zipfile module

- Your Python programs can both create and open (or extract) ZIP files using functions in the `zipfile` module

command	meaning
`exampleZip = zipfile.ZipFile('example.zip‘, mode = ‘r’)`	Opens the example.zip file as ZipFile object in read mode. (can be changed as in `open()` command)
`exampleZip.namelist()`	returns a list of strings for all the files and folders contained in the ZIP file.
`exampleZip.extractall()` OR `exampleZip.extractall('C:\\my_folder')`	extracts all the files and folders from a ZIP file into the current working directory (or the one you specified)
`exampleZip.extract(‘test.txt’)`	will extract a single file from the ZIP file.

Creating and adding to zip file

- This code will create a new ZIP file named new.zip that has the compressed contents of spam.txt.

  ```
  >> import zipfile
  >> newZip = zipfile.ZipFile('new.zip', 'w')
  >> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
  >> newZip.close()
  ```

- The second argument is the compression type parameter, which tells the computer what algorithm it should use to compress the files; you can always just set this value to `zipfile.ZIP_DEFLATED`. (This specifies the *deflate* compression algorithm, which works well on all types of data.)

time module

- Your computer's system clock is set to a specific date, time, and time zone. The built-in time module allows your Python programs to read the system clock for the current time.
- The most common time reference in programming: Unix epoch (12 AM on January 1, 1970)

Functions	meaning
`time.time()`	returns the number of seconds since Unix epoch as a float value (this number is called *epoch timestamp*)
`time.sleep()`	If you need to pause your program for a while, call the time.sleep() function and pass it the number of seconds you want your program to stay paused

subprocess module

- Your Python program can start other programs on your computer with the `Popen()` function in the built-in subprocess module. (The `P` in the name of the `Popen()` function stands for process.)

Functions	meaning
`subprocess.Popen()`	If you want to start an external program from your Python script, pass the program's filename to subprocess.Popen()

- Example: opening the calculator program in windows:

```
>> import subprocess
>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
```

subprocess module

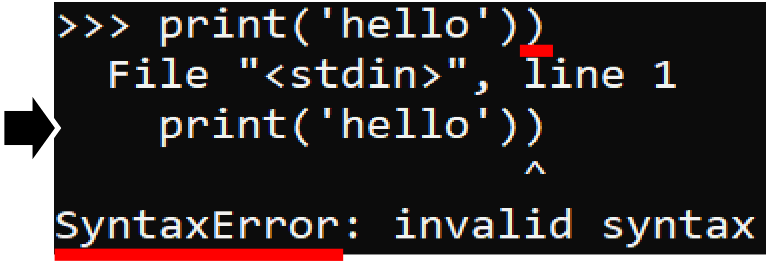
- Opening files with default application:
  - on windows: `subprocess.Popen(['start', 'hello.txt'], shell=True)`
  - on mac-OS: `subprocess.Popen(['open', '/Applications/Calculator.app/'])`

- Example: opening a sound file
  - `subprocess.Popen(['start', ‘tavalod.mp3'], shell=True)`

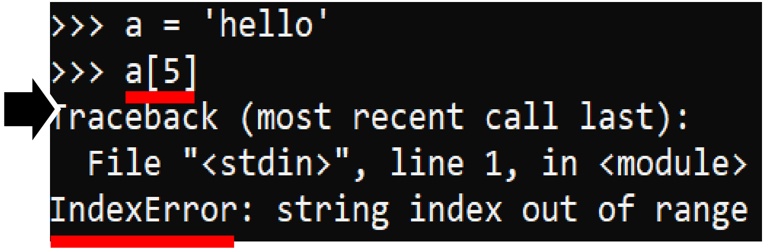
Errors and exceptions

- We have two types of errors in python:
    - Syntax error
    - Exception error

1. *Syntax error:* this type of error occurs whenever you write a syntactically wrong statement.
2. *Exception error:* this type of error occurs whenever syntactically correct Python code results in an error.



```
>>> print('hello'))
  File "<stdin>", line 1
    print('hello'))
                  ^
SyntaxError: invalid syntax
```



