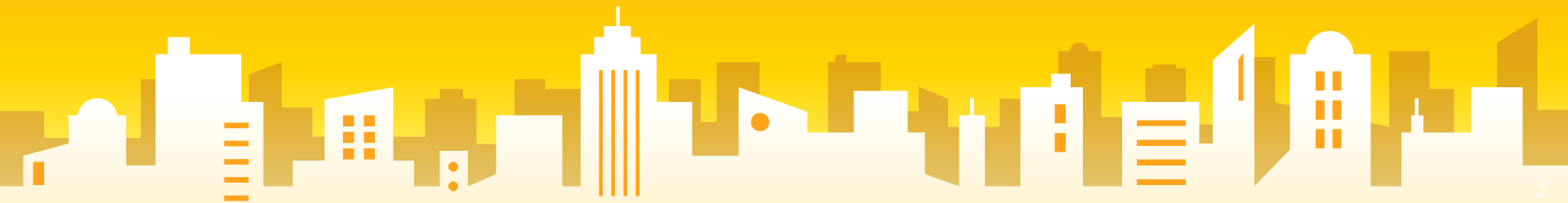
Exceptions in Python

First Lesson. (Most Important)



Syntax Error

```
>>> if a < 3
  File "<interactive input>", line 1
    if a < 3
           ^
SyntaxError: invalid syntax
```

Exceptions in Python

- Errors can also occur at runtime and these are called exceptions. They occur, for example, when a file we try to open does not exist (FileNotFoundError), dividing a number by zero (ZeroDivisionError), module we try to import is not found (ImportError) etc.

- Whenever these type of runtime error occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Python Built-in Exceptions

locals()['__builtins__']

Exception	Cause of Error
AssertionError	Raised when `assert` statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the `input()` functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's `close()` method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits interrupt key (Ctrl+c or delete).

What are exceptions in Python?

- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.
- For example, if function A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.
- If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

Catching Exceptions in Python

- In Python, exceptions can be handled using a try statement.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.
- It is up to us, what operations we perform once we have caught the exception. Here is a simple example.

Example

```
import sys

randomList = ['a', 0, 2]

for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print("The reciprocal of",entry,"is",r)
```

Example Description

- In this program, we loop until the user enters an integer that has a valid reciprocal. The portion that can cause exception is placed inside try block.

- If no exception occurs, except block is skipped and normal flow continues. But if any exception occurs, it is caught by the except block.

- Here, we print the name of the exception using ex_info() function inside sys module and ask the user to try again. We can see that the values 'a' and '1.3' causes ValueError and '0' causes ZeroDivisionError.

Catching Specific Exceptions in Python

- In the above example, we did not mention any exception in the except clause.
- This is not a good programming practice as it will catch all exceptions and handle every case in the same way. We can specify which exceptions an except clause will catch.
- A try clause can have any number of except clause to handle them differently but only one will be executed in case an exception occurs.
- We can use a tuple of values to specify multiple exceptions in an except clause. Here is an example pseudo code.

Example

```
try:
    # do something
    pass

except ValueError:
    # handle ValueError exception
    pass

except (TypeError, ZeroDivisionError):
    # handle multiple exceptions
    # TypeError and ZeroDivisionError
    pass

except:
    # handle all other exceptions
    pass
```

Raising Exceptions

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.
- We can also optionally pass in value to the exception to clarify why that exception was raised.

Example

```
>>> raise KeyboardInterrupt
Traceback (most recent call last):
...
KeyboardInterrupt

>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument

>>> try:
...       a = int(input("Enter a positive integer: "))
...       if a <= 0:
...             raise ValueError("That is not a positive number!")
... except ValueError as ve:
...       print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```

try...finally

- The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.
- For example, we may be connected to a remote data center through the network or working with a file or working with a Graphical User Interface (GUI).
- In all these circumstances, we must clean up the resource once used, whether it was successful or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee execution.
- Here is an example of file operations to illustrate this.

Example

```
try:
    f = open("test.txt",encoding = 'utf-8')
    # perform file operations
finally:
    f.close()
```

Python Custom Exceptions

- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- However, sometimes you may need to create custom exceptions that serves your purpose.
- In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived form this class.

Example

```
>>> class CustomError(Exception):
...       pass
...

>>> raise CustomError
Traceback (most recent call last):
...
__main__.CustomError

>>> raise CustomError("An error occurred")
Traceback (most recent call last):
...
__main__.CustomError: An error occurred
```

Example: User-Defined Exception in Python

```
# define Python user-defined exceptions
class Error(Exception):
    """Base class for other exceptions"""
    pass

class ValueTooSmallError(Error):
    """Raised when the input value is too small"""
    pass

class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    pass

# our main program
# user guesses a number until he/she gets it right

# you need to guess this number
```

Example: User-Defined Exception in Python

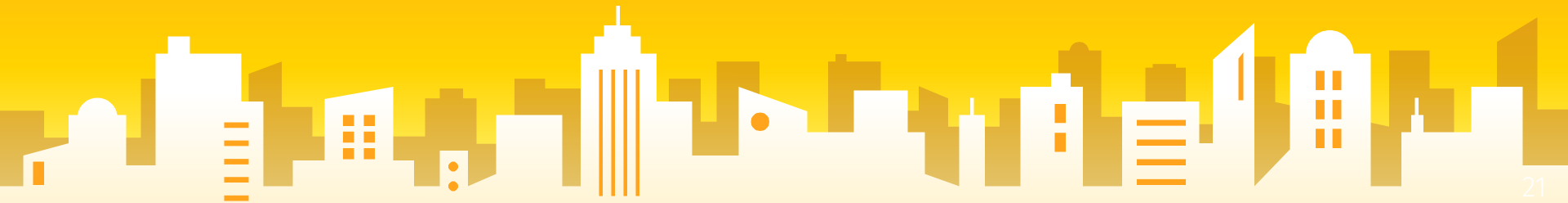
```
number = 10

while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
        print()
    except ValueTooLargeError:
        print("This value is too large, try again!")
        print()

print("Congratulations! You guessed it correctly.")
```

Write a program to guess the correct number using exception.

Assignment



Python Networking

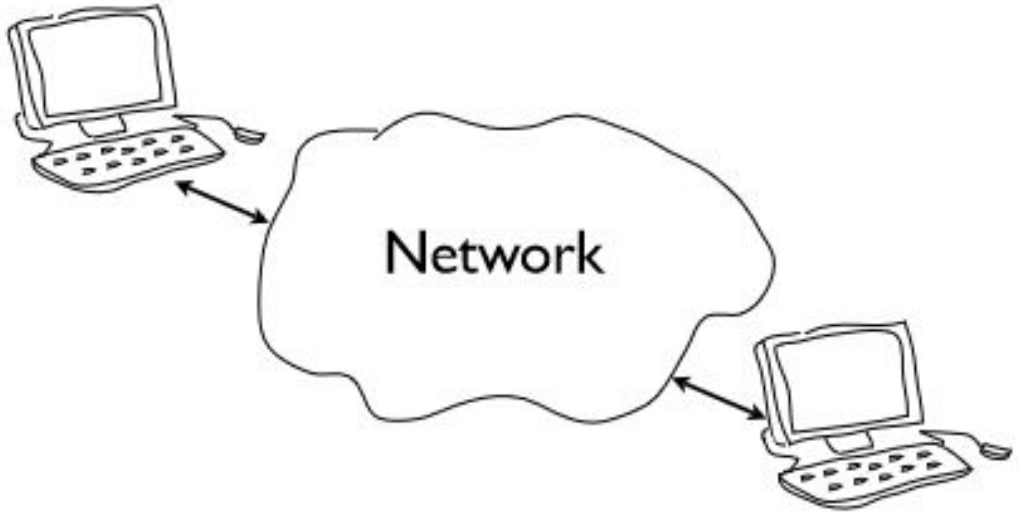
Second Lesson

Python Networking

- Network programming is a major use of Python
- Python standard library has wide support for network protocols, data encoding/decoding, and other things you need to make it work
- Writing network programs in Python tends to be substantially easier than in C/C++

The Problem

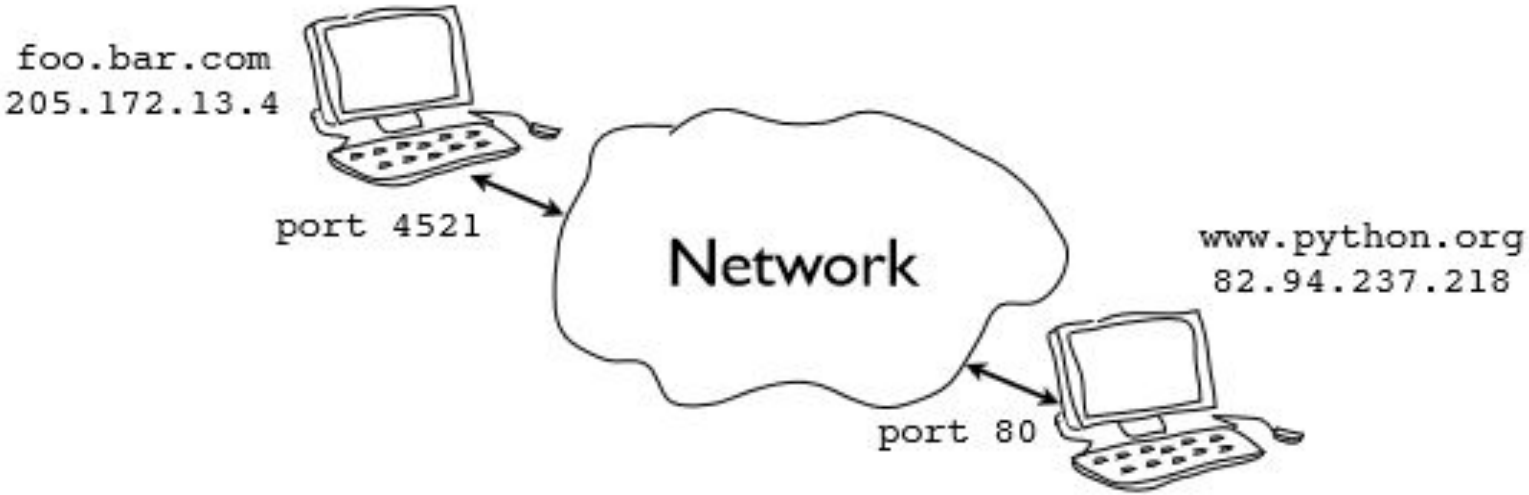
- Communication between computers



- It's just sending/receiving bits

Network Addressing

- Machines have a hostname and IP address
- Programs/services have port numbers



Standard Ports

- Ports for common services are preassigned

21	FTP
22	SSH
23	Telnet
25	SMTP (Mail)
80	HTTP (Web)
110	POP3 (Mail)
119	NNTP (News)
443	HTTPS (web)

- Other port numbers may just be randomly assigned to programs by the operating system

Connections

- Each endpoint of a network connection is always represented by a host and port #
- In Python you write it out as a tuple (host,port)

```
("www.python.org",80)
("205.172.13.4",443)
```

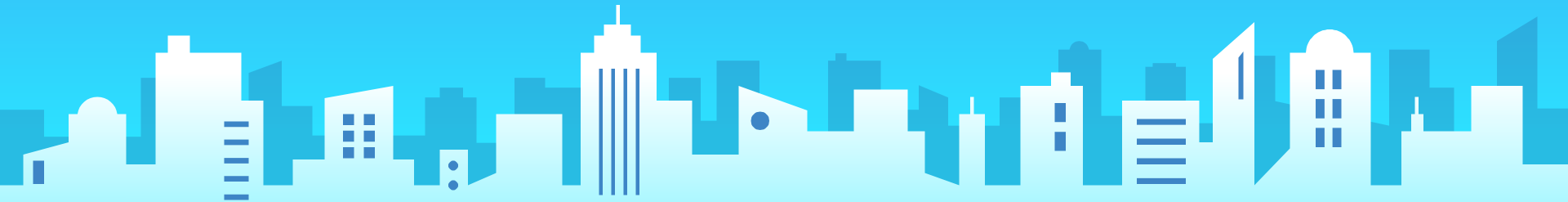
- In almost all of the network programs you'll write, you use this convention to specify a network address

Client/Server Concept

- Each endpoint is a running program
- Servers wait for incoming connections and provide a service (e.g., web, mail, etc.)
- Clients make connections to servers

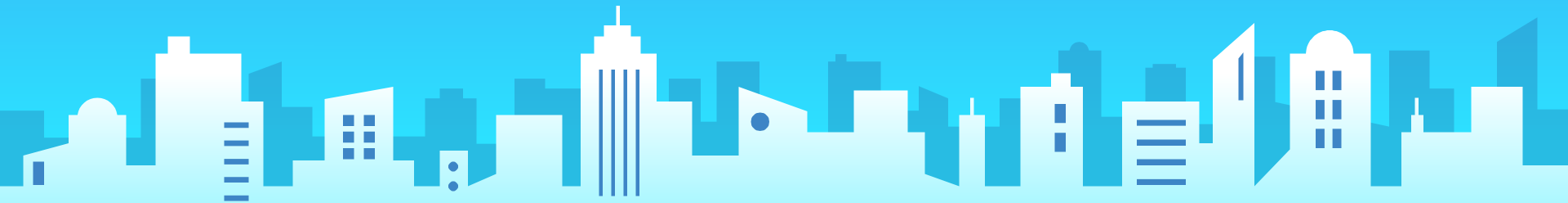
The GET Request (requests Module)

```
requests.get('https://api.github.com')
response = requests.get('https://api.github.com')
print(response.text)
```



Status Codes

```
if response.status_code == 200:
    print('Success!')
elif response.status_code == 404:
    print('Not Found.')
```

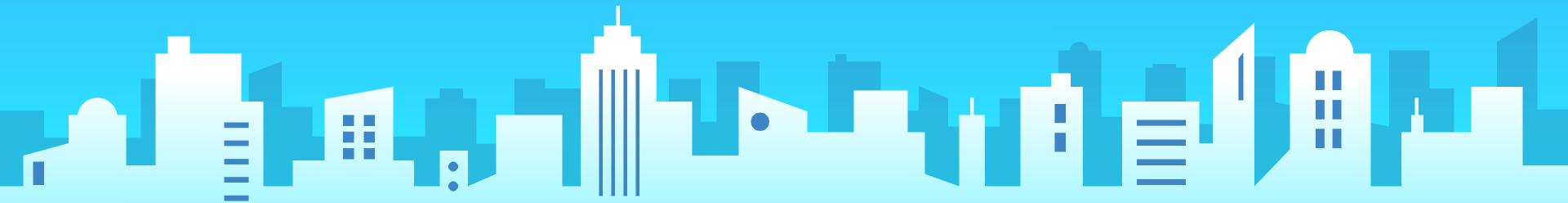


Request with Parameters

```
# Search GitHub's repositories for requests
response = requests.get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
)
# Inspect some attributes of the `requests` repository
json_response = response.json()
repository = json_response['items'][0]
print(f'Repository name: {repository["name"]}')  # Python 3.6+
print(f'Repository description: {repository["description"]}')
```

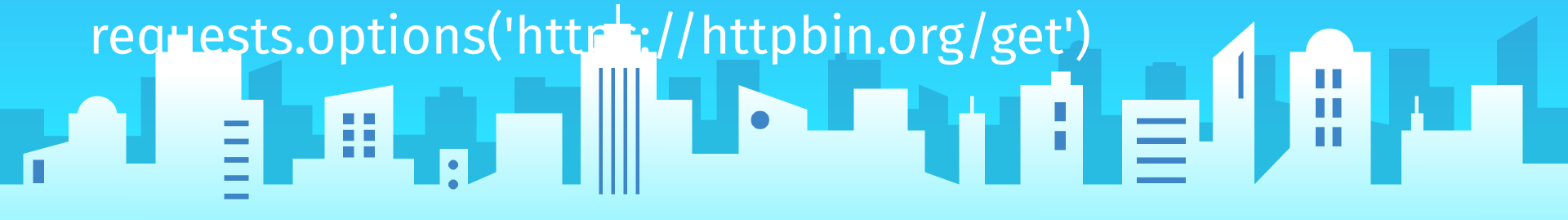
Request Headers

```
response = requests.get(
    'https://api.github.com/search/repositories',
    params={'q': 'requests+language:python'},
    headers={'Accept':
'application/vnd.github.v3.text-match+json'},
  )
```



Other Methods

```
requests.post('https://httpbin.org/post', data={'key':'value'})
requests.put('https://httpbin.org/put', data={'key':'value'})
requests.delete('https://httpbin.org/delete')
requests.head('https://httpbin.org/get')
requests.patch('https://httpbin.org/patch',data={'key':'value
'})
requests.options('https://httpbin.org/get')
```



Thank you

Miss You...