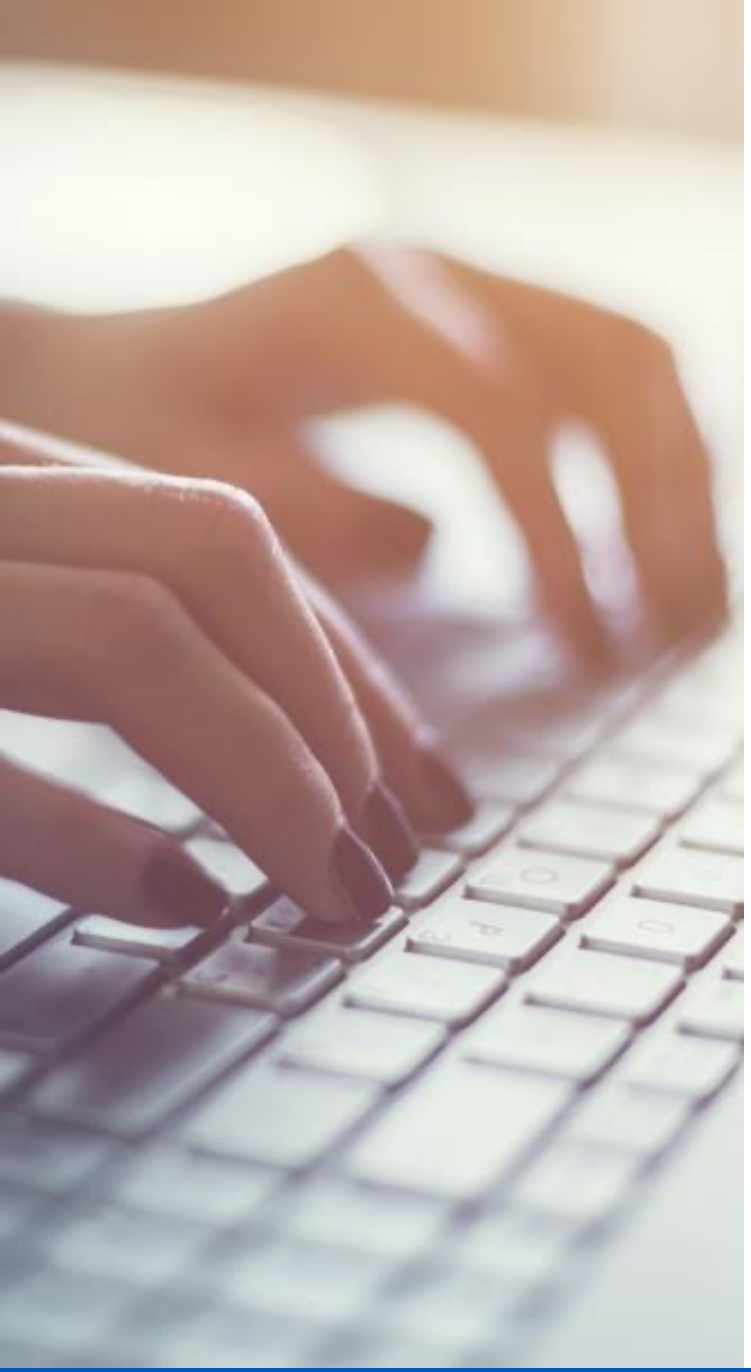
Advanced Python

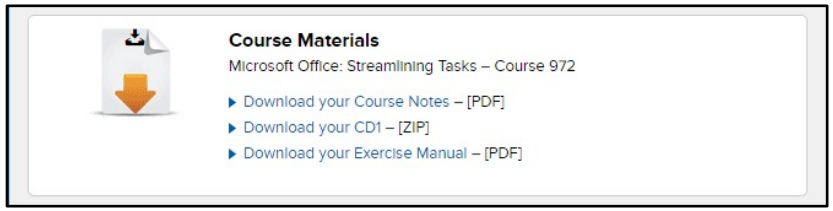
LEARNING TREE INTERNATIONAL



Access Your Course Materials

Do Now

1. Log in to your My Learning Tree Account
2. Find your upcoming course, and click VIEW
3. Under Course Materials, click the links to open your course files



4. Right-click or use the icons to download and save the files to an easily accessible location

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Course load

- **You may need to update the course load on your VM to the latest version**
- **This can be obtained from GitHub**
  - `cd ~/course`
  - `git pull`
- **As this is an advanced course, third-party Python libraries have not been pre-installed**
  - This makes the dependencies more explicit
  - Less "magic"

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Course Objectives

- **Introduce some of Python's more advanced features, covering:**
  - tools
  - libraries
  - methods

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Topics

- Setting Up a Development Environment
- Advanced Language Features
- Command-line UX
- Debugging
- Testing
- Packages
- Concurrency
- Profiling
- Improving Performance

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Additional topics

- Regular Expressions
- Working with Data
- Dashboards

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Setting Up a Development Environment

LEARNING TREE INTERNATIONAL

Contents

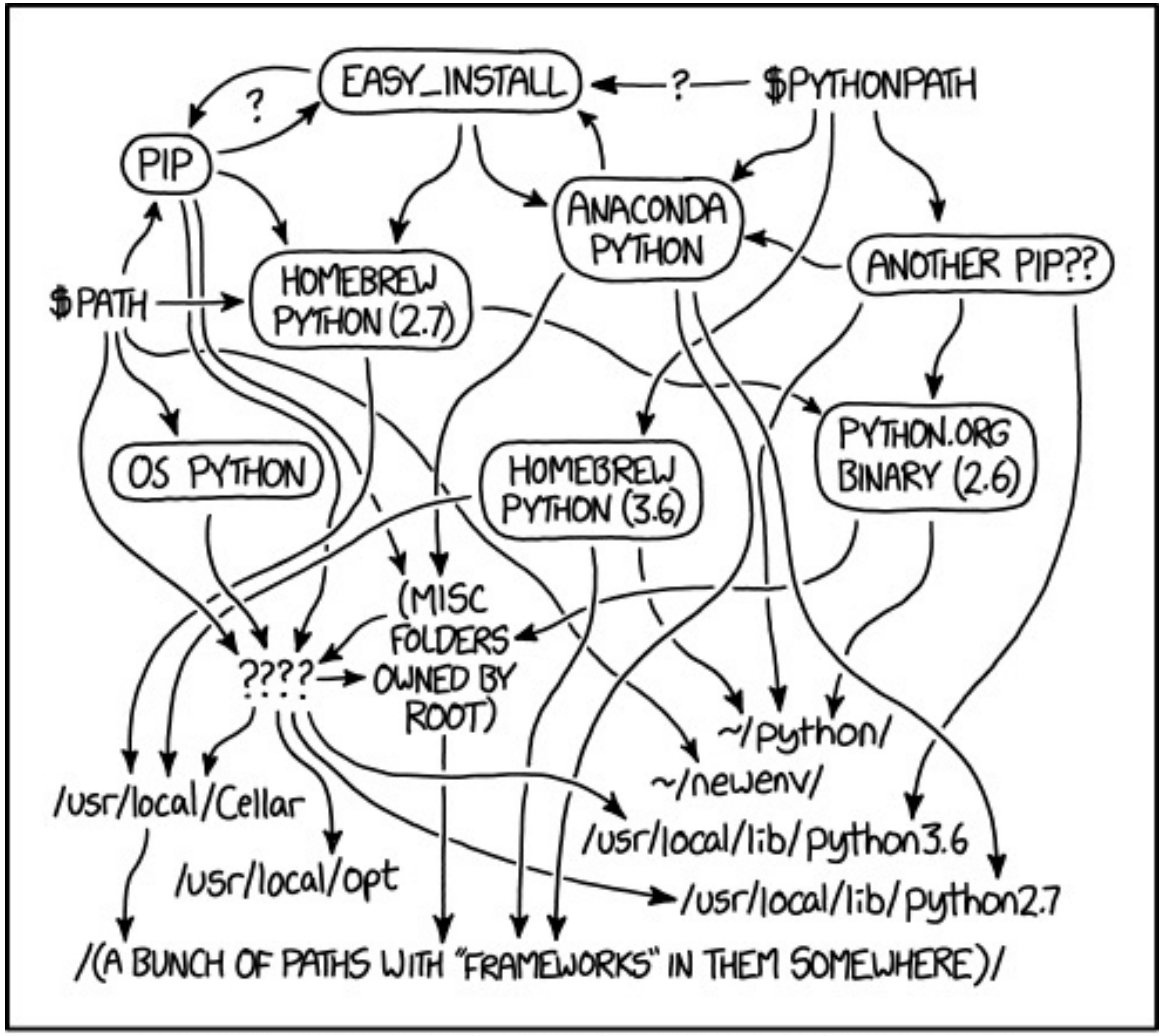
- ► Virtual environments
- ► Configuring the IDE
- ► Structuring projects

