`, `ReferenceError`, `ResourceWarning`, `RuntimeError`, `RuntimeWarning`, `StopAsyncIteration`, `StopIteration`, `SyntaxError`, `SyntaxWarning`, `SystemError`, `SystemExit`, `TabError`, `TimeoutError`, `ZeroDivisionError`.
- Special values: `False`, `None`, `True`.
- Built-in functions: `abs`, `all`, `any`, `ascii`, `bin`, `bool`, `bytearray`, `bytes`, `callable`, `chr`, `classmethod`, `compile`, `complex`, `copyright`, `credits`, `delattr`, `dict`, `dir`, `divmod`, `enumerate`, `eval`, `exec`, `exit`, `filter`, `float`, `format`, `frozenset`, `getattr`, `globals`, `hasattr`, `hash`, `help`, `hex`, `id`, `input`, `int`, `isinstance`, `issubclass`, `iter`, `len`, `license`, `list`, `locals`, `map`, `max`, `memoryview`, `min`, `next`, `object`, `oct`, `open`, `ord`, `pow`, `print`, `property`, `quit`, `range`, `repr`, `reversed`, `round`, `set`, `setattr`, `slice`, `sorted`, `staticmethod`, `str`, `sum`, `super`, `tuple`, `type`, `vars`, `zip`.
- Special names: `__build_class__`, `__debug__`, `__doc__`, `__import__`, `__loader__`, `__name__`, `__package__`, `__spec__`.

These names are used for special purposes in Python and are not meant to be used as regular variables or functions. `pk.py` is the package, named `packages`.

```
fibp
|-- cal
|   `-- cal.py
|-- pt
   `-- pt.py
```

`cal.py` and `pt.py` are files under the `cal` and `pt` directories respectively, inside the package. `cal.py`:

```
def f(n):
if n < 2:
 return n
else:
 return f(n-1) + f(n-2)
```

Fibonacci function

This is a Python script defining a recursive function `f` that calculates the Fibonacci sequence. The function takes an integer `n` as an argument and returns the `n`th number in the Fibonacci sequence. If the input `n` is less than 2, the function simply returns `n`. Otherwise, it recursively calls itself with arguments `n-1` and `n-2` and adds the results to get the Fibonacci number at position `n`. def f(x): # function ‘f’ is not defined in the given text, so it cannot be translated

def fl(n): return [f(i) for i in range(5)]

def p(l): print(l, end=’’) “‘

The given Chinese text describes the definition of three Python functions: ‘fl’, ‘p’, and ‘f’. However, the function ‘f’ is not defined in the text, so it cannot be translated into English. The ‘fl’ function takes one argument ‘n’ and returns a list generated by applying the ‘f’ function to each element in the range from 0 to 4. The ‘p’ function takes one argument ‘l’ and prints each element in the list with a space as the separator. The text does not provide any information about what the ‘f’ function does, so it cannot be translated without additional context. def pln(l): print(l)

pk.py:

import fibp.cal.cal import fibp.pt.pt

No Chinese text provided for translation. The given code is Python. `fibp.cal.cal(pt.pt(10))`

In this translation, I assumed that `fibp.cal.cal` and `fibp.pt.pt` are functions or modules named `cal` and `pt` respectively, which are located inside the package named `fibp`. The number `10` is an argument passed to the `pt` function. I'm unable to directly translate the given Chinese text as it is not provided in the text. The given text appears to be a programming code snippet written in Python. It seems to be calling a function `cal` with an argument of `10`, and applying the `pt.p` function to the result of that call with the `cal.fl` function as its operator. Without knowing the definition of the `cal` and `pt.p` functions, it's impossible to provide an accurate English translation.