CS305 Lab2
Brief introduction to Python & Wireshark

Dept. Computer Science and Engineering
Southern University of Science and Technology

SUSTech
Southern University
of Science and Technology

Thanks to HHQ. Zhang

Part. A

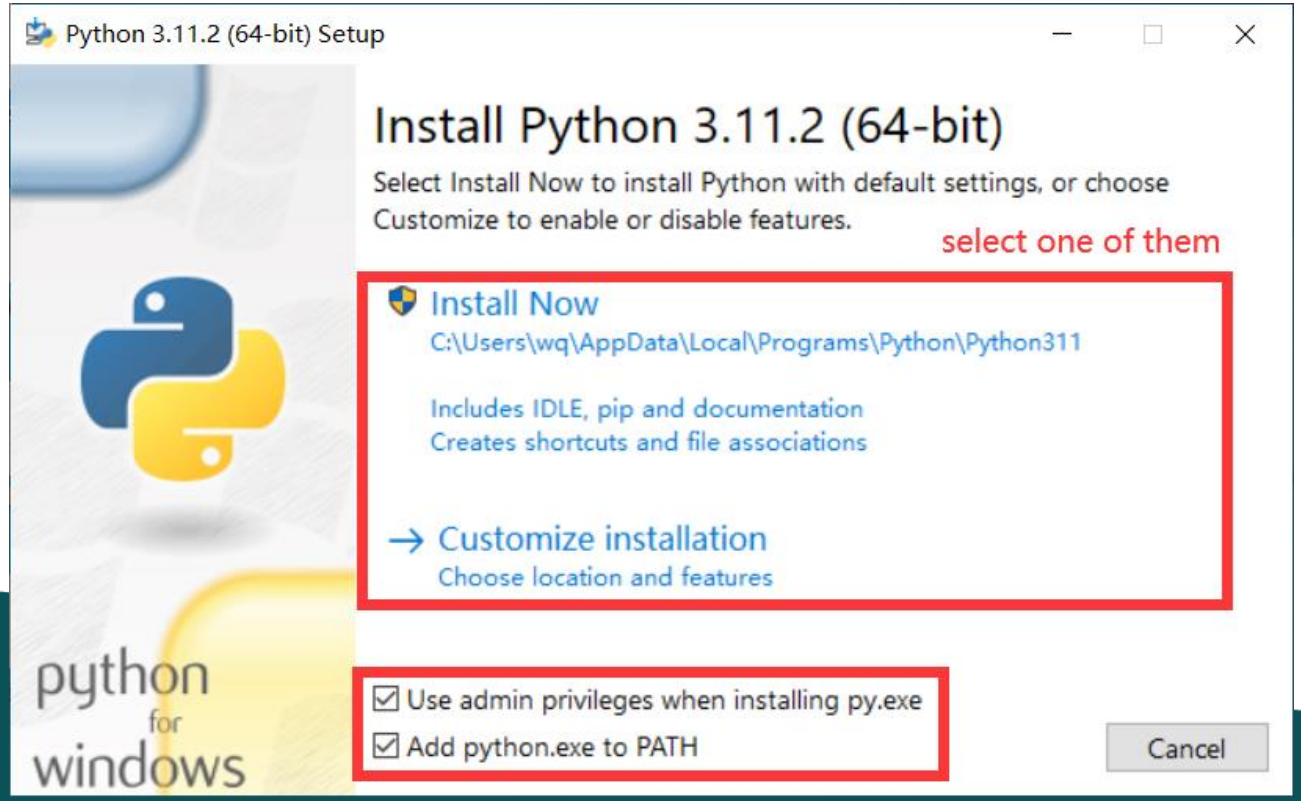
Introduction to Python

Python

- Python is an interpreted high-level object-oriented programming language.
- First release in 1991.
- Official Tutorial: https://docs.python.org/3/tutorial/

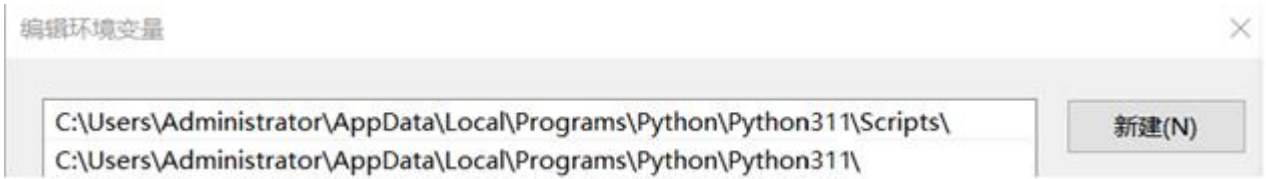
Install python(1)

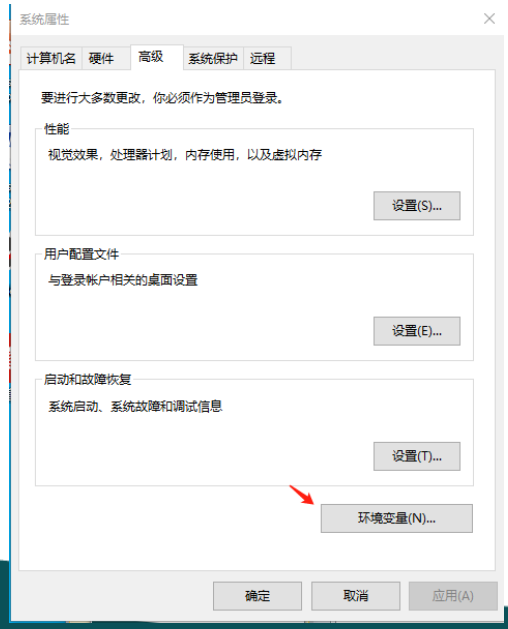
- The installation package can be got from **https://www.python.org/downloads/**
- You can choose install it by **default settings** or **customize installation**.
- It is highly recommend that choose 'Add python.exe to PATH', or you need to set PATH by hand as next page shows.



Install python(2)

- If the ‘Add python.exe to PATH’ is not set while installing, configure ‘Path’ manually according to the following steps after the installation.
  - Right click ‘my computer’ on the desktop
  - select ‘attribute’-> ‘advanced attribute’->environment variable
  - configure ‘Path’ with the path where python.exe belongs and its subdirectory ‘Scripts’





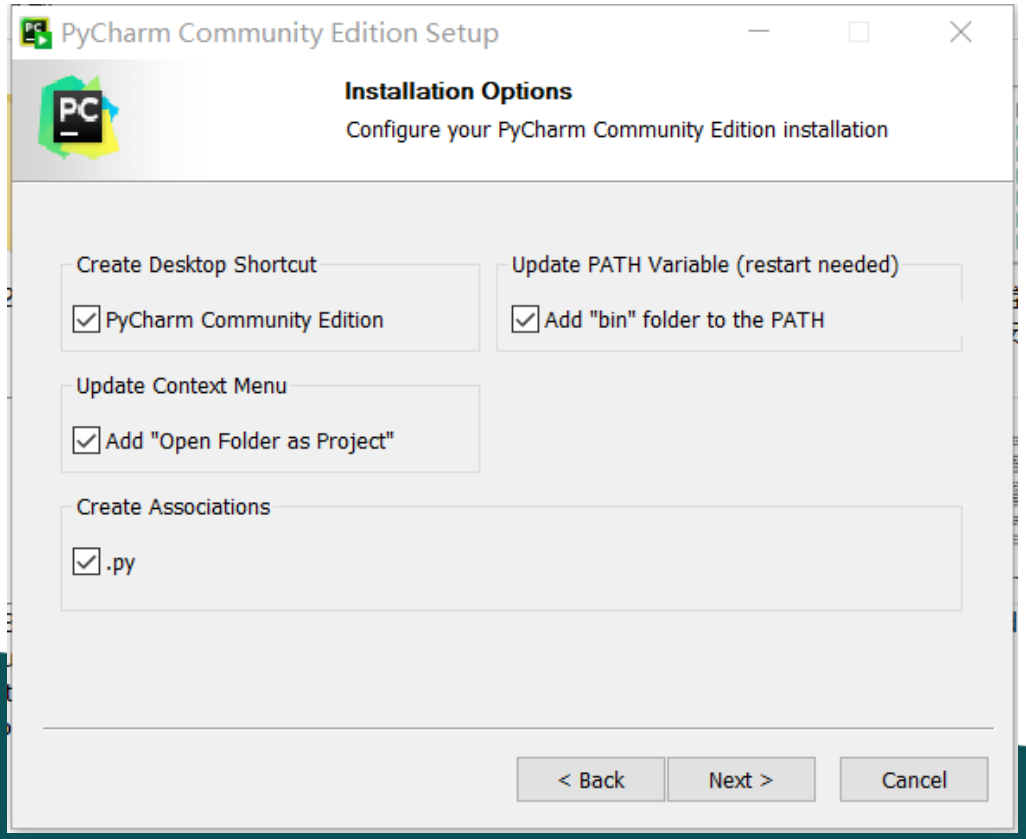
Read-Eval-Print Loop

- Python has an REPL playground.
- Type and get feedback.

```
C:\Users\Administrator>python
Python 3.8.6rc1 (tags/v3.8.6rc1:08bd63d, Sep  7 2020, 23:10:23) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>> _
```