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Virtual environments

- **Supports project isolation and dependency management**
- **Just a folder…with three major features**
  - A `site-packages` subfolder where libraries are stored
  - Path modifications to run executables in the virtual environment
  - Script that ensure that we execute our code against the Python interpreter and libraries contained in the virtual environment

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Benefits of virtual environments

- Prevent package sprawl
- Resolve dependency nightmares
- Create self-contained projects
- Increase reproducibility
- Containers (e.g. Docker) offer more advanced isolation than virtual environments

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Additional Ubuntu 20.04 dependencies

```
# Support creating virtual environments via the command line
sudo apt install python3.8-venv

# Support creating virtual environments via PyCharm
sudo apt install python3-distutils
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating virtual environments from the terminal

```
python3 -m venv ./venv
source ./venv/bin/activate
```

```
python -m venv ./venv
./venv/Scripts/activate.bat
```

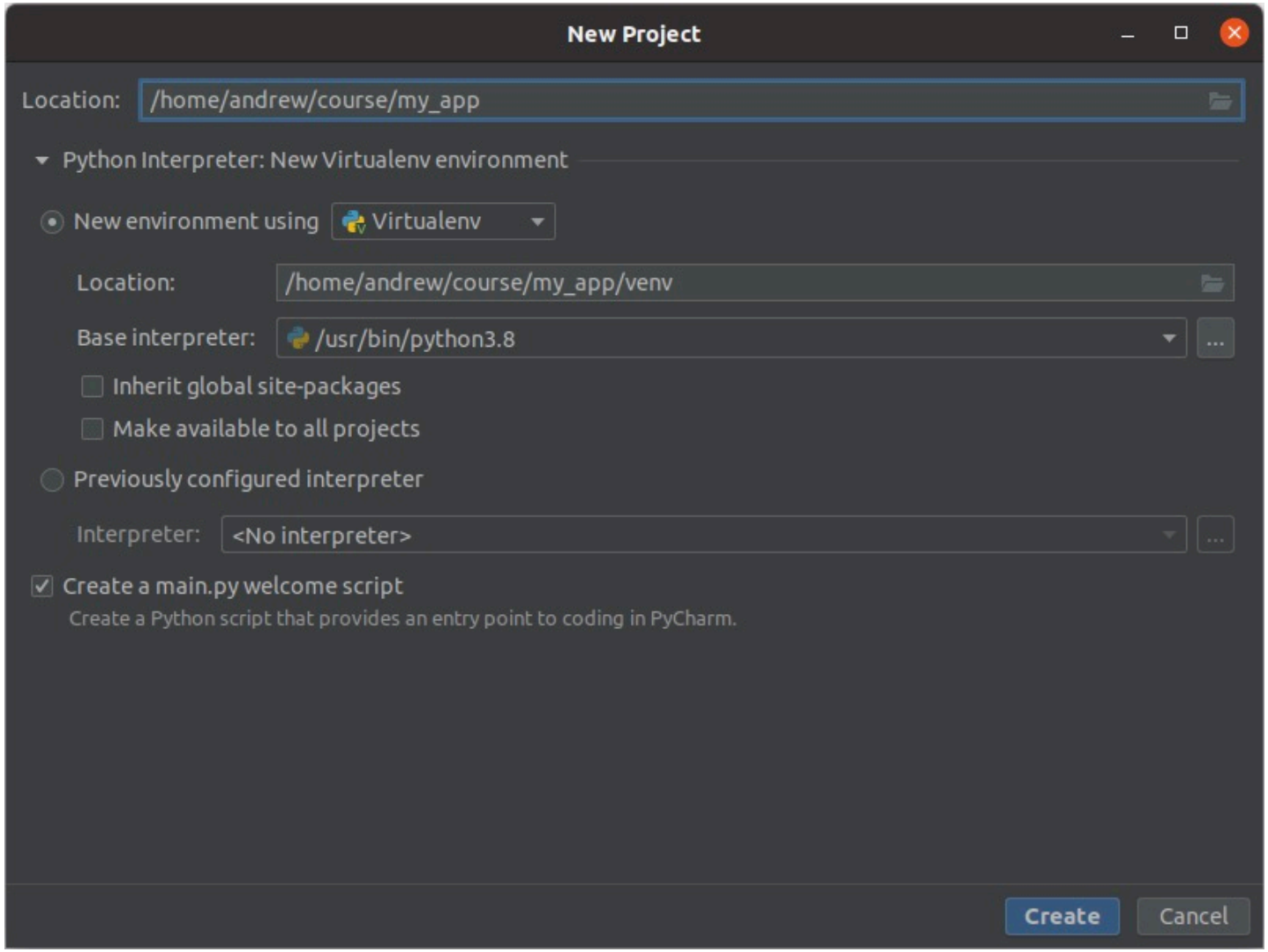
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Deactivating virtual environments

```
# When a virtual environment is active
deactivate
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating virtual environments using PyCharm



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Virtualenv folder structure

```
venv/
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── activate.ps1
│   ├── activate_this.py
│   ├── activate.xsh
│   ├── pip
│   ├── pip3
│   ├── pip-3.8
│   ├── pip3.8
│   ├── python -> /usr/bin/python3.8
│   ├── python3 -> python
│   ├── python3.8 -> python
│   ├── wheel
│   ├── wheel3
│   ├── wheel-3.8
│   └── wheel3.8
├── lib
│   └── python3.8
│       └── site-packages
│           ├── _distutils_hack
│           ├── distutils-precedence.pth
│           ├── pip
│           ├── pip-21.1.2.dist-info
│           ├── pip-21.1.2.virtualenv
│           ├── pkg_resources
│           ├── __pycache__
│           ├── setuptools
│           ├── setuptools-57.0.0.dist-info
│           ├── setuptools-57.0.0.virtualenv
│           ├── _virtualenv.pth
│           ├── _virtualenv.py
│           ├── wheel
│           ├── wheel-0.36.2.dist-info
│           └── wheel-0.36.2.virtualenv
└── pyvenv.cfg
```

```
tree -L 4 ./venv
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cloning virtual environments

