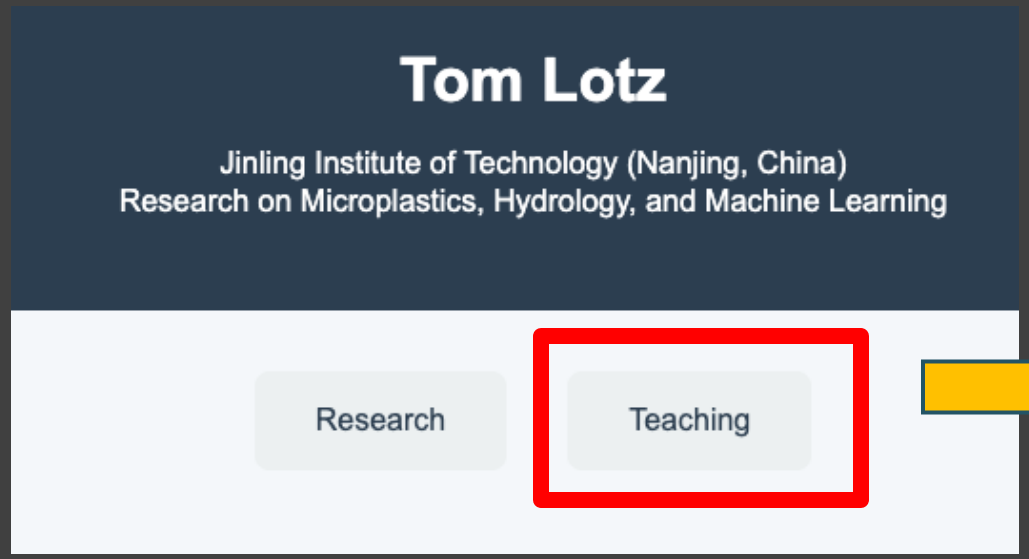
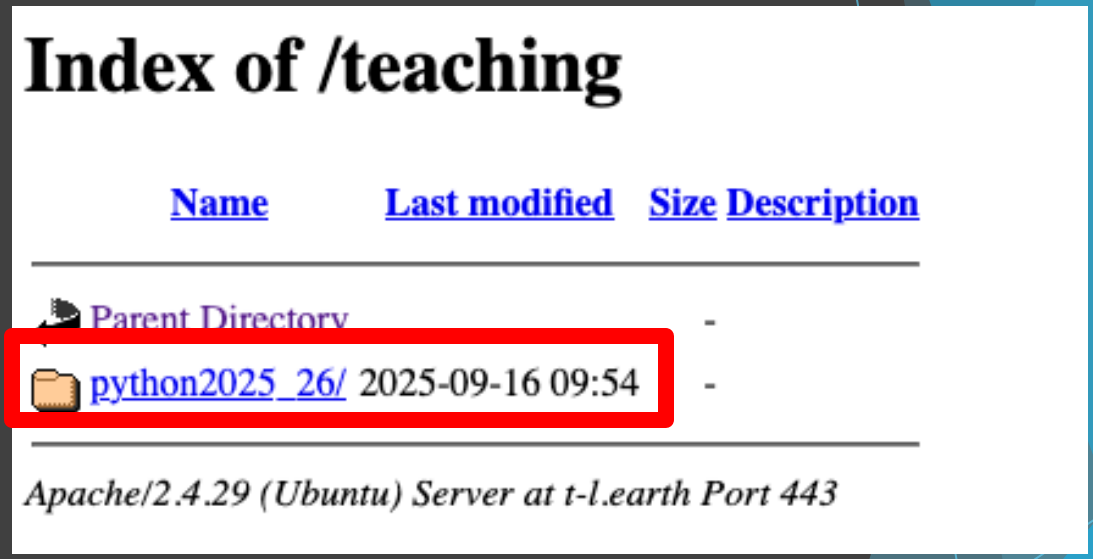
Download the cheat sheets and slides from here

t-l.earth





Python语言程序设计

Python Programming

2025/26

Session 03

Tom Lotz (tom.lotz@outlook.com)

Content

Review

What Is a List?