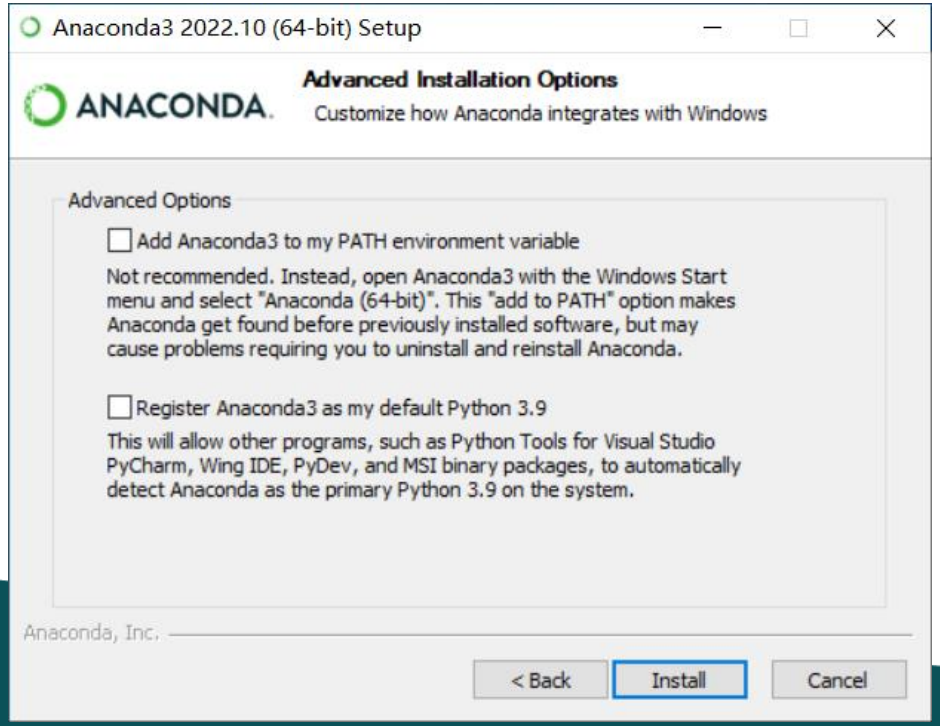
Install IDE: PyCharm as an example

- The installation package can be got from **https://www.jetbrains.com.cn/en-us/pycharm/**
- Official Tutorial: **https://www.jetbrains.com.cn/help/pycharm/quick-start-guide.html**



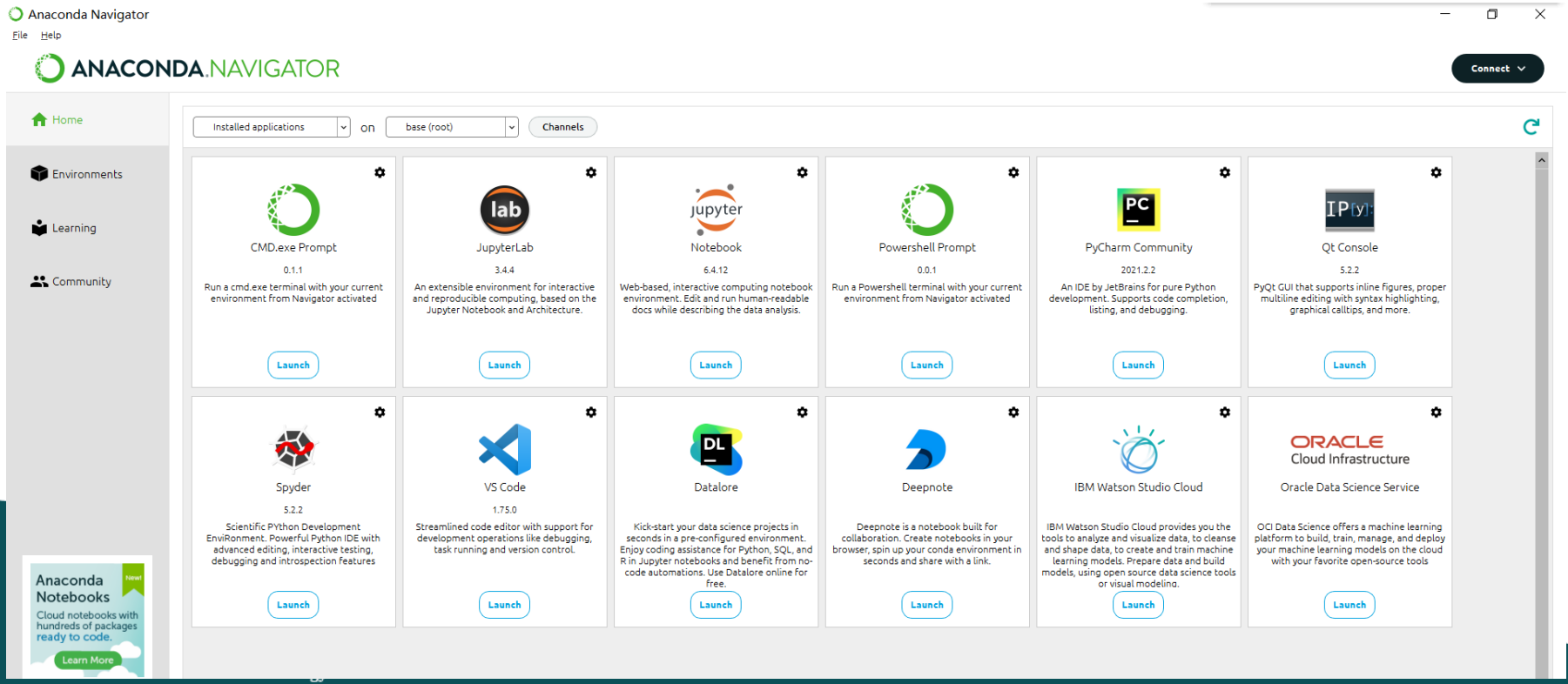
Install Anaconda(optional)

- Open source package management system and environment management system.
- Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine.
- The installation package can be got from **https://www.anaconda.com/**