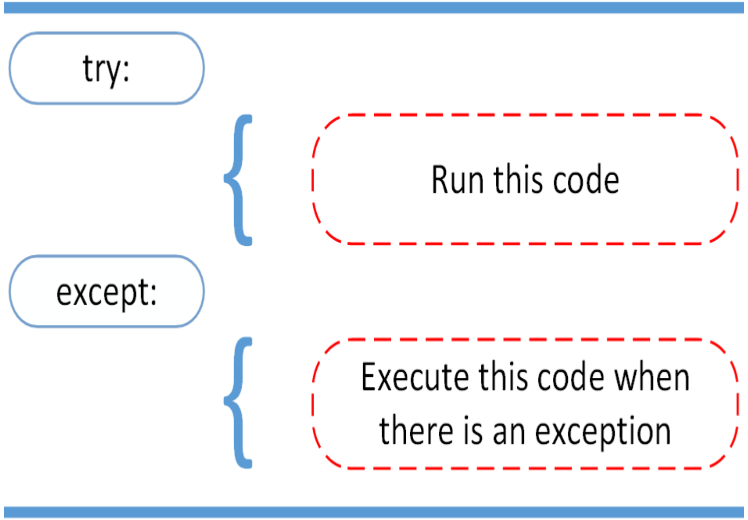
```
>>> a = 'hello'
>>> a[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Errors and exceptions

- There are various numbers of exception errors (or exceptions) in python. In the last slide example, it was an `IndexError`.

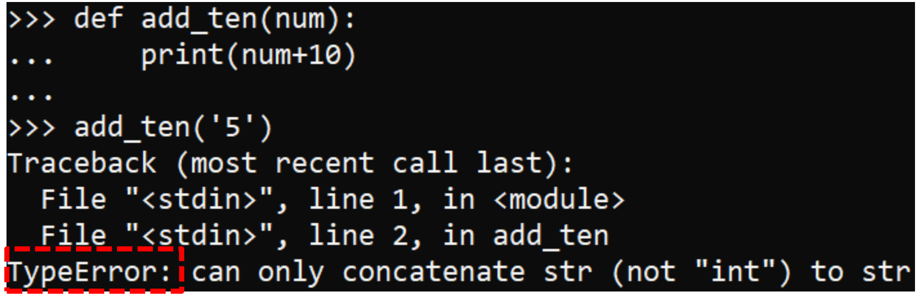
- The `try` and `except` block in Python is used to catch and handle exceptions. Python executes code following the `try` statement (The code that could potentially have an error is put in a `try` clause)

- The code that follows the except statement is the program's response to any exceptions in the preceding try clause.



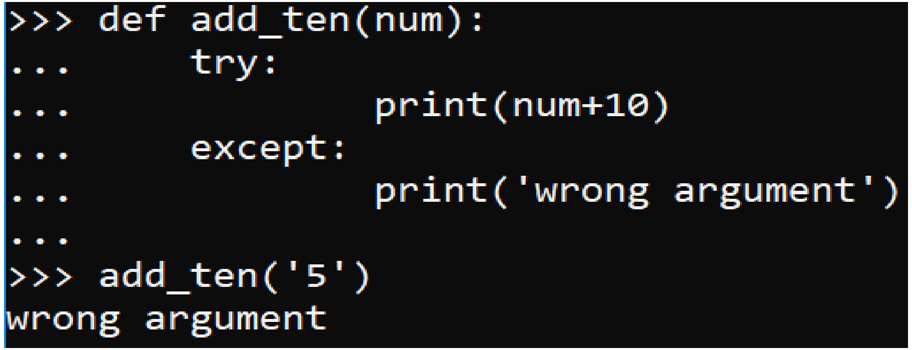
Errors and exceptions

- Example:
  - □ A function without `try`, `except` statements