- A list is a collection of items in a particular order.
- Lists can contain:
  - Strings, numbers, booleans, other lists
  - Even mixed types (though not recommended)
- Think of a list as a shelf where each item has a position.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

Accessing Elements by Index

- Use brackets to get an item: list[index]
- Index starts at 0
- Negative indices count from the end

```
names = ['Tom', 'Anna', 'Zhang Wei']

print(names[0]) # Tom
print(names[-1]) # Zhang Wei
```

Changing Elements in a List

- You can change any value by accessing it via its index
- Syntax: list[index] = new_value

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles[0] = 'ducati'
print(motorcycles)
```

Adding Items with append()

- append() adds an element to the end of the list
- Often used to build lists dynamically

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
print(motorcycles)
```

Inserting Elements

- insert(index, value) adds an item at any position
- Shifts following elements right

```
motorcycles = ['honda', 'yamaha']
motorcycles.insert( __index: 1,  __object: 'suzuki')
print(motorcycles)
```

Removing Items with del

- del removes an item by index
- You cannot access the value after deleting

```
motorcycles = ['honda', 'yamaha', 'suzuki']
del motorcycles[1]
print(motorcycles)
```

Removing Items with pop()

- pop() removes the last item by default
- You can store and use the removed item

```
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.pop()
print(motorcycles) # ['honda', 'yamaha']

popped = motorcycles.pop()
print(popped) # suzuki
```

pop() from Specific Position

- You can specify the index inside pop(index)

```
motorcycles = ['honda', 'yamaha', 'suzuki']
first = motorcycles.pop(0)
print(first)
```

Removing by Value with remove()

- remove(value) deletes the first occurrence
- Useful when you know the value, not the index

```
motorcycles = ['honda', 'yamaha', 'ducati']
motorcycles.remove('ducati')
print(motorcycles) # ['honda', 'yamaha']
```

Looping Through a List

- Use a for loop to repeat actions for every item
- Syntax: for variable in list:

```
magicians = ['alice', 'david', 'carolina']

for magician in magicians:
    print(magician)
```

Sorting a List

- sort() changes the list permanently
- sorted() returns a new sorted list
- Use reverse=True for reverse order

```
cars = ['bmw', 'audi', 'toyota']
cars.sort()
cars.sort(reverse=True)
```

Slicing a List

- Get part of a list: list[start:end]
- End index is not included

```
players = ['a', 'b', 'c', 'd', 'e']
print(players[1:4]) # ['b', 'c', 'd']
print(players[-3:]) # ['c', 'd', 'e']
```

List Comprehensions


```
squares = [value**2 for value in range(1, 11)]
```

- Loop through numbers from 1 to 10
- Square each number
- Store each result in the list squares

```
squares = []
for value in range(1, 11):
    squares.append(value**2)
```

List Comprehensions

- Very flexible

```
names = ['tom', 'anna', 'bob']
names = [name.title() for name in names]
```

Copying a list safely

- Use slicing ([:]) to copy a list
- Avoid direct assignment, which creates a reference

```
original = ['pizza', 'pasta']
copy = original[:]
copy.append('salad')
print(original) # ['pizza', 'pasta']
print(copy) # ['pizza', 'pasta', 'salad']
```

Scary time!

Conditional Logic

Decisions in Python

- Programs often need to make decisions
- Python uses if statements to check conditions
- Conditions evaluate to True or False

Why decisions?

- Websites check if a username already exists
- Games award points depending on actions
- Programs behave differently based on input

Conditional Tests

- A conditional test is an expression that returns True or False.

```
1 == 2
2 == 2

car = 'bmw'
print(car == 'bmw') # True
print(car == 'audi') # False
```

Equality (==)

- Check if two values are the same:

```
car = 'bmw'
car == 'bmw' # True
```

Inequality (!=)

- Check if values are different:

```
car = 'bmw'
car == 'bmw' # True
car != 'bmw' # False
car != 'audi' # True
```

Case Sensitivity

- Python is case sensitive:

```
car = 'Audi'
print(car == 'audi') # False
print(car.lower() == 'audi') # True
```

Numerical Comparisons

- Use comparison operators:

```
age = 19
print(age < 21) # True
print(age >= 21) # False
```

Logical Operators: and

- Both conditions must be True:

```
age_0 = 22
age_1 = 18
print(age_0 >= 21 and age_1 >= 21) # False
```