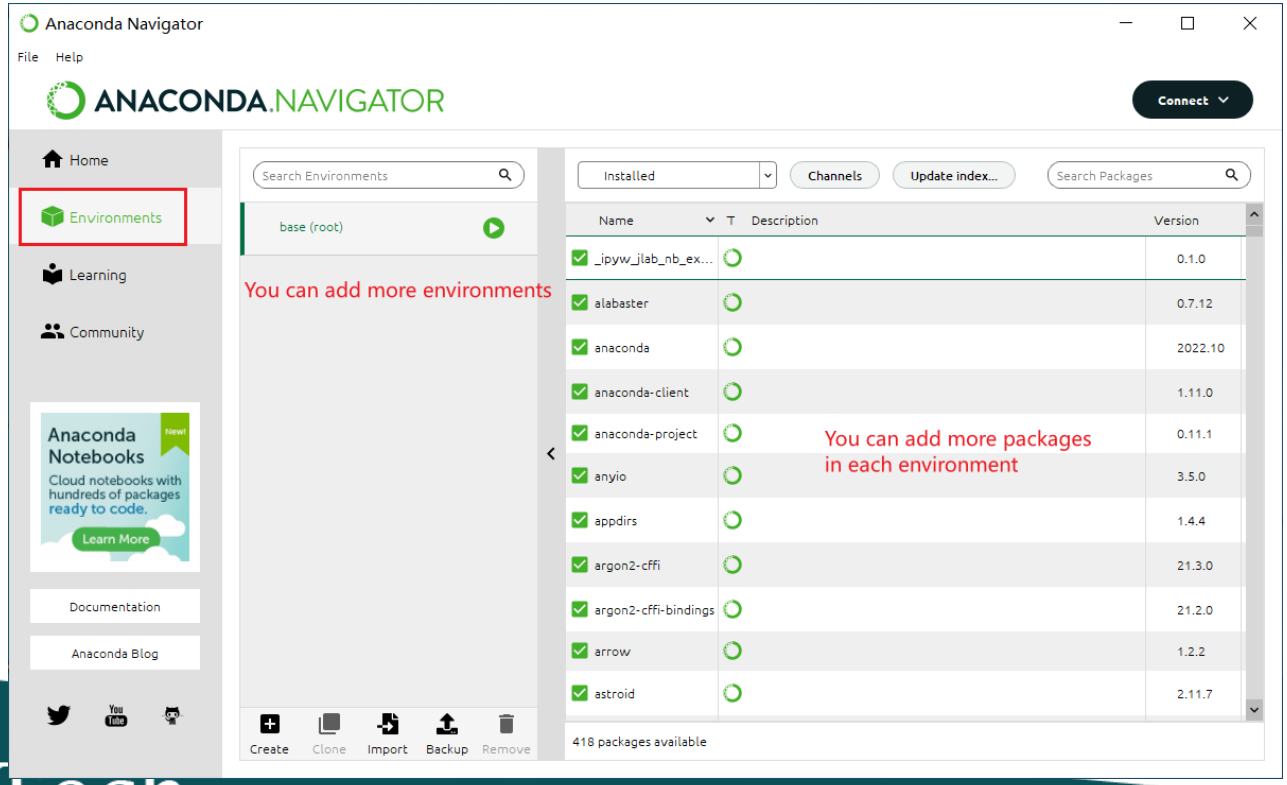
Anaconda usage

- Anaconda comes with lots of tools, including editor, interpreter, and IDE.
- Anaconda includes more than 180 scientific packages and their dependencies, including conda and Python.



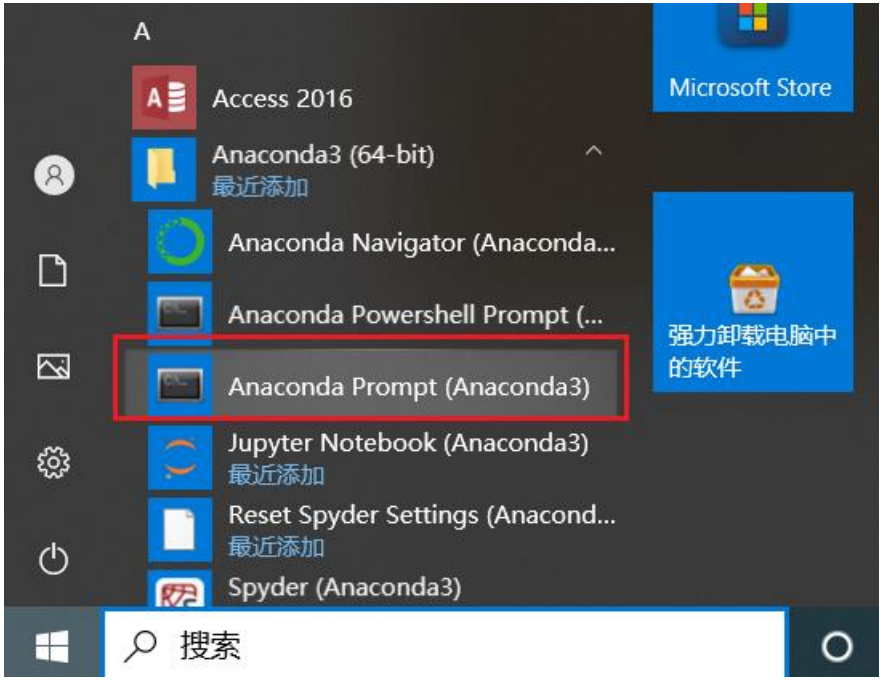
Anaconda environment

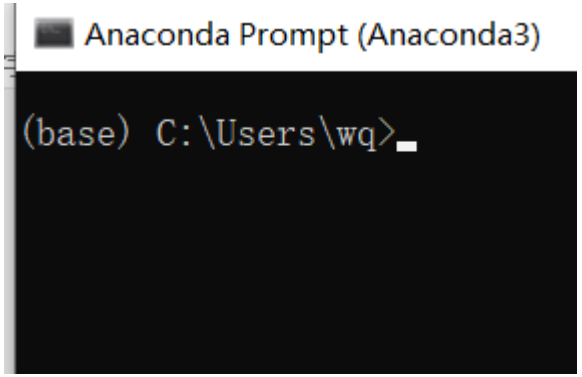
- Anaconda can create, save, load and switch environments for different projects conveniently.



Anaconda environment

- Anaconda command line.





Frequently-used conda commands

- check help information: conda –h
- check conda version: conda --version
- install package: conda install *** (for example, django)
- list all the packages: conda list
- update the package: conda update ***
- remove the package: conda remove ***
- create a new environment: conda create –n *** python = version
- activate the environment: activate ***
- quit the environment: conda deactivate ***
- delete an environment: conda remove –n *** --all
- list all environments: conda env list

Basic Types and Operations

- The following standard types are built in the interpreter:
  - **Numeric** Types — int, float, complex
  - **Boolean** Type — True, False
  - **Text Sequence** Type — str
  - **Sequence** Types — list, tuple, range
  - **Set** Type & **Dict** Type
  - **Binary Sequence** Types — bytes, byte array
- There are predefined operations on each type

- Ref: https://docs.python.org/3/library/stdtypes.html

Sequence Types

- **List**

  animals = **[**'dog', 'cat', 'bird'**]**

  animals[0] # => 'dog'

  animals[0] = 'puppy'

```
>>> animals = ['dog','cat','bird']
>>> animals[0]
'dog'
>>> animals[0]='puppy'
```

- **Tuple**

  animals = **(**'dog', 'cat', 'bird'**)**

  animals[0] # => 'dog'

  animals[0] = 'puppy'

  Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

  TypeError: 'tuple' object does not support item assignment

```
>>> animals = ('dog','cat','bird')
>>> animals[0]
'dog'
>>> animals[0]='puppy'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

Unpacking from Sequence Types

- **List**

  foo, bar = **[**'dog', 'cat'**]**

  foo # => 'dog'

  bar # => 'cat'

```
>>> foo, bar = ['dog', 'cat']
>>> foo
'dog'
>>> bar
'cat'
```

- **Tuple**

  foo, bar = **(**'dog', 'cat'**)**

  foo # => 'dog'

  bar # => 'cat'

```
>>> foo, bar = ('dog', 'cat')
>>> foo
'dog'
>>> bar
'cat'
```

Set & Dict

- **Set**

```
animals = set()
animals.add('dog')
animals  # => {'dog'}
```

```
>>> animals = set()
>>> animals.add('dog')
>>> animals
{'dog'}
```

- **Dict**

```
alias = dict()
alias['dog'] = 'puppy'
alias[['pig']] = ['hog']
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

```
>>> alias = dict()
>>> alias['dog']='putty'
>>> alias[['pig']] = ['hog']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Immutable & Mutable

- Mutable: it is possible to change its content
- **Immutable Type: Numeric, Boolean, str, tuple, bytes, etc.**
- **Mutable Type: list, dict, set, etc.**
- Example:

```
>>> cubes = [1, 8, 27, 65, 125]   # cubes here is a list
>>> cubes[3] = 64                 # replace the item whose index is 3
>>> cubes
[1, 8, 27, 64, 125]
```

- Only **Immutable types** can be **key of dict** or **member of set**.

Boolean Values

- Following values are treated as **False**:
  - **None**, **False**
  - **0**, **0.0**, **0j**, **Decimal(0)**, **Fraction(0, 1)**
  - **''**, **()**, **[]**, **{}**, **set()**, **range(0)**

- Otherwise they are **True**

```
>>> bool(None)
False
>>> bool(Fraction(0,2))
False
>>> bool('')
False
>>> bool(' ')
True
>>> bool(Fraction(1,2))
True
```

Flow Control — if

- Example:

```
foo = []
if foo:
    print(foo)
else:
    if foo == []:
        print('100% sure foo is empty')
    else:
        print('what hell?')
```

Flow Control — if

- Example:

```
foo = [1, 2, 3, 4]
if foo:
    print(foo)
else:
    if foo == []:
        print('100% sure foo is empty')
    else:
        print('what hell?')
```

Flow Control — for

- Example:

```
foo = ['dog', 'cat', 'bird']
for bar in foo:
    print(bar)
for index, value in enumerate(foo):
    print('%d: %4s' % (index, value))
    print('{0}: {1}'.format(index, value))
for i in range(10):
    print(i,end=" ")
```

```
dog
cat
bird
```

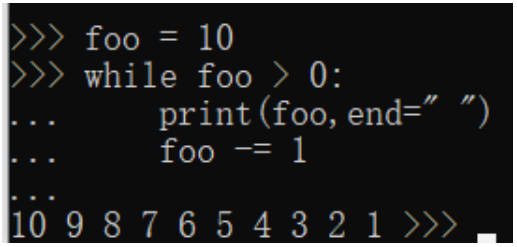
```
0:  dog
0: dog
1:  cat
1: cat
2: bird
2: bird
```

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

Flow Control — while

- Example:

```
foo = 10
while foo > 0:
    print(foo, end=" ")
    foo -= 1
```



Defining Functions

- Example:

```
def fib(n):  # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

Defining Functions

- Example:

```
def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)    # see below
        a, b = b, a+b
    return result
```

Closure

- A closure is an inner function that has access to the outer (enclosing) function's variables.
- Example:

```
def add(x):
    def addX(y):
        return y + x
    return addX
foo = add(1)
print(foo(2)) # => 3
```

Practise:
After the defination of function "add", run the following test, what's the testing result?

```
foo = add(1)
print(foo(2))
print(foo(3))
goo = add(100)
print(goo(2))
print(foo(4))
```

Defining Classes

```
class Animal:
    def __init__(self, name):
        self.name = name

class Duck(Animal):
    def __init__(self, name):
        super(Duck, self).__init__(name)
    def quack(self):
        print(self.name, ' Quack')
```

Duck Type

- "If it walks like a duck and it quacks like a duck, then it must be a duck"

```
class Dog(Animal):
    def __init__(self, name):
        super(Dog, self).__init__(name)
    def quack(self):
        print(self.name, ' Quack')
```

Duck Type

- "If it walks like a duck and it quacks like a duck, then it must be a duck"

```
def testDuck(duck):
    duck.quack()


duck = Duck('Tommy')
dog = Dog('Fox')
testDuck(duck)
testDuck(dog)
```

```
Tommy  Quack
Fox  Quack
```

Practise:

1. What's the testing result if run testDuck('duck')?
2. Modify the name of parameter of “testDuck” from “duck” to “x” , will the running result of “testDuck(duck)” and “testDuck(dog)” be changed?
3. If Duck and Dog don't share the same parents, will the running result of “testDuck(duck)” and “testDuck(dog)” be changed?

Module

- Save our fib functions(fib and fib2) into fibs.py

```
import fibs
fibs.fib(5) # => 0 1 1 2 3
result = fibs.fib2(5) # => [0, 1, 1, 2, 3]
```

Practise:

Add two functions fib3 and fib4 to fibs.py, the parameter of these two function is the number of first items of fibonacci sequence. fib3 print the specified number of first items of fibonacci sequence, fib4 returen a list which includes the specified number of first items of fibonacci sequence.

for example:

```
fibs.fib3(10) # => 0 1 1 2 3 5 8 13 21 34
result = fibs.fib3(10) # => [0, 1, 1, 2, 3,5,8,13,21,34]
```

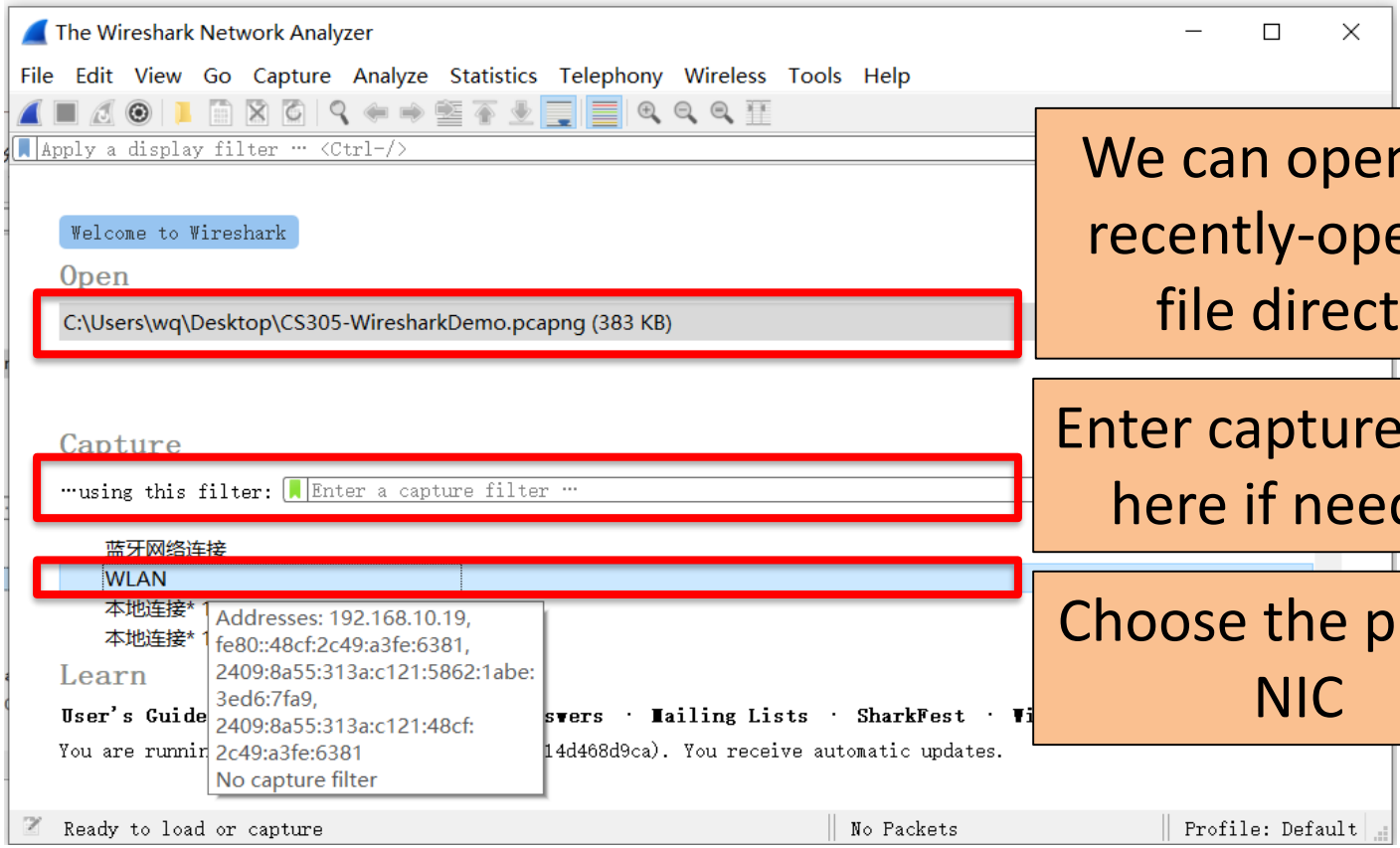
Part. B

Packet Capture and Analysis

Wireshark

- Wireshark is a free and open-source packet analyzer.
  It is used for network trouble shooting, analysis, software and communications protocol development, and education.
- Official Website: http://www.wireshark.org/
- Alternative utilities:
  - Tcpdump
  - Tshark
- Tip: new version of Wireshark uses Npcap instead of Winpcap.
- Wireshark User's Guide:
  - file:///C:/Program%20Files/Wireshark/Wireshark%20User's%20Guide/index.html (may varies in different PCs, depending on the Wireshark installation location on your PC)
  - https://gitlab.com/wireshark/wireshark/-/wikis/home

Main Interface



We can open the recently-opened file directly

Enter capture filter here if needed

Choose the proper NIC

Capture Filter (1)