```
# Assume activated virtual environment

pip install numpy
pip freeze > requirements.txt
```

```
# requirements.txt
numpy==1.21.2
```

```
# Assume activated virtual environment

pip install -r requirements.txt
```

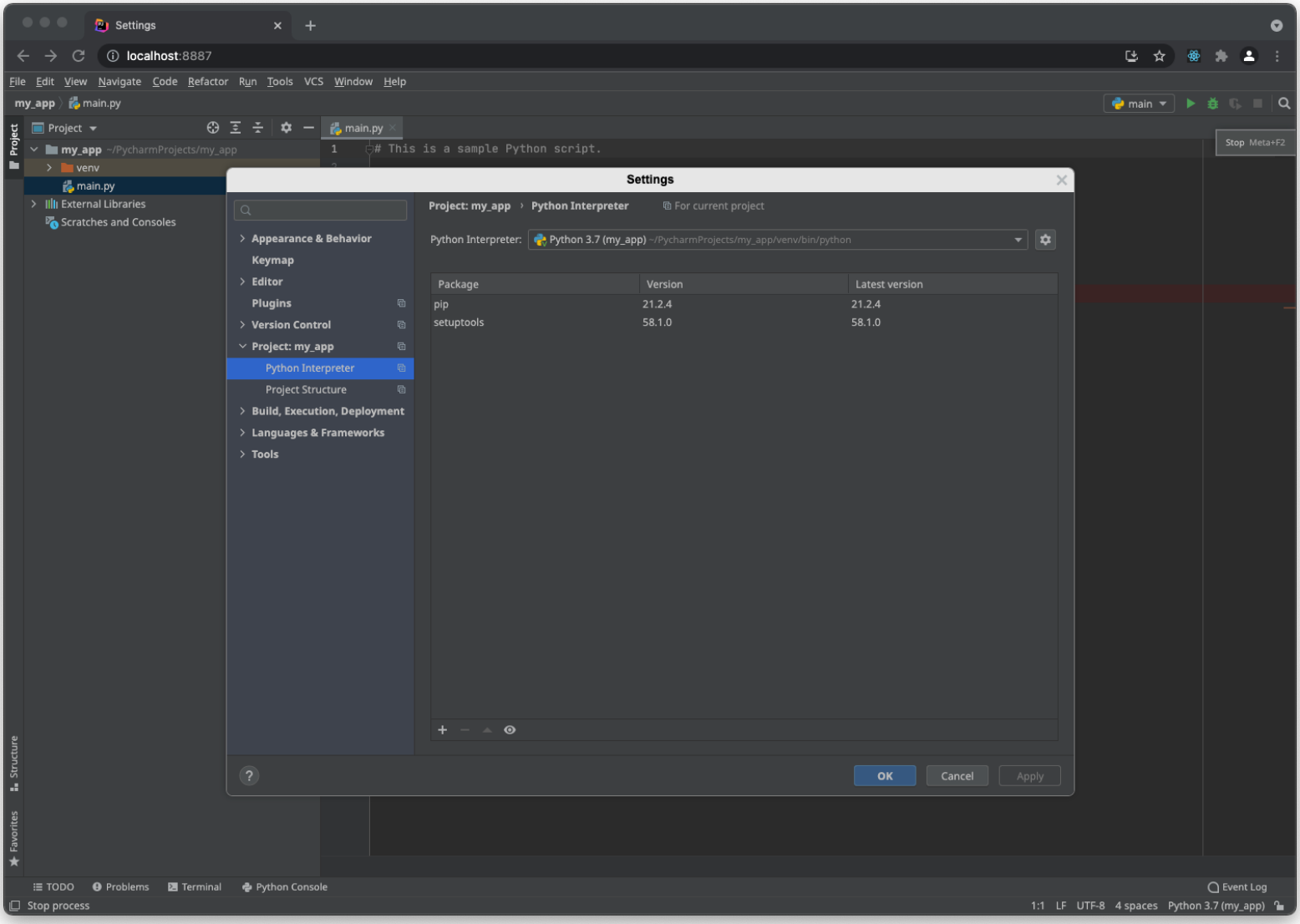
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Alternatives to Virtualenv

- **Virtualenv**
  - Available out of the box (no separate install)
- **Pipenv**
  - Combines virtual environment creation and package management
  - Tracks dependencies
- **Conda**
  - General purpose package manager (not just Python)
  - Provides dependency resolution
- **Poetry**
  - Similar to Pipenv
  - Uses different configuration format (TOML)
- **Pyenv**
  - Makes it easy to manage multiple versions of Python (like NVM for Node.js)
- **Docker**

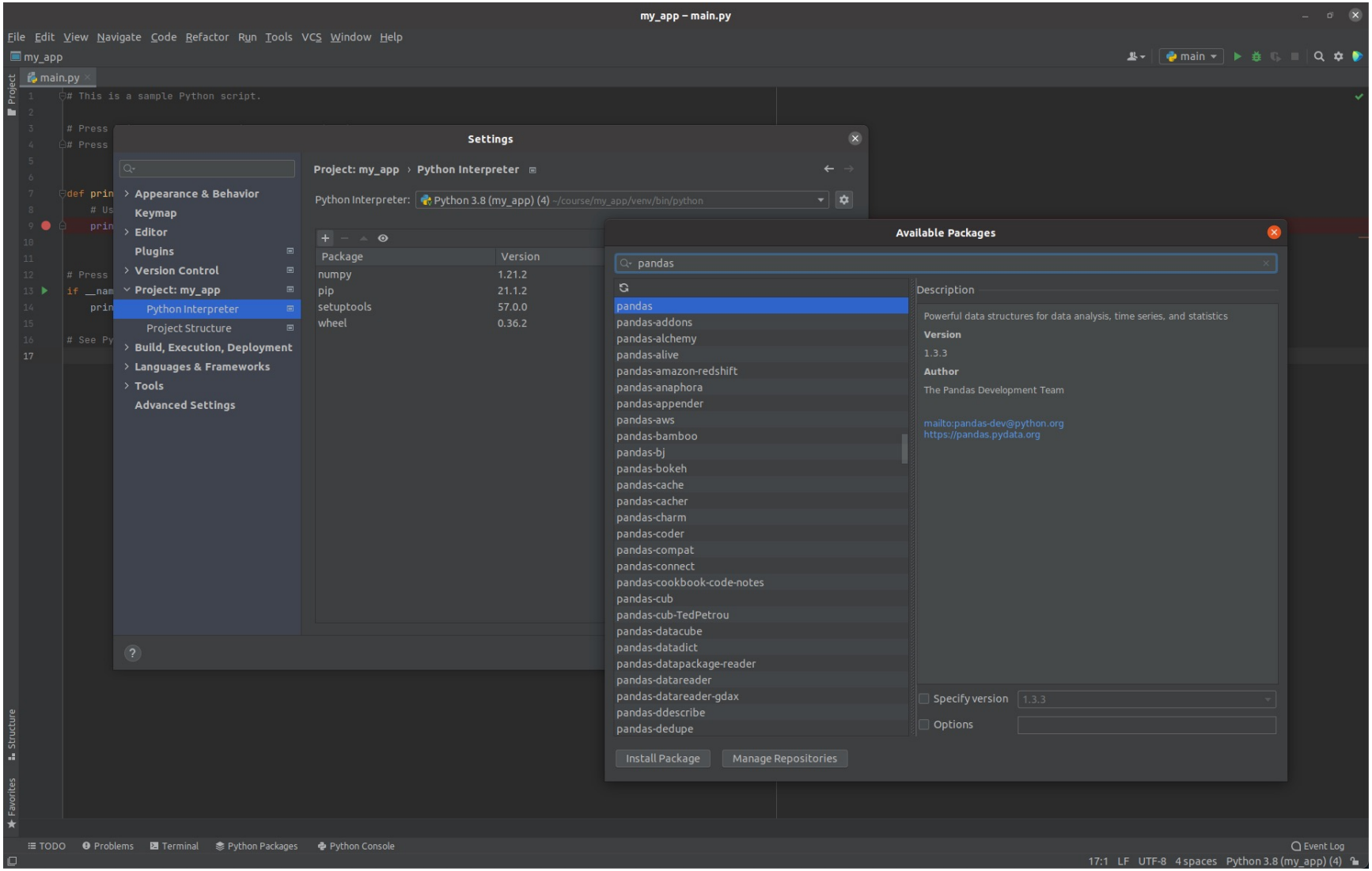
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Running PyCharm *inside* a container



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Installing packages using PyCharm



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Virtual environments
- **Configuring the IDE**
- Structuring projects

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Code formatters

- **Automatically reformat your code based on varies rules**
- **Those rules can be fixed (opinionated) or customisable**
- **Developers no longer make formatting decisions**
  - Just write and save
- **Removes the need for detailed coding style guides**
  - Higher level style guides (e.g. patterns, architecture) are still required
- **Makes code easier to read as blocks have the same visual layout every time**
  - Readable code is easier to debug
- **Easier to onboard new developers are the codebase uses a single style**
- **Reduces “cosmetic” merge conflicts as all developers use the same coding style**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Theme and fonts

- **Default Intellij theme is Darcula**
- **Additional themes can be installed as plug-ins**
  - Cobalt 2
  - Gruvbox
  - Monokai (Sublime Text)
  - Solarized
  - Tomorrow
- **JetBrains developed their own coding font (JetBrains Mono)**
  - Supports ligatures
- **Fira Code is another popular coding font**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Python code formatters

- **autopep8**
  - An older formatter
  - Formats code to adhere to PEP8 standards, but doesn’t focus as much on style
- **black**
  - Opinionated formatter---in the vein of Prettier in the JavaScript ecosystem
  - Formats to PEP8 standards, but also formats heavily for style
- **yapf**
  - Developed by Google
  - Highly configuration
  - Doesn’t format to PEP8 standards out of the box

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Configure black in PyCharm

1. **Install black**
   - `pip install black`
2. **Locate black**
   - `which black` (Linux)
   - `where black` (Windows)
3. **Open External tools in PyCharm**
   - File | Settings | Tools | External Tools
4. **Add a new external tool**
   - Name: black
   - Description: Black is the uncompromising Python code formatter.
   - Program: <install location from Step 2>
   - Arguments: `"$FilePath$"`
5. **Format file using Tools | External Tools | black**
6. **Alternatively, use the BlackConnect plug-in**
   - https://plugins.jetbrains.com/plugin/14321-blackconnect

https://black.readthedocs.io/en/stable/integrations/editors.html#pycharm-intellij-idea

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Linting

- **Linters analyse source code to flag programming errors or "smells"**
- **The major categories of linting are**
  - Structural (e.g. code formatters)
  - Logical (e.g. unused function, mismatched arguments)
  - Analytical (e.g. lines of code, cyclomatic complexity)
- **PyCharm provides code formatting (structural) and code inspections (logical)**
- **Code inspections can be run on Git commits**
- **Pylint is a popular third-party logical linter**
  - Supported by a PyCharm plug-in (Pylint)
  - Useful if you require linting that is consistent across different IDEs
- **Radon is a full-featured analytical linter for Python code**
  - Can be configured in PyCharm as an external tool

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Virtual environments
- Configuring the IDE
- **Structuring projects**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Guard statement

```
def main():
    print("I'm alive!")


if __name__ == "__main__":
    main()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Project structure

```
.
├── app
│   ├── __init.py__
│   ├── main.py
│   └── utils.py
├── docs
│   ├── conf.py
│   └── index.md
├── LICENSE
├── README.md
├── requirements.txt
├── setup.py
└── tests
    ├── test_advanced.py
    └── test_basic.py
```

pyscaffold can be used to scaffold more complex projects

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Simple CLI app project structure

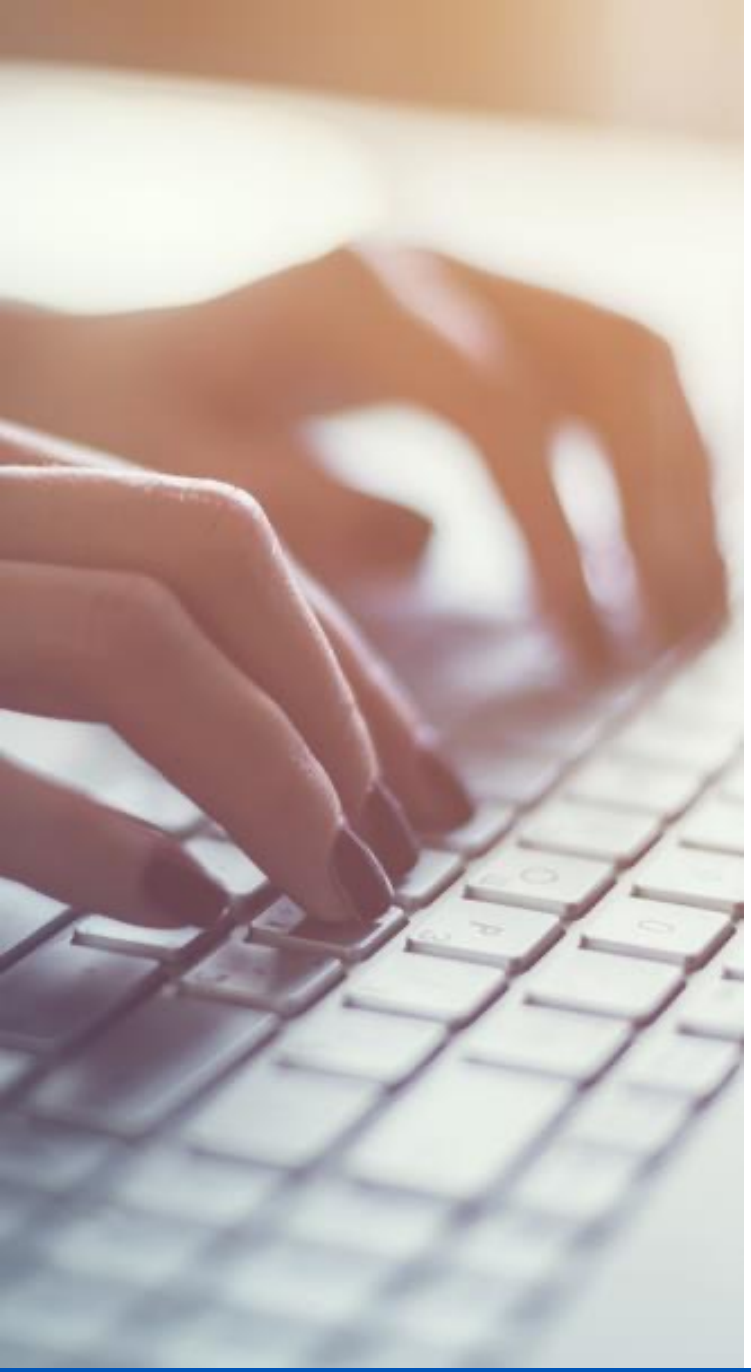
```
cli_app
├── .gitignore
├── LICENSE
├── main.py
├── README.md
├── requirements.txt
├── setup.py
└── tests.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cookiecutter

```
pip install cookiecutter
git clone https://github.com/drivendata/cookiecutter-data-science.git
cookiecutter cookiecutter-data-science/

# Popular templates
# - full-stack-fastapi-postgresql
# - cookiecutter-data-science
# - cookiecutter-flask
# - fastapi-react
# - Python Packages Project Generator
# - copier
# - cookiecutter-django-vue-graphql-aws
# - cookietemple
# - cookiecutter-django-wagtail
# - cookiecutter-qt-app
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a new project in PyCharm

Try it for yourself

- **Create a new project, with a new virtual environment**
- **Add a couple of packages using `pip` or PyCharm (e.g. pyscaffold, numpy)**
  - Check that the packages are in your virtual environment's site_packages folder
- **Write a simple program that uses one of your installed packages**
- **Freeze your environment as a requirements.txt file**
- **Create a new project using the cookiecutter-data-science template**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Isolate dependencies using virtual environments (or containers)**
- **Lean on tooling**
  - IDEs
  - Code formatters
  - Linters
- **Use guard statements to facilitate code reuse**
- **Agree on standard project structures/templates**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Advanced Language Features

LEARNING TREE INTERNATIONAL

Contents

- ▶ **Attributes**
- ▶ Closures
- ▶ Context managers
- ▶ Decorators
- ▶ Monkey patching
- ▶ Metaprogramming
- ▶ Data classes
- ▶ Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Attributes

```
def streaming_average(n):
    streaming_average.count += 1
    streaming_average.total += n

    return streaming_average.total / streaming_average.count

streaming_average.count = 0
streaming_average.total = 0

print(streaming_average(2))
print(streaming_average(3))
print(streaming_average(4))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Attributes
- **Closures**
- Context managers
- Decorators
- Monkey patching
- Metaprogramming
- Data classes
- Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Closures

- **Combination of function and its surrounding state**
  - State is the lexical environment
- **Created by defining a function within a function**
  - The inner function can access the outer function's variables---even after the outer function returns
- **Lightweight alternative to classes**
- **Provide data encapsulation**
  - No need for global state
  - Preferable to exposing function attributes
- **Functions can be stateful**
- **Can be used to achieve partial application and currying**
  - Techniques used extensively in functional programming

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Closures in Python

```python
def make_streaming_average():
    count = 0
    total = 0

    def inner(n):
        nonlocal count, total

        count += 1
        total += n

        return total / count

    return inner

streaming_average = make_streaming_average()
print(streaming_average(2))
print(streaming_average(3))
print(streaming_average(4))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Sales tax calculator

```
def make_sales_tax_calculator(rate):
    def calculator(amount):
        return amount * rate

    return calculator

vat_calculator = make_sales_tax_calculator(0.2)
print(vat_calculator(100))

new_york_sales_tax_calculator = make_sales_tax_calculator(0.04)
print(new_york_sales_tax_calculator(100))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Closing on references

```
def make_sales_tax_calculator(rates):
    def calculator(amount):
        return amount * (1 + rates["uk"])

    return calculator

rates = {"uk" : 0.2}
vat_calculator = make_sales_tax_calculator(rates)
# vat_calculator = make_sales_tax_calculator(dict(rates)) to fix
print(vat_calculator(100))
rates["uk"] = 0.05
print(vat_calculator(100))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Modifying closure state

```
def make_sales_tax_calculator(rate):
    def calculator(amount):
        return amount * rate

    def change_rate(value):
        nonlocal rate
        rate = value

    calculator.change_rate = change_rate

    return calculator

vat_calculator = make_sales_tax_calculator(0.2)
print(vat_calculator(100))
vat_calculator.change_rate(0.05)
print(vat_calculator(100))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Attributes
- Closures
- **Context managers**
- Decorators
- Monkey patching
- Metaprogramming
- Data classes
- Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Context managers

```
with expression as target:
  do_stuff(target)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Context manager examples

```python
with open("dracula.txt") as f:
  book = f.readlines()

  # File is automatically closed at end of block
```

```python
import pandas as pd

with (
    pd.option_context("display.max_rows", 6),
    pd.option_context("display.max_columns", 4),
):
    # Options are applied to statements in block

    print(df)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Use cases for context managers

- Open and close
- Lock and release
- Change and reset
- Create and delete
- Enter and exit
- Start and stop
- Setup and teardown

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Custom context managers

```
from contextlib import contextmanager
from time import perf_counter

@contextmanager
def timer_context_manager():
    start = perf_counter()
    end = 0.0

    yield lambda: end - start

    end = perf_counter()

from time import sleep

with timer_context_manager() as timer:
    sleep(0.5)

print(timer())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Handling exceptions in context managers

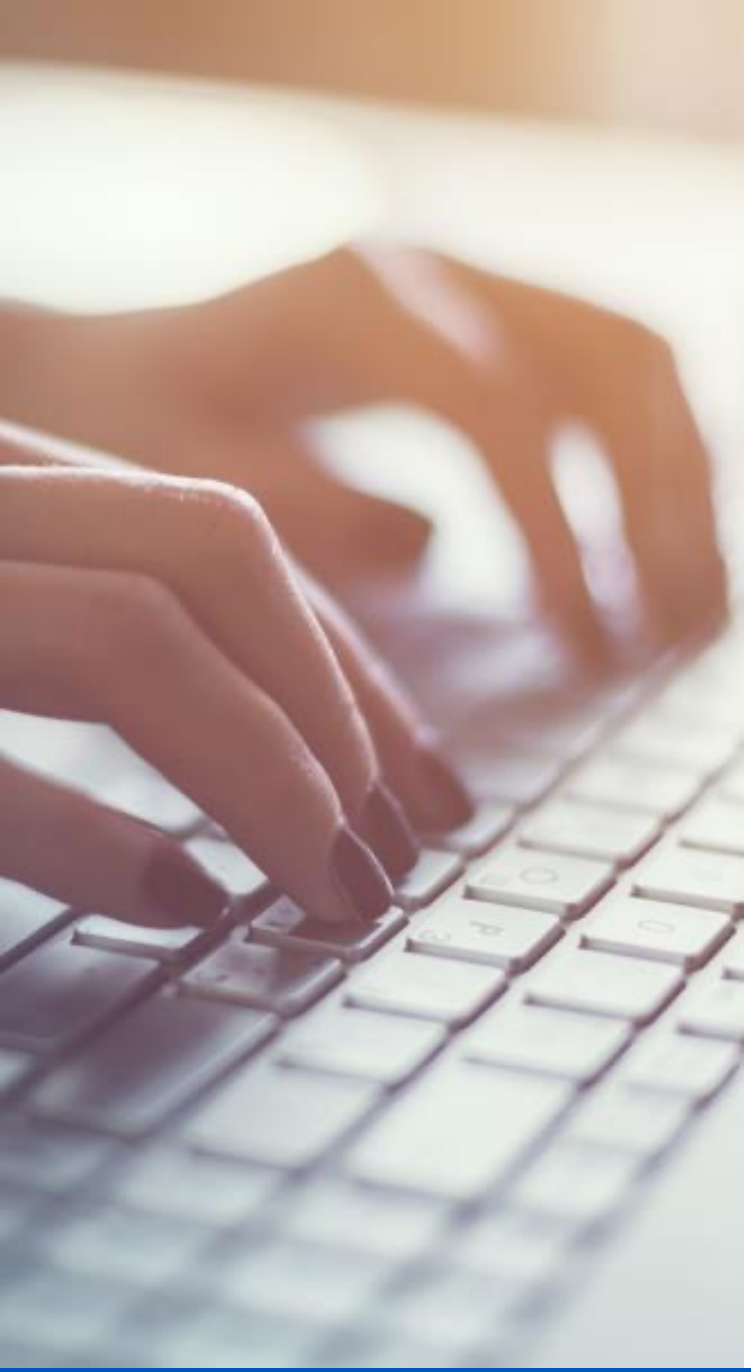
```python
import json

@contextmanager
def json_context_manager(document_path):
    try:
        f = open(document_path)
        document = json.load(f)
    except FileNotFoundError:
        f = None
        document = None

    yield document

    if f:
        f.close()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Using JSON context manager

```
from pprint import pprint as pp

with json_context_manager("missing.json") as document:
    if document:
        pp(document[234])
    else:
        print("Document doesn't exist")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a context manager

Try it for yourself

- **Create a context manager that reports the time when an operation (or sequence of operations) completed**
- **Solution:**
  - context_manager_solution.ipynb

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Attributes
- Closures
- Context managers
- **Decorators**
- Monkey patching
- Metaprogramming
- Data classes
- Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Decorators

```python
def halloween_decorator(func):
    def wrapper():
        print("Boo!")

        func()

    return wrapper

@halloween_decorator
def say_hi():
    print("Hi.")

say_hi()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Decorating functions with arguments

```
def halloween_decorator(func):
    def wrapper(*args, **kwargs):
        print("Boo!")

        func(*args, **kwargs)

    return wrapper

@halloween_decorator
def say_hi(name):
    print(f"Hi, {name}.")

say_hi("Andrew")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Returning values from decorated functions

```python
def halloween_decorator(func):
    def wrapper(*args, **kwargs):
        print("Boo!")

        return func(*args, **kwargs)

    return wrapper

@halloween_decorator
def say_hi(name):
    return f"Hi, {name}."

greeting = say_hi("Andrew")
print("Greeting:", greeting)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Decorators with arguments

```python
def halloween_decorator(exclamation):
    def decorator(func):
        def wrapper(*args, **kwargs):
            print(exclamation)

            func(*args, **kwargs)

        return wrapper

    return decorator

@halloween_decorator("Woo...")
def say_hi():
    print("Hi.")

say_hi()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Preserving decorated function identities

```python
import functools

def halloween_decorator(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        print("Boo!")

        func(*args, **kwargs)

    return wrapper

@halloween_decorator
def say_hi():
    return "Hi."

print(say_hi.__name__)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Stateful decorators

```
def track_invocations_decorator(func):
    invocations = {}

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        count = invocations.get(func.__name__, 0) + 1
        invocations[func.__name__] = count

        print(f"{func.__name__} has been called {count} time(s)")

        func(*args, **kwargs)

    return wrapper
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Using a stateful decorator

```
@track_invocations_decorator
def say_hi():
    print("Hi.")

@track_invocations_decorator
def scare_me():
    print("Boo!")

say_hi()
scare_me()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Debugging decorator

```python
import functools

def debug(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        args_representation = [repr(a) for a in args]
        kwargs_representation = [f"{k}={v!r}" for k, v in kwargs.items()]
        signature = ", ".join(args_representation + kwargs_representation)
        print(f"Calling {func.__name__}({signature})")

        value = func(*args, **kwargs)
        print(f"{func.__name__!r} returned {value!r}")

        return value

    return wrapper
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Using the debugging decorator

```
@debug
def cube(n):
    return n**3

print(cube(n=4))
```

```
import math

factorial = debug(math.factorial)

print(factorial(6))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Caching decorator

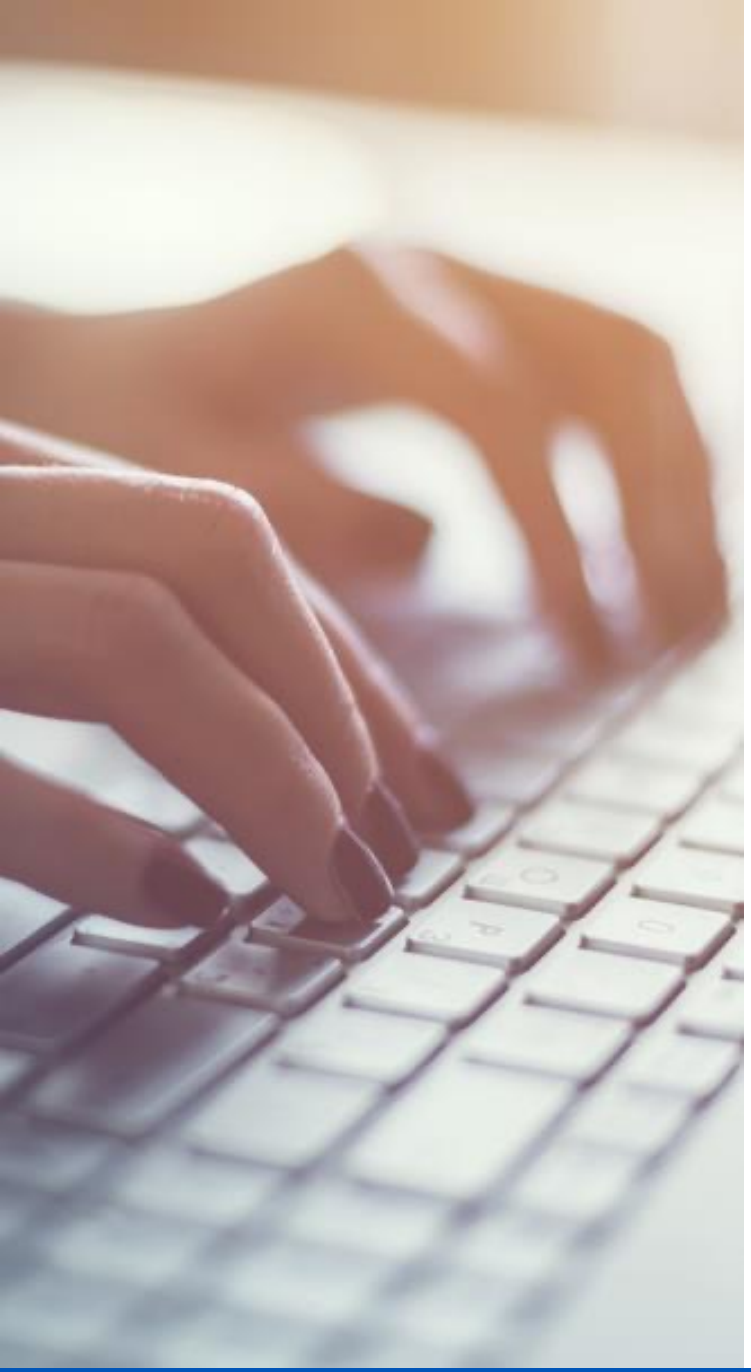
```python
import functools

def cache(func):
    cached_values = {}

    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        key = args + tuple(kwargs.items())
        if key not in cached_values:
            cached_values[key] = func(*args, **kwargs)

        return cached_values[key]

    return wrapper
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Using the caching decorator

```python
import math

@cache
def is_prime(n):
    if n <= 1:
        return False

    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False

    return True

from timeit import timeit

print(timeit(lambda: is_prime(270001008058013), number=1))
print(timeit(lambda: is_prime(270001008058013), number=1))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a timing decorator

Try it for yourself

- **Create a decorator that calculates, and reports, the time taken for a block of code to run to completion**
- **Solution:**
  - decorator_solution.ipynb

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Attributes
- Closures
- Context managers
- Decorators
- **Monkey patching**
- Metaprogramming
- Data classes
- Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Monkey patching

- **Dynamically modify behaviour of code at runtime**
- **Commonly applied to libraries, classes or attributes**
- **Powerful technique, but produces code that is hard to maintain**
- **Don't do it unless you really have to**
- **Use cases include**
  - Mocking/Testing
  - Extending legacy code

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Monkey patching functions

```
import numpy as np

def estimate_project_duration(min, max):
    return np.random.uniform(min, max)

print(estimate_project_duration(5, 10))

np.random.uniform = lambda low, high: np.random.triangular(
    low, (low + high) / 2, high
)

print(estimate_project_duration(5, 10))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Attributes
- Closures
- Context managers
- Decorators
- Monkey patching
- **Metaprogramming**
- Data classes
- Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Metaprogramming

- **Metaprogramming allows a program to inspect and manipulate itself**
- **Python supports metaprogramming through metaclasses**
- **Everything in Python is an object**
  - Classes must have a class which instantiates them
  - Classes have type `type`
- **Classes are instantiated by metaclasses**
- **We can create metaclasses by inheriting from `type`**
- **Types can be created dynamically**
  - Attributes and methods can be added dynamically
- **Most Python developers will never need to use metaclasses**
  - There are usually simpler ways to solve problems

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Dynamically creating classes

```python
import math

def init_circle(self, radius):
    self.radius = radius

Circle = type("Circle", (), {
    "__init__": init_circle,
    "calculate_area": lambda x: math.pi * x.radius ** 2
})

circle = Circle(5)

print(circle.calculate_area())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Plug-in registration

```
class PlugInRegistry(type):
    plug_ins = []
    def __init__(cls, name, bases, attrs):
        if name != "PlugIn":
            PlugInRegistry.plug_ins.append(cls)

class PlugIn(metaclass=PlugInRegistry):
    pass

class ImagePlugIn(PlugIn):
    pass

class AudioPlugIn(PlugIn):
    pass

class VideoPlugIn(PlugIn):
    pass

print(PlugInRegistry.plug_ins)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Metaprogramming use cases

- Plug-in systems
- Decorator management
- Code analysis
- Code generators
- Frameworks
- ORMs
- IDE features
- DSLs

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

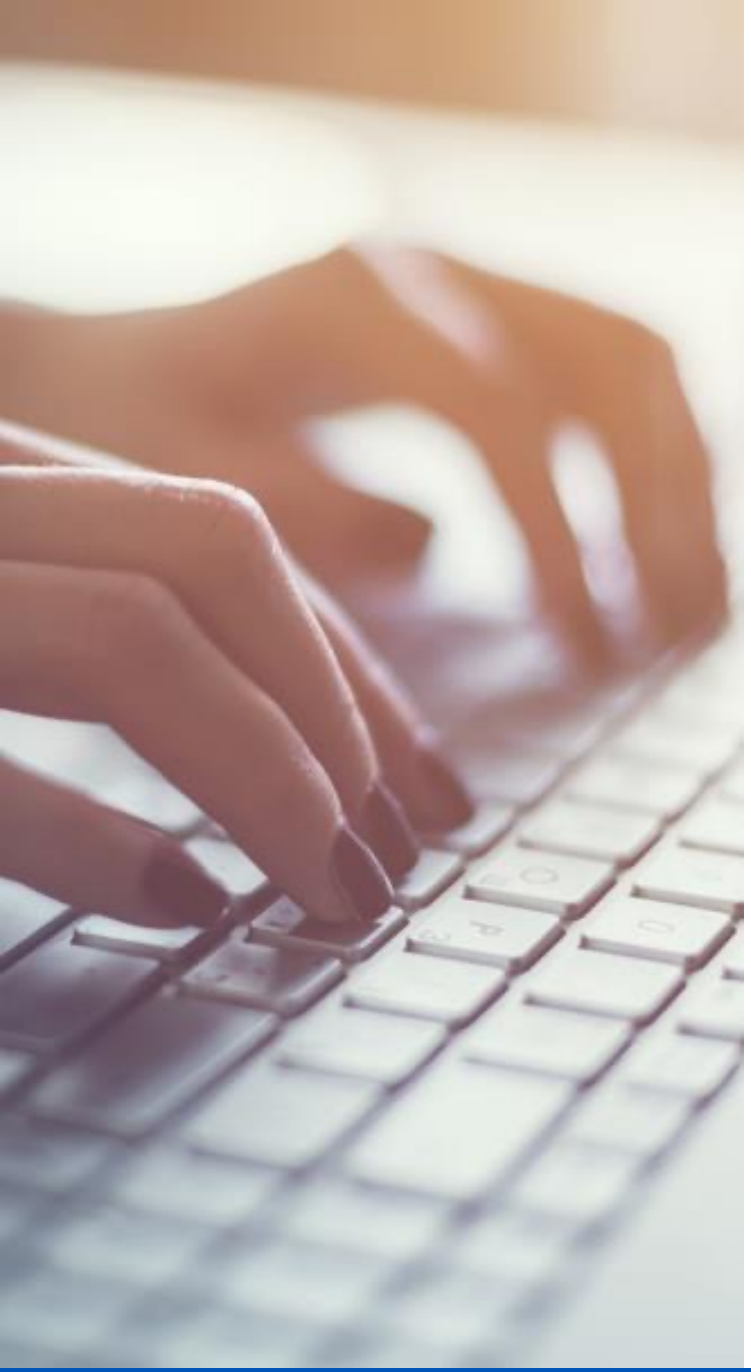
- Attributes
- Closures
- Context managers
- Decorators
- Monkey patching
- Metaprogramming
- **Data classes**
- Structural pattern matching

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Data classes

```python
from dataclasses import dataclass

@dataclass
class Wager:
    selection: str
    kind: str
    price: float

bet1 = Wager("Chelsea", "win", 5.)
bet2 = Wager("Chelsea", "win", 5.)

print(bet1)
print(bet1.selection)
print(bet1 == bet2)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Evaluate a poker hand

Try it for yourself

- **Create a dataclass for a card**
  - Requires suit and value
  - Use numbers for Jack, Queen, King and Ace to keep it simple
- **Create a hand of five cards**
- **Write a function to check if a hand is a full-house**
  - 3 of one kind, 2 of another
- **Solution:**
  - poker_solution.ipynb

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Attributes
- Closures
- Context managers
- Decorators
- Monkey patching
- Metaprogramming
- Data classes
- **Structural pattern matching**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Structural pattern matching

```
file_type = "gif"

match file_type:
    case "gif":
        print("Process JPG file")
    case "jpg":
        print("Process PNG file")
    case "png":
        print("Process JPG file")
    case _:
        print("Process unknown file")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Structural pattern matching with enums

```python
from enum import Enum

class FileType(Enum):
    GIF = 0
    JPG = 1
    PNG = 2

file_type = FileType.GIF

match file_type:
    case FileType.GIF:
        print("Process JPG file")
    case FileType.GIF:
        print("Process PNG file")
    case FileType.GIF:
        print("Process JPG file")
    case _:
        print("Process unknown file")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Structural pattern matching with multiple elements

```
fractional_price = [3, 0]

match fractional_price:
    case [n, 0]:
        print("Infinity")
    case [n, 1]:
        print(n)
    case [n, d]:
        print(n / d)
    case _:
        print("Not a fractional price)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Structural pattern matching with guards

```
fractional_price = [3, -1]

match fractional_price:
    case [n, d] if (n < 0) or (d < 0):
        print("Cannot have negative prices")
    case [n, 0]:
        print("Infinity")
    case [n, 1]:
        print(n)
    case [n, d]:
        print(n / d)
    case _:
        print("Not a fractional price)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- Closures allow us to encapsulate data
- Context managers improve code quality by coordinating actions (e.g. for resource management)
- Decorators allow us to extend the behaviour of our code
- Monkey patching allows us to modify existing code, but can be dangerous
- Metaclasses allow us to do metaprogramming, for the special use cases that require it
- Dataclasses ease working with data
- Structural pattern matching can produce much cleaner code

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Object-Oriented Programming

LEARNING TREE INTERNATIONAL

Object-oriented programming

- **Python is a multi-paradigm programming language**
- **Object-oriented programming (OOP) is based around the creation of objects**
- **Objects contain both data and code**
- **Sits atop three "pillars"**
  - Inheritance (Specialisation)
  - Polymorphism
  - Encapsulation
- **Allows code reuse**
- **Introduced in Smalltalk**
  - Popularised by Java
- **Promoted as reducing the gap between business analysts and programmers**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating classes

```
class BankAccount:
    def __init__(self, balance=0):
        self.__balance = balance

    @property
    def balance(self):
        return self.__balance

    def deposit(self, amount):
        self.__balance += amount

        return self.balance

    def withdraw(self, amount):
        self.__balance -= amount

        return self.balance
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Inheritance (and creating objects)

```
class BankAccountError(Exception):
    pass

class CurrentAccount(BankAccount):
    def __init__(self, balance=0, overdraft=0):
        super().__init__(balance=balance)

        self.__overdraft = overdraft

    def withdraw(self, amount):
        if amount > self.balance + self.__overdraft:
            raise BankAccountError("your balance (inc. overdraft) is too
low")

        return super().withdraw(amount)

current_account = CurrentAccount(overdraft=100)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Extending behaviour

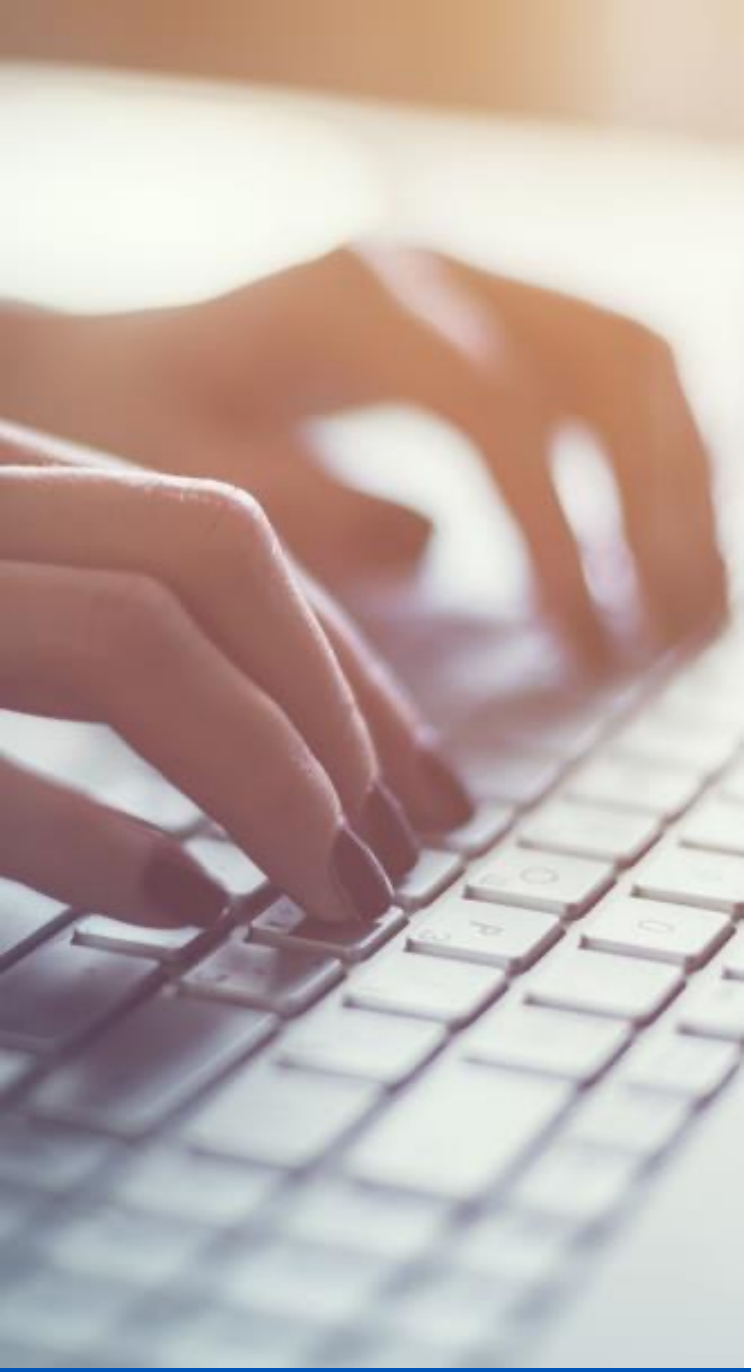
```
class SavingsAccount(BankAccount):
    interest_rate = 0.02

    def withdraw(self, amount):
        if amount > self.balance:
            raise BankAccountError("your balance is too low")

        return super().withdraw(amount)

    def credit_interest(self):
        self.deposit(self.balance * SavingsAccount.interest_rate)

        return self.balance

savings_account = SavingsAccount()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Multiple inheritance

```
class Mortgage:
    pass

class CurrentAccountMortgage(CurrentAccount, Mortgage):
    pass
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a new account class

Try it for yourself

- **Create a starter account aimed at kids**
  - New accounts are opened with a balance of £100
- **Create a `StarterAccount` class to implement this new account**
- **Instantiate and object of this class and check it behaves as expected**
- **Solution:**
  - bank_account_solution.ipynb

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Problems with OOP

- **Inheritance hierarchies tend to be tightly coupled**
  - Makes refactoring very difficult
- **As inheritance hierarchies grow, it's difficult to maintain consistency**
  - Very easy to violate the Liskov Substitution Principle
- **Focus on "things", rather that actions, results in complex patterns**
  - CommandExecutors with a single Command method
  - Various factories, builders and managers
- **Lack of clear separation between state and behaviour complicates concurrent execution**
- **"Object-oriented programming is an exceptionally bad idea which could only have originated in California." -- Edsger Dijkstra**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Python is a multi-paradigm language**
  - It supports OOP, but it's not mandatory
- **OOP tends to be more common in large, LOB apps (e.g. corporate web apps)**
- **Fields like data science and automation tend to naturally favour a functional programming (FP) style**
- **OOP is overkill for small scripts**
  - Procedural programming (or FP) are more suitable

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Command-line UX

LEARNING TREE™
INTERNATIONAL

UX

- Python is very well suited to writing command-line apps
- It can be used in place of shell scripting
- Even simple scripts usually not to parameterised
- Changing code to update parameters is error-prone---and tedious
- When you return to a script after a while, it's not always obvious what parameters are required
- Good UX can reduce amount of documentation required
- UX should form part of any script that isn't going to be deleted within 24 hours

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Command-line argument libraries

- `sys.argv`
  - Built-in
  - Simple
  - Minimal
- **argparse**
  - Batteries included
  - Lacking sophistication
- **Click**
  - Third-party
  - Feature-rich
  - Uses decorators
  - Currently active
  - Docopt is another popular choice

https://click.palletsprojects.com/

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

sys.argv

```python
import sys

if len(sys.argv) != 2:
    print(f"Usage: {sys.argv[0]} <file path>")
else:
    file_path = sys.argv[1]

    with open(file_path) as f:
        lines = f.readlines()
        print(f"{file_path} contains {len(lines)} line(s).")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

argparse

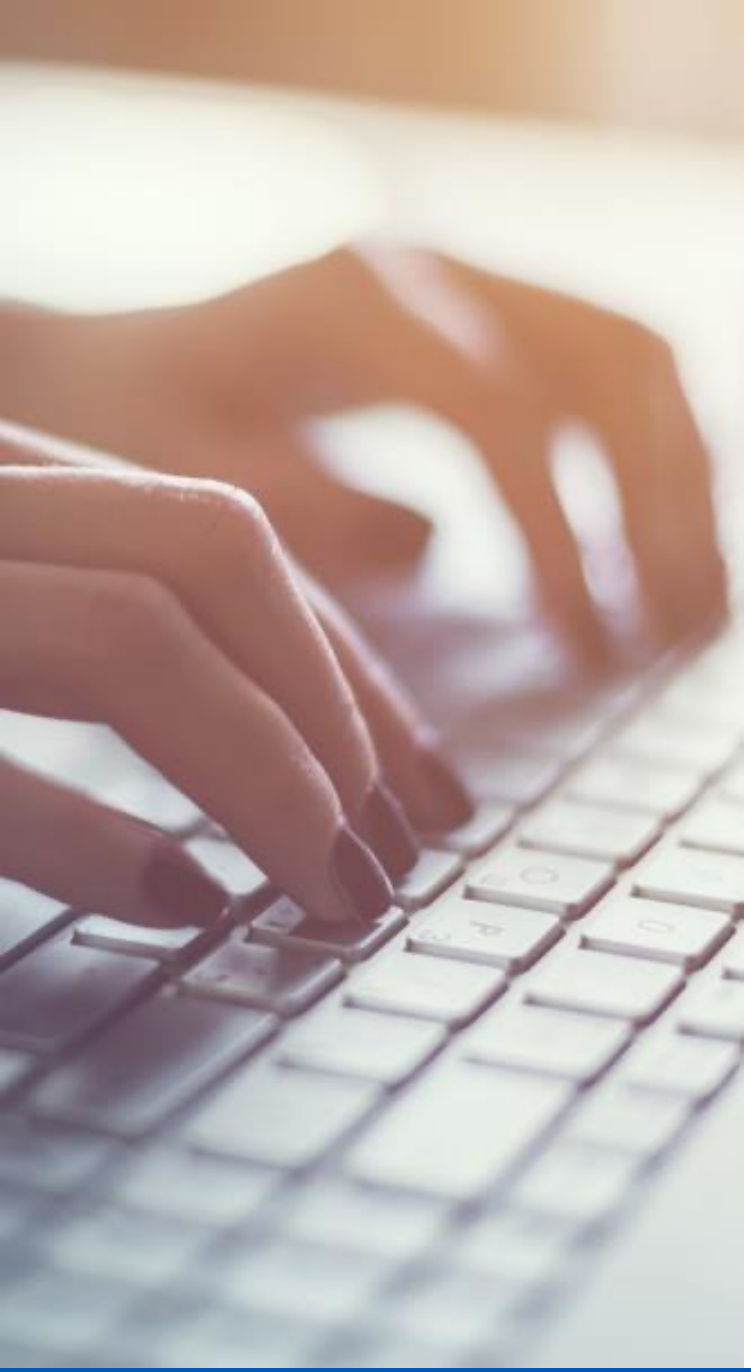
- **Parser is defined using function calls**
  - Can be confusing to "visualise" the options from the code
- **"Magic behaviour" distinguishes between arguments and options**
  - Can cause problems with more complicated requirements
- **Have support for subparsers, allowing the creation of fairly complicated Uis**
- **Being part of the standard distribution means that it's well-documented**
- **Quick to get start**
  - Isn't particularly intuitive when moving beyond the basics

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Click

- **Click is highly configurable**
  - But has sensible defaults to get started
- **It has an extensive range of option types**
  - Flags
  - Multi-value
  - Passwords
  - Ranges
  - Prompting
- **Uses the decorator pattern**
- **Information can be passed to nested commands using contexts**
- **Extensive documentation**
  - https://click.palletsprojects.com/en/8.0.x/#documentation

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a command-line UI

Try it for yourself

- **Create a UI for a metric to imperial conversion app**
- **It should support**
  - miles to km (1.609344 km = 1mile)
  - lbs to kg (0.4535924 kg = 1 lb)
  - °F to °C (°F = °C * 1.8 + 32)
- **A `--imperial` option should support the reverse conversion**
- **Click is recommended, but feel free to use argparse if you feel that better suits your anticipated requirements.**
- **Solution:**
  - metric_imperial_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Command-line apps are fairly common in Python**
- **Most useful apps need to be parameterised**
- **Good UX isn't just for GUI apps**
- **Popular command-line UI libraries include**
  - `sys.argv`
  - argparse
  - Click
- **Click is powerful and flexible, so will pay back the investment in time learning it**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Debugging