Logical Operators: or

- Only one condition needs to be True:

```
age_0 = 22
age_1 = 18
print(age_0 >= 21 or age_1 >= 21) # True
```

Membership: in

- Check if a value is in a list:

```
requested_toppings = ['mushrooms', 'onions']
print('mushrooms' in requested_toppings) # True
```

Membership: not in

- Check if a value is not in a list:

```
banned_users = ['andrew', 'carolina']
user = 'marie'
print(user not in banned_users) # True
```

Boolean Variables

- Booleans are just True / False:

```
game_active = True
can_edit = False

True == False
True == True
True != False
True != True

bigger = 1 > 0
print(bigger) # True
```

Key Takeaways

- Conditional tests return True or False.
- Operators: ==, !=, <, >, <=, >=.
- Use .lower() for case-insensitive checks.
- Combine tests with and / or.
- Use in / not in to check lists.
- Boolean variables (True, False) help track state.

if Statement

- Check if a condition is met and act accordingly

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
```

if Statement

- Check if a condition is met and act accordingly

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
```

```
if condition is True:
    execute this code

if condition is False:
    execute this code
```

if-elif-else Chains

- Check if a condition is met and act accordingly

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
else:
    price = 40
```

if-elif-else Chains

- We can leave out else, if all conditions are covered

```
age = 70
if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
elif age >= 65:
    price = 20
```

Independent if Statements

- We can also use a series of if statements

```
toppings = ['mushrooms', 'extra cheese']

if 'mushrooms' in toppings:
    print("Adding mushrooms.")

if 'extra cheese' in toppings:
    print("Adding extra cheese.")
```

if Statements with Lists

- Checking special items:

```
available_toppings = ['mushrooms', 'extra cheese']
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for topping in requested_toppings:
    if topping not in available_toppings:
        print("Sorry, ",topping," is not available today.")
else:
    print("Adding ",topping)
```

Key Takeaways

- if → check a single condition.
- if-else → choose between 2 outcomes.
- if-elif-else → select between multiple exclusive outcomes.
- Independent ifs → all conditions are checked.
- With lists → handle missing items, empty lists, or unavailable options.
- Order of conditions matters!

User Input

User Input

- Programs often need information from users.
- Input makes programs interactive.
- Python uses the input() function.

Why do we need input?

- Games ask for player names.
- Websites ask for usernames and passwords.
- Programs need numbers to calculate results.
- Without input, programs always behave the same.

The input() Function

- input() shows a prompt to the user.
- Waits for the user to type and press Enter.
- Stores the response in a variable.

```
message = input("Tell me something: ")
print(message)
```

Clear Prompts

- Always make the prompt clear:

```
# Good prompt:
name = input("Please enter your name: ")
print(f"Hello, {name}!")

# Bad prompt:
name = input(": ")
```

Input is Always a String

- The input is always stored as a string

```
age = input("How old are you? ")
print(age) # '21'
print(type(age)) # type: string
```

Converting Input

- Use (for example) int() to convert to a number

```
age = input("How old are you? ")
age = int(age)

if age >= 18:
    print("You can vote!")
else:
    print("Too young to vote.")
```

The Modulo Operator %

- Gives the remainder after division
- Useful to check if a number is even or odd.

```
print(4 % 3) # 1
print(6 % 3) # 0
```

Even or Odd Checker

```
number = input("Enter a number: ")
number = int(number)

if number % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

Key Takeaways

- Use input() to get user data.
- Always give clear prompts.
- Input is a string → convert with int() for numbers.
- % operator helps check for patterns (e.g., even/odd).
- Combining input with if makes interactive programs.

While Loops & Interaction

While Loops & Interaction

- Repeat actions as long as a condition is True.
- Used in games, data entry, interactive programs.
- Games run until the player quits.
- Programs can ask for input many times.
- Useful when you don’t know in advance how many repetitions are needed.

The while Loop

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

- What does it do?