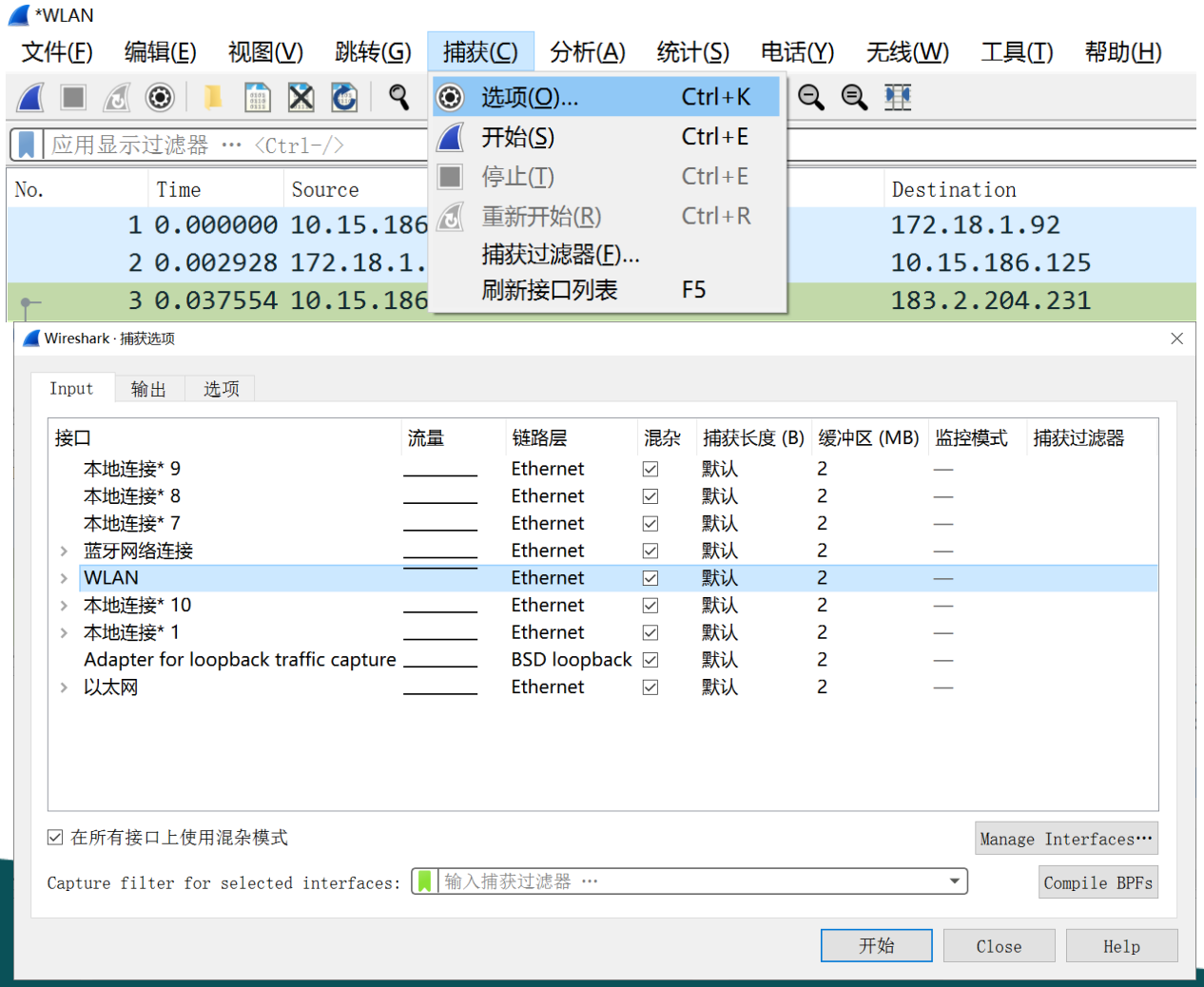
- Capture filter allows you to select the packets you want from all the packets captured by Wireshark.
- A proper capture filter can reduce the workload of Wireshark and the size of raw packets.
- Capture filter **is not** a display filter.
- Wireshark capture filters are written in libpcap filter language.
- Basic syntax: ***[not] primitive [and|or [not] primitive ...]***
  - Green: valid capture filter
  - Red: invalid capture filter

Capture Filter (2)

- Example:
  - host 172.18.5.4: This example captures traffic to and from the host 172.18.5.4
  - port 53: This example captures DNS traffic
  - tcp port 23 and host 10.0.0.5: This example captures telnet traffic to and from the host 10.0.0.5
  - dst net 192.168.0.0/24: This example captures traffic to a range of destination IP address from 192.168.0.0 to 192.168.0.255
- More syntax explanation and examples:
  - https://gitlab.com/wireshark/wireshark/-/wikis/CaptureFilters
  - http://www.tcpdump.org/manpages/pcap-filter.7.html

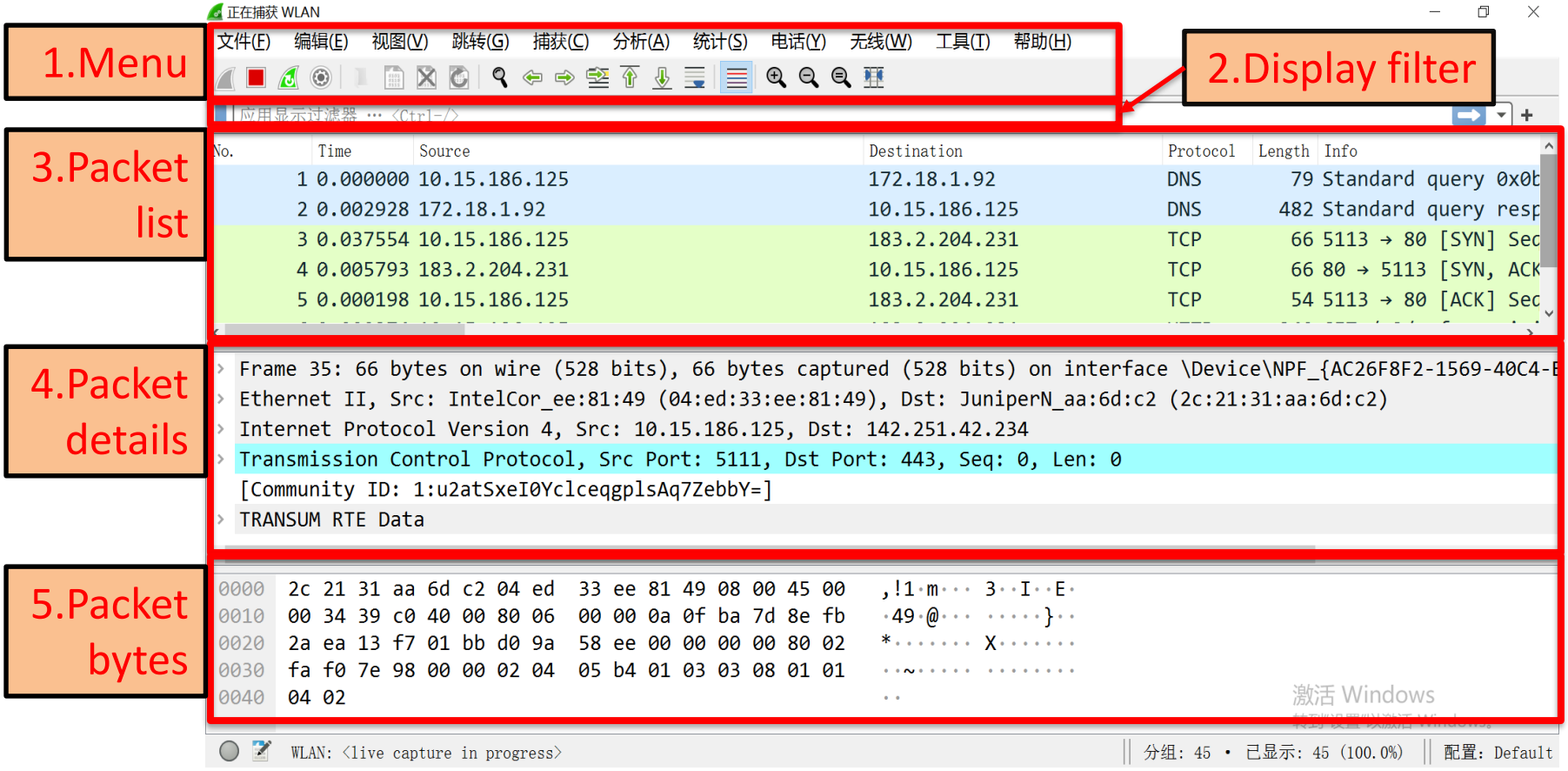
Capture Filter (3)

- Set capture filters after starting
- Capture -> Options



SUSTech
Southern University of Science and Technology

Display Interface



SUSTech
Southern University of Science and Technology

Display Filter

- After the capture starts, the display filter can be set to accurately hide the packet you don’t care.
- Display filter can be change at anytime on the fly.
- Filters are evaluated against each individual packet.
- Boolean expressions dealing with packet properties.
- Supports regular expressions.
- Can either be manually constructed, composed via the expressions menu or composed based on a selected packet's properties.

Build Display Filter Expressions

- Enter regular expressions in filter text box
  - Green: valid filter
  - Red: invalid filter
  - Yellow: may produce unexpected results
- Packet based filter
  - Filters can be constructed on the basis of individual packets by right clicking on a packet and selecting either:
  - Prepare as filter: creates a filter
  - Apply as filter: creates a filter and applies it to the trace
  - Follow TCP Stream: creates a filter from a TCP packet's stream number and applies it to the trace.

Display Filter Expressions (1)

- Uses Perl regex syntax
- Comparing Values
- Compound Filters

Table 2. Display Filter Logical Operations

English	C-like	Description
and	&&	Logical AND
or	||	Logical OR
xor	^^	Logical XOR
not	!	Logical NOT
[…]		Subsequence
in		Set Membership

Table 1. Display Filter Comparison Operators

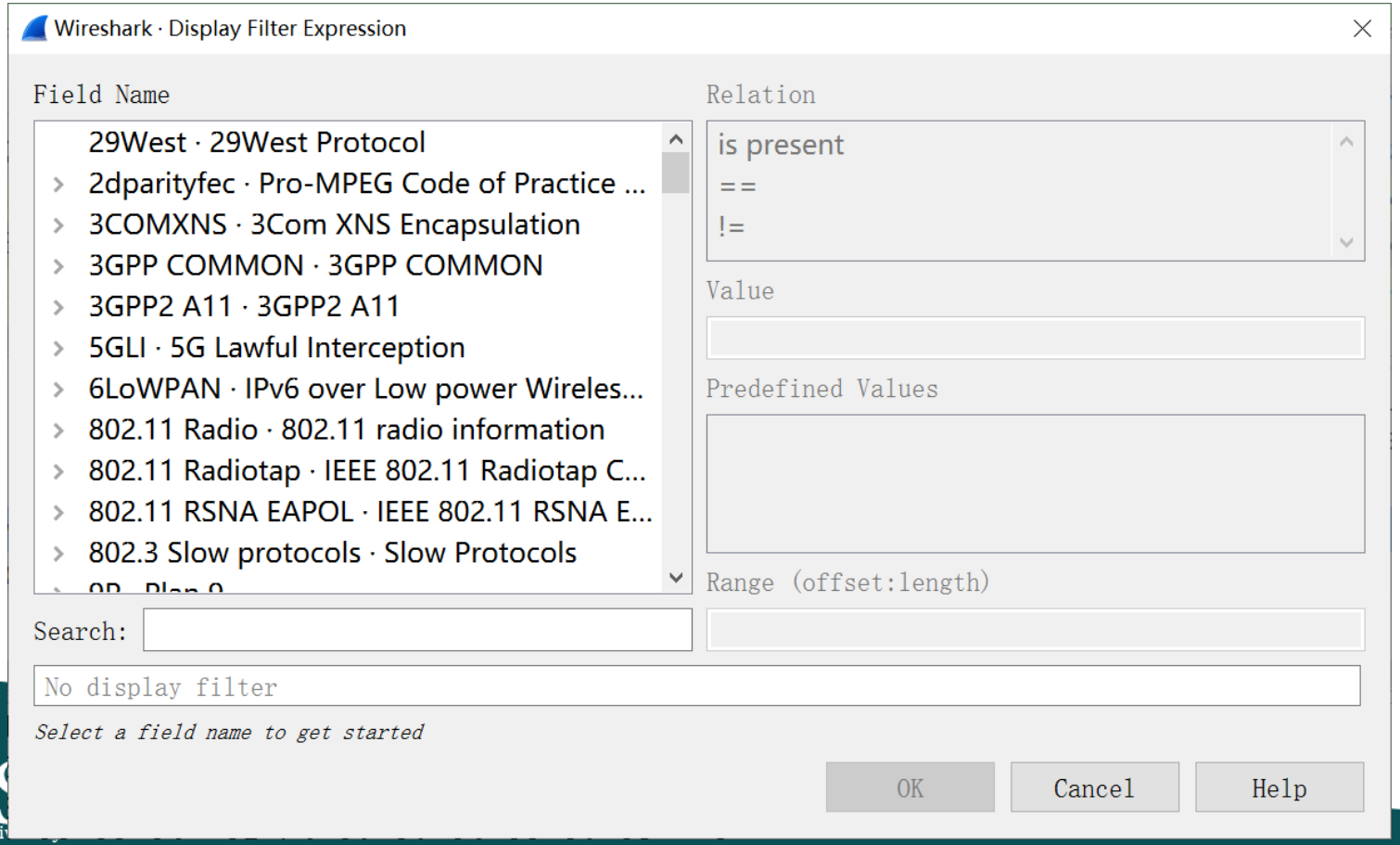
English	C-like	Description
eq	==	Equal
ne	!=	Not equal
gt	>	Greater than
lt	<	Less than
ge	>=	Greater than or equal to
le	<=	Less than or equal to
contains		Protocol, field or slice contains a value
matches	~	Protocol or text field matches a Perl-compatible regular expression
Bitwise_and	&	Bitwise AND is non-zero

Display Filter Expressions (2)

- Examples:
  - tcp.port eq 25 or icmp: Shows only SMTP (port 25) and ICMP traffic.
  - ip.len le 1500: Shows the IP packets whose length field is less than or equal to 1500 bytes.
  - ip.src != xxx.xxx.xxx.xxx && ip.dst != xxx.xxx.xxx.xxx && sip : Filter by a protocol (e.g. SIP) and filter out unwanted IPs.
  - http.request.uri matches “(gif)$”: Display all HTTP requests in which the uri ends with “gif”.
- More examples:
  - file:///C:/Program%20Files/Wireshark/Wireshark%20User's%20Guide/ChWorkBuildDisplayFilterSection.html (may varies in different PCs, depending on the Wireshark installation location on your PC)
  - https://gitlab.com/wireshark/wireshark/-/wikis/DisplayFilters

“Display Filter Expression” Dialog Box (1)

- Analyze -> Display filter expression…



“Display Filter Expression” Dialog Box (2)

- Field name: selects the packet property.
  - Every protocol with filterable fields is listed at the top level.
  - You can search for a particular protocol entry by entering the first few letters of the protocol name.
  - By expanding a protocol name you can get a list of the field names available for filtering for that protocol.
- Relation: selects the Boolean test.
- Value: Arbitrary Textual or Numeric value against which the selected packet property is tested.
- Predefined values: common values against which the selected packet property is tested.
- Search
- Range

Capture Filter VS. Display Filter

- Usage
  - Capture filters are much more limited and are used to reduce the size of a raw packet capture.
  - Display filters are used to hide some packets from the packet list.
- Syntax
  - tcp port 80
  - tcp.port == 80
- Setting
  - Capture filters are set before starting a packet capture and cannot be modified during the capture.
  - Display filters on the other hand do not have this limitation and you can change them on the fly.

Packet List Pane (1)

- Displays all of the packets in the trace in the order they were recorded.
- Columns
  - Time: the timestamp at which the packet crossed the interface.
  - Source: the originating host of the packet.
  - Destination: the host to which the packet was sent.
  - Protocol: the highest level protocol that Wireshark can detect.
  - Length: the length in bytes of the packet on the wire.
  - Info: an informational message pertaining to the protocol in the protocol column.

Packet List Pane (2)

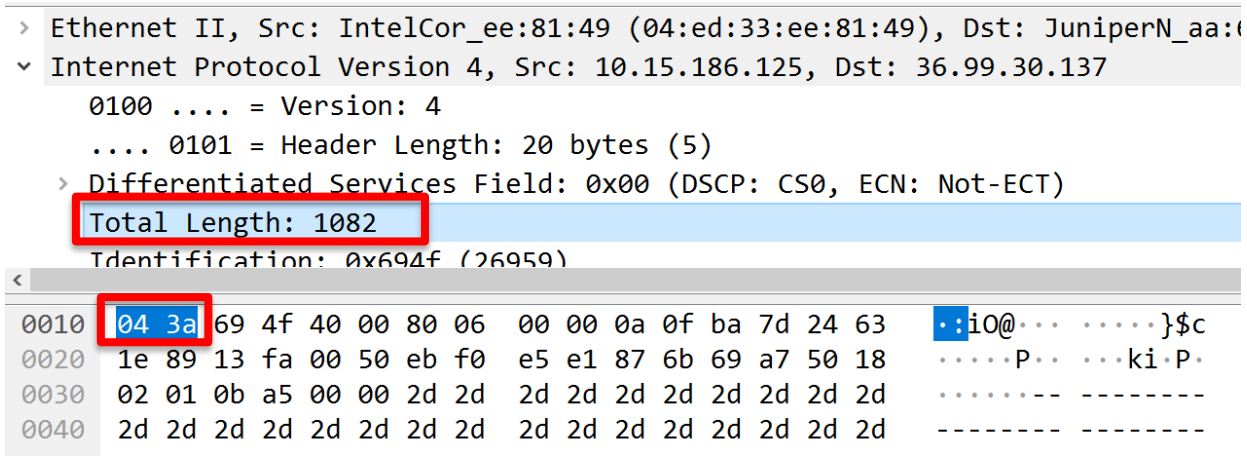
- Default Coloring
  - Gray: TCP packets
  - Black with red letters: TCP Packets with errors
  - Green: HTTP Packets
  - Light Blue: UDP Packets
  - Pale Blue: ARP Packets
  - Lavender: ICMP Packets
  - Black with green letters: ICMP Packets with errors
- Colorings can be changed under View -> Coloring Rules

Packet Details Pane

- Display detailed information about the currently selected packet.
- All packet layers are displayed in the tree menu.
- Any portion of any layer can be exported via a right click and selecting Export Selected Packet Bytes.

Packet Bytes Pane

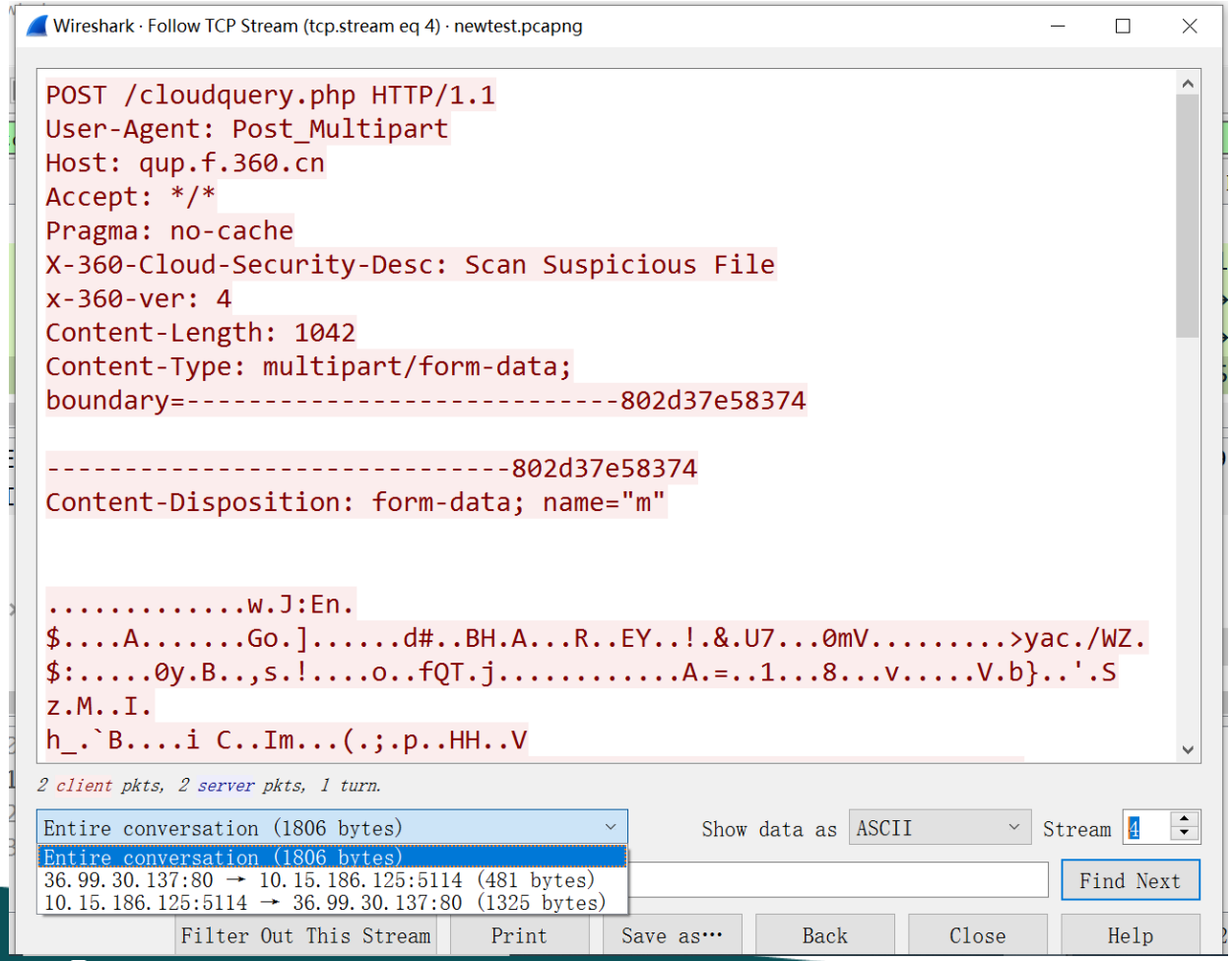
- Displays the raw packet bytes.
- The selected packet layer is highlighted.
- Network byte order verification
  - The high byte data is at the low address.
  - The low byte data is at the high address.
  - The large end mode.



Trace Analysis (1)

- Display packets belong to the same stream.
- Dialog box
  - Analyze -> Follow -> ** Stream
  - Right click the specified packet -> Follow -> **Stream
- Useful for debugging or analyzing any TCP based application layer protocol.
- Protocols supported
  - TCP, UDP, DCCP, TLS, HTTP, HTTP/2, QUIC, SIP

Trace Analysis (2)



SUSTech
Southern University
of Science and Technology

Statistics

- General statistics
  - Capture File Properties about the capture file
  - Protocol Hierarchy of the captured packets.
  - Conversations e.g. traffic between specific IP addresses.
  - Endpoints e.g. traffic to and from an IP addresses.
  - I/O Graphs visualizing the number of packets (or similar) in time.
- Protocol specific statistics
  - Service Response Time between request and response of some protocols.
  - Various other protocol specific statistics.

“Capture File Properties” Dialog

- General information about the current capture file.
- Information:
  - Details: Notable information about the capture file.
    - File: General information about the capture file.
    - Time: The timestamps of the first and the last packet in the file along with their difference.
    - Capture Information about the capture environment.
    - Interfaces Information about the capture interface or interfaces.
    - Statistics: A statistical summary of the capture file.
  - Capture file comments

“Protocol Hierarchy” Window

- This is a tree of all the protocols in the capture.
- Protocol hierarchy columns
  - Protocol: This protocol’s name.
  - Percent Packets: The percentage of protocol packets relative to all packets in the capture.
  - Packets: The total number of packets of this protocol.
  - Percent Bytes
  - Bytes: The total number of bytes of this protocol.
  - Bits/s: The bandwidth of this protocol relative to the capture time.
  - End Packets: The absolute number of packets of this protocol where it was the highest protocol in the stack (last dissected).
  - End BytesEnd Bits/s: The bandwidth of this protocol relative to the capture time where was the highest protocol in the stack (last dissected).
- Useful for determining the types, amounts, and relative proportions of protocols within a trace.

“Endpoints” Window

- A network endpoint is the logical endpoint of separate protocol traffic of a specific protocol layer.
- Endpoint and Conversation types:
  - Bluetooth, Ethernet, Fibre Channel, IEEE 802.11, FDDI, IPv4, IPv6, IPX, JXTA, NCP, RSVP, SCTP, TCP, Token Ring, UDP, USB
- For each supported protocol, a tab is shown in this window.
- Each row in the list shows the statistical values for exactly one endpoint.
- Name resolution will be done if selected in the window and if it is active for the specific protocol layer.
- Limit to display filter will only show conversations matching the current display filter.
- “Endpoint Types” button lets you choose which traffic type tabs are shown.

“Conversation” Window

- A network conversation is the traffic between two specific endpoints.
- Each row in the list shows the statistical values for exactly one conversation.
- Compared to endpoints window, this one adds four columns:
  - Rel Start/ Abs Start: the start time of the conversation
  - the duration of the conversation in seconds
  - the average bits (not bytes) per second in each direction

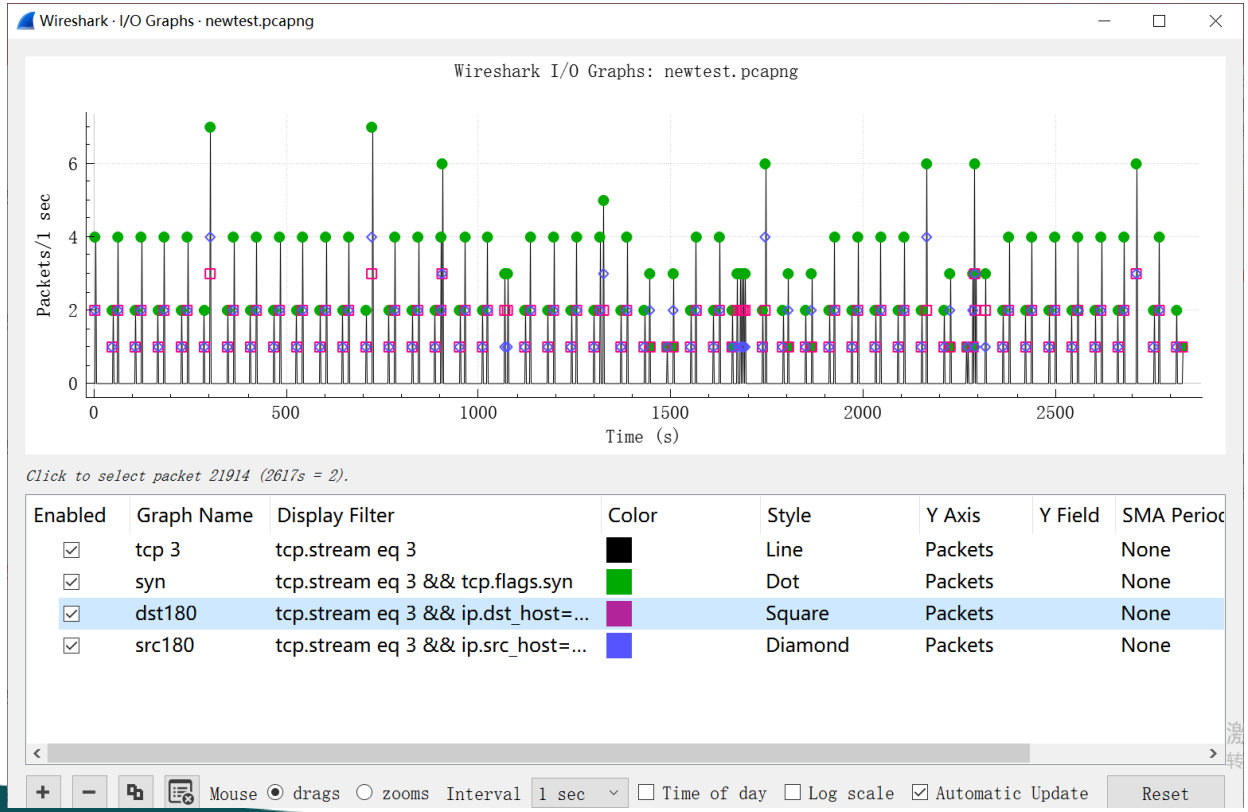
“Packet Lengths” Window

- Shows the distribution of packet lengths and related information.
- Information is broken down by packet length ranges.
  - Packet Lengths
  - Count
  - Average
  - Min Val, Max Val: The minimum and maximum lengths in this range.
  - Rate (ms): The average packets per millisecond for the packets in this range.
  - Percent: The percentage of packets in this range, by count.
  - Burst Rate: Packet bursts are detected by counting the number of packets in a given time interval and comparing that count to the intervals across a window of time. Statistics for the interval with the maximum number of packets are shown. By default, bursts are detected across 5 millisecond intervals and intervals are compared across 100 millisecond windows.
  - Burst Start: The start time, in seconds from the beginning of the capture, for the interval with the maximum number of packets.

“I/O Graphs” Window

- You can plot packet and protocol data in a variety of ways.

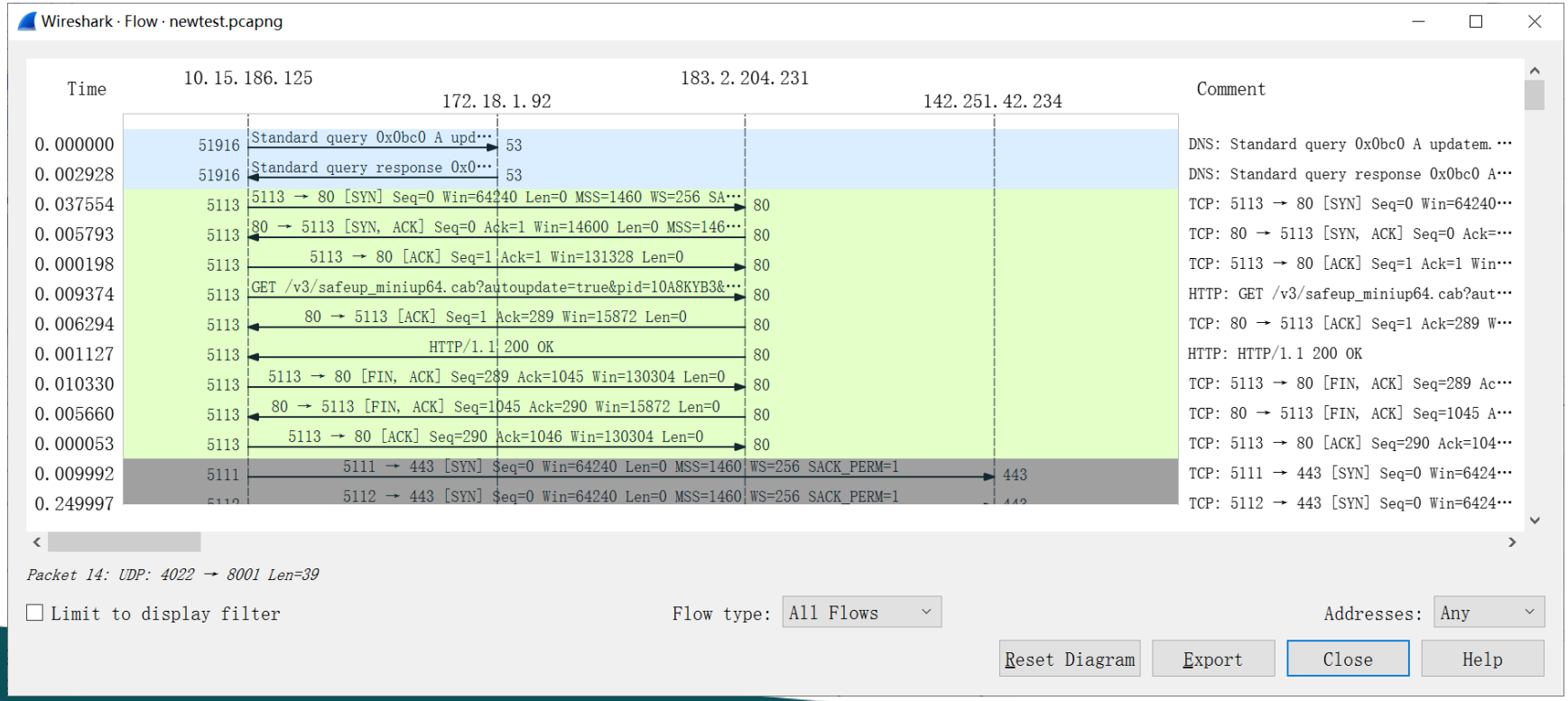
- Double clicking to change the setting of columns.
- Use + and – buttons to add or remove a row.
- Do not forget to check “Enabled” list to show in the graph.



SUSTech
Southern University of Science and Technology

“Flow Graphs” Window

- Shows connections between hosts.
- Useful for understanding seq. and ack. calculations.
- Each vertical line represents the specific host.



Practice 2.1

- Number sequence generation
  - Requirements:
    - Suppose the initial number is 2023.
    - Multiply the last two digits and append the last digit of the product to the end of the initial number. For example, (1) the last two digits of 2023 are 2 and 3, so the product is 6. Append 6 to the end of 2023; (2) Then multiply 3 and 6 again and the result is 18. Append 8 to the end of 20236; (3) Keep doing the above procedure and the final number will be like 2023688428……
    - Now you are required to generate a number with a specific length.
  - Input: an integer number that indicates the number of digits to generate
  - Output: the last generated digit
  - Example: input 10, and output 4 (e.g., 2023688428688428……, 2023 don't account for the length, and the 10^{th} digital is 4 in red.)

Practice 2.2

Use Wireshark to capture packets and answer the questions with your screenshots:

1. Launch a http session between your host and “www.example.com”

   1-1. What are the capture filter conditions? List as more as you can.

   1-2. What are the display filter conditions? List as more as you can.

   1-3. Find a HTTP packet whose destination is your localhost in this http session, and find what are the decimal and hexadecimal representations of the source IP address, source port, destination IP address and destination port?

2. Launch a http session between your host and “www.baidu.com”

   2-1. Answer the question 1-3 based on the new http session between your host and “www.baidu.com”

   2-2. List the items which value is same in the answers of both question 1-3 and 2-1.

Practice 2.3 (Optional)

Use ICMPv4 to trace route between your computer (source) and www.163.com (destination). Use a proper capture filter and display filter separately to show this session. And then answer the following questions with words and screenshots on both the execution result of command(DOS) and capture result of Wireshark:

1. How many 'time-to-live exceed' and 'echo reply' response messages are received? What's the source IP address of the 1st received 'time-to-live exceed' message, What's the source IP address of the 1st received 'echo reply' message?

2. Calculate the RTT (round-trip time) between your host and www.163.com based on the packets captured. Are they the same with RTT from command execution result?

3. Add the value of hops (between source and destination) and TTL value of ICMPv4 messages received by source (which sends ICMPv4 echo request). Is it the initial value of TTL from ICMPv4 message send by source or the ICMPv4 message send by destination? How to prove this conclusion?