```
>>> def add_ten(num):
...     print(num+10)
...
>>> add_ten('5')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in add_ten
TypeError: can only concatenate str (not "int") to str
```

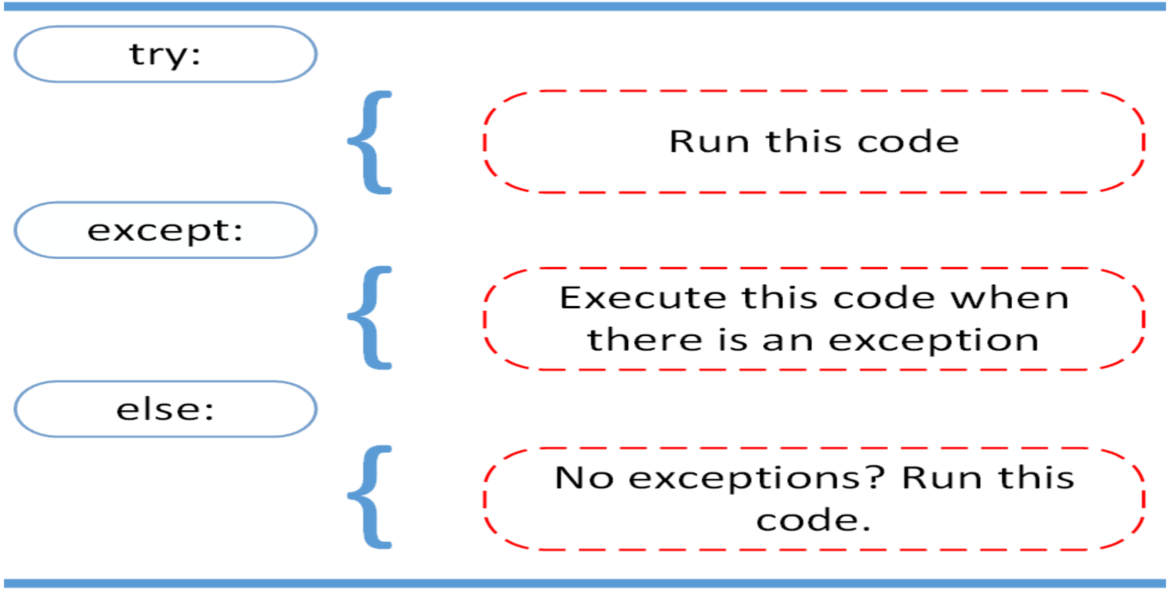
- Example:
  - □ A function with `try`, `except` statements



```
>>> def add_ten(num):
...     try:
...             print(num+10)
...     except:
...             print('wrong argument')
...
>>> add_ten('5')
wrong argument
```

Errors and exceptions

- In Python, using the `else` statement, you can instruct a program to execute a certain block of code only in the absence of exceptions.



try:

Run this code

except:

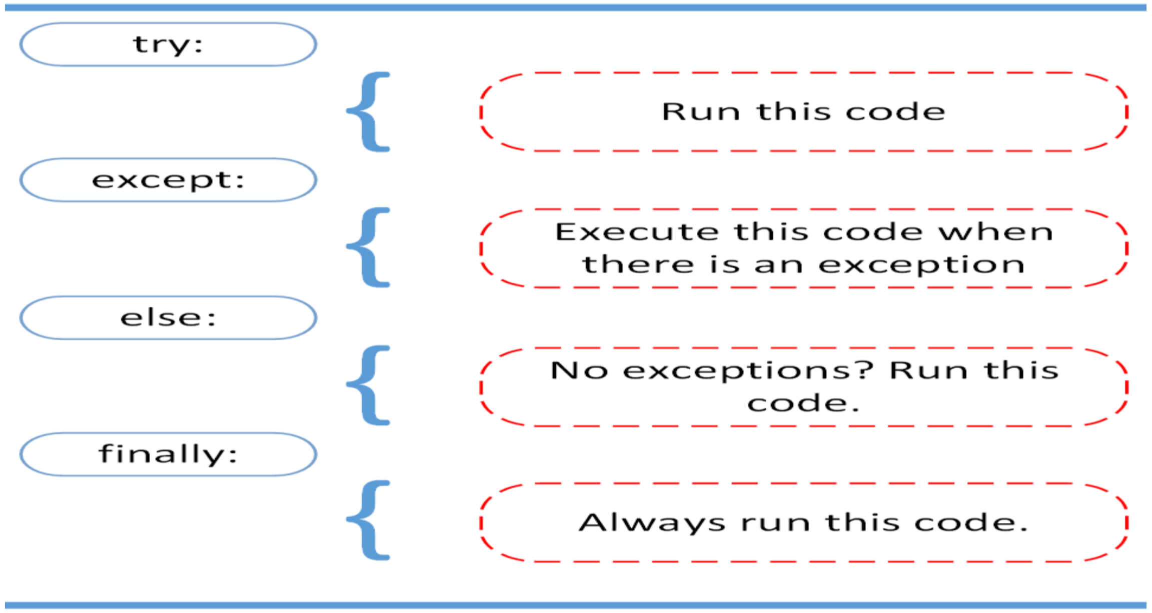
Execute this code when there is an exception

else:

No exceptions? Run this code.

Errors and exceptions

- `finally` enables you to execute sections of code that should always run, with or without any previously encountered exceptions.



try:

Run this code

except:

Execute this code when there is an exception

else:

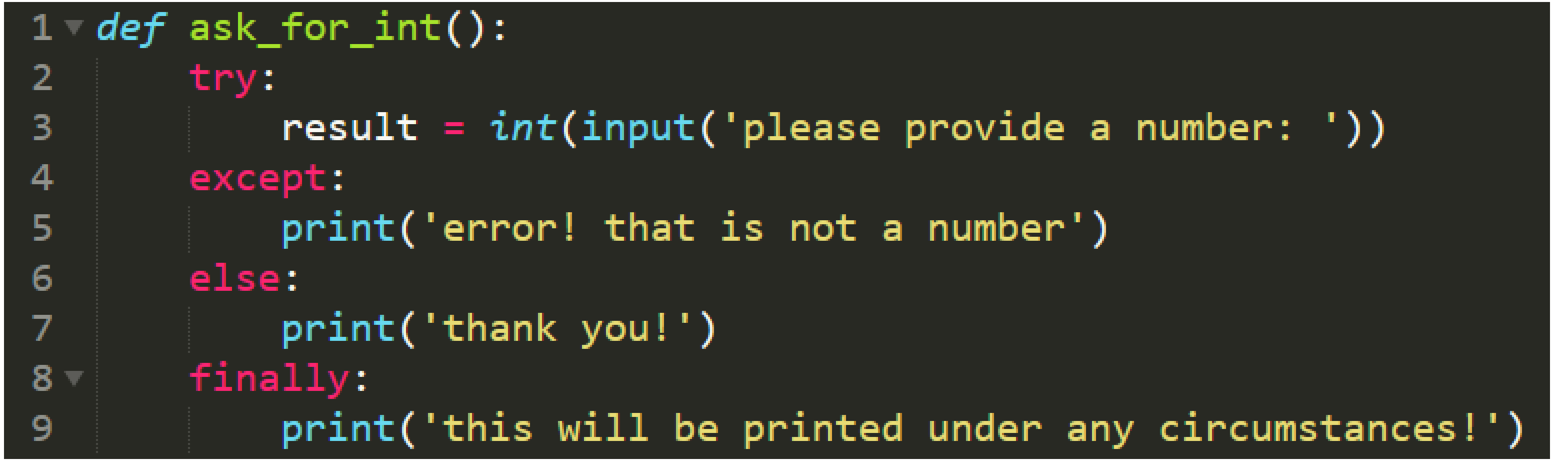
No exceptions? Run this code.

finally:

Always run this code.

Errors and exceptions

- Example:



```
def ask_for_int():
    try:
        result = int(input('please provide a number: '))
    except:
        print('error! that is not a number')
    else:
        print('thank you!')
    finally:
        print('this will be printed under any circumstances!')
```

- Call the ask_for_int function and provide various inputs to see the result.

Sets

- Definition: Unordered collection of unique elements. (just like a set in mathematics)
- Syntax: set (iterable) OR with curly braces {}
  - `Set([1, 2, 3, 1])` OR `{1, 2, 3, 1}` ☞ returns `{1, 2, 3}`
- Examples:
  - `a = {1, 2, 3}`
  - `b = {3, 4, 5}`
  - `len(a)` ☞ returns `3`
  - `4 in b` ☞ returns `True`

Operation	$a \cup b$	$a \cap b$	$a - b$
Python command	`a | b` OR `a.union(b)`	`a & b` OR `a.intersection(b)`	`a - b` OR `a.difference(b)`

map function

- Python map() function is used to apply a function on all the elements of specified iterable and return map object. Python map object is iterable, so we can iterate over its elements.
- Syntax:
  - `map(function, iterable)`
- Example:

  -
    ```
    >> def my_func(num):
    …     return num**2
    >> my_list = [1, 2, 3]
    >> for n in map(my_func, my_list):
    …     print(n)
     1
     4
     9
    ```

filter function

- The filter() method filters the given iterable with the help of a function that tests each element in the iterable to be true or not. The function should return either `True` or `False`.
- Syntax:
  - `filter(function, iterable)`
- Example:

  -
    ```
     >> def my_func(num):
    …      return num%2 == 0
    >> my_list = [1, 2, 3, 4, 5]
    >> for n in filter(my_func, my_list):
    …      print(n)
     2
     4
    ```

lambda expression

- lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are just needed where they have been created.
- Lambda functions are mainly used in combination with the functions filter() and map().
- Example:

normal function:

```
def add_func(x, y):
    return x + y
```

lambda expression:

```
add_func = lambda x, y: x + y
```

lambda expression

- Most of the times, we don’t name the lambda expressions. we just define them wherever we need them.
- Example:

map() with lambda expression:

```
>> lst = [1, 2, 3]
>> for num in map(lambda x:x**2, lst):
       print(num)
1
4
9
```

filter() with lambda expression:

```
>> lst = [1, 2, 3, 4]
>> for num in filter(lambda x:x%2==0,lst):
       print(num)
2
4
```