Built-in escape route

- While loops need a defined ending criteria

```
while True:
    age = input("Enter a number: ")
    age = int(age)
```

Built-in escape route

- Using a Flag
- Better if there are multiple exit criteria

```
active = True
times = 0

while active: # same as: while active == True:
    message = input("Enter text (or 'quit'): ")
    if message == 'quit':
        active = False
    if times > 10:
        active = False
    else:
        print(message)
        times += 1 # same as: time =  time + 1
```

Using break

- while True runs forever until break.
- Clean way to exit immediately.

```
while True:
    city = input("Enter a city ('quit' to stop): ")
    if city == 'quit':
        break
    print(f"I'd love to go to {city}!")
```

Using continue

- Continue returns immediately back to the while condition, skipping all code below

```
number = 0
while number <6:
    number += 1
    if number == 3:
        continue
    print(number)
```

```
1
2
4
5
6
```

Example

- Move items between lists

```
unconfirmed = ['alice', 'brian', 'candace']
confirmed = []

while unconfirmed:
    user = unconfirmed.pop()
    print(f"Verifying {user}")
    confirmed.append(user)
```

Example

- Remove items from a list

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat']
while 'cat' in pets:
    pets.remove('cat')
    print(pets)
```

Key Takeaways

- while runs as long as condition is True.
- Use flags or break for flexible exit conditions.
- Use continue to skip part of the loop.
- Be careful to avoid infinite loops.
- Can process lists and dictionaries until empty.

Exercises

Instructions

- Try to finish as many exercises as possible.
- Indicate the name and number of each exercise above your code.
- Save your exercise results and send the file to tom.lotz@outlook.com at the end of the session.
- Include your name in the email.

Alien colors

- 5-3. Alien Colors #1
  - Create a variable alien_color with value 'green', 'yellow', or 'red'.
  - If it is 'green' → print “Player earned 5 points”.
  - Write one version where the test passes and one where it fails.

- 5-4. Alien Colors #2
  - If the alien is green → 5 points.
  - If not → 10 points.
  - Write one version for each case.

- 5-5. Alien Colors #3
  - Use if-elif-else.
  - Green → 5 points, Yellow → 10 points, Red → 15 points.
  - Write three versions to test all branches.

If Chains & Lists

- 5-6. Stages of Life
  - Use variable age.
  - <2 → baby, 2-4 → toddler, 4-13 → kid, 13-20 → teenager, 20-65 → adult, ≥65 → elder.

- 5-7. Favorite Fruit
  - Make a list favorite_fruits with 3 fruits.
  - Write 5 independent if tests.
  - If a fruit is in the list → print “You really like bananas!” (etc.).

If Chains & Lists

- 5-8. Hello Admin
  - Make a list of usernames (include 'admin').
  - Loop through names.
  - If 'admin' → print special greeting.
  - Else → print normal greeting.

- 5-9. No Users
  - Check if list of users is empty.
  - If empty → print “We need to find some users!”.

If Chains & Lists

- 5-10. Checking Usernames
  - current_users list, new_users list.
  - If new username already exists (case-insensitive) → say not available.
  - Else → say available.

- 5-11. Ordinal Numbers
  - Make list 1–9.
  - Use if-elif-else to print the numbers with correct ending: 1st, 2nd, 3rd, 4th....

- 5-12. Styling if Statements
  - Review your code.
  - Follow PEP 8 spacing: if age < 4: not if age<4:.

User Input

- 7-1. Rental Car
  - Ask the user which rental car they want.
  - Print “Let me see if I can find you a Subaru” (or chosen car).

- 7-2. Restaurant Seating
  - Ask how many people are in the dinner group.
  - If >8 → say they must wait.
  - Else → table ready.

- 7-3. Multiples of Ten
  - Ask the user for a number.
  - If number % 10 == 0 → say it’s a multiple of 10.
  - Else → not a multiple of 10.

While Loops

- 7-4. Pizza Toppings
  - Ask for toppings until user types 'quit'.
  - Print message for each topping added.

- 7-5. Movie Tickets
  - Ask user's age.
  - <3 → free, 3–12 → $10, >12 → $15.
  - Loop until user quits.

While Loops

- 7-6. Three Exits
  - Modify Exercise 7-4 or 7-5 three ways:
  - Stop loop with condition in while.
  - Use an active flag.
  - Use break when user types 'quit'.

- 7-7. Infinity
  - Write a loop that never ends.
  - Stop it manually with CTRL+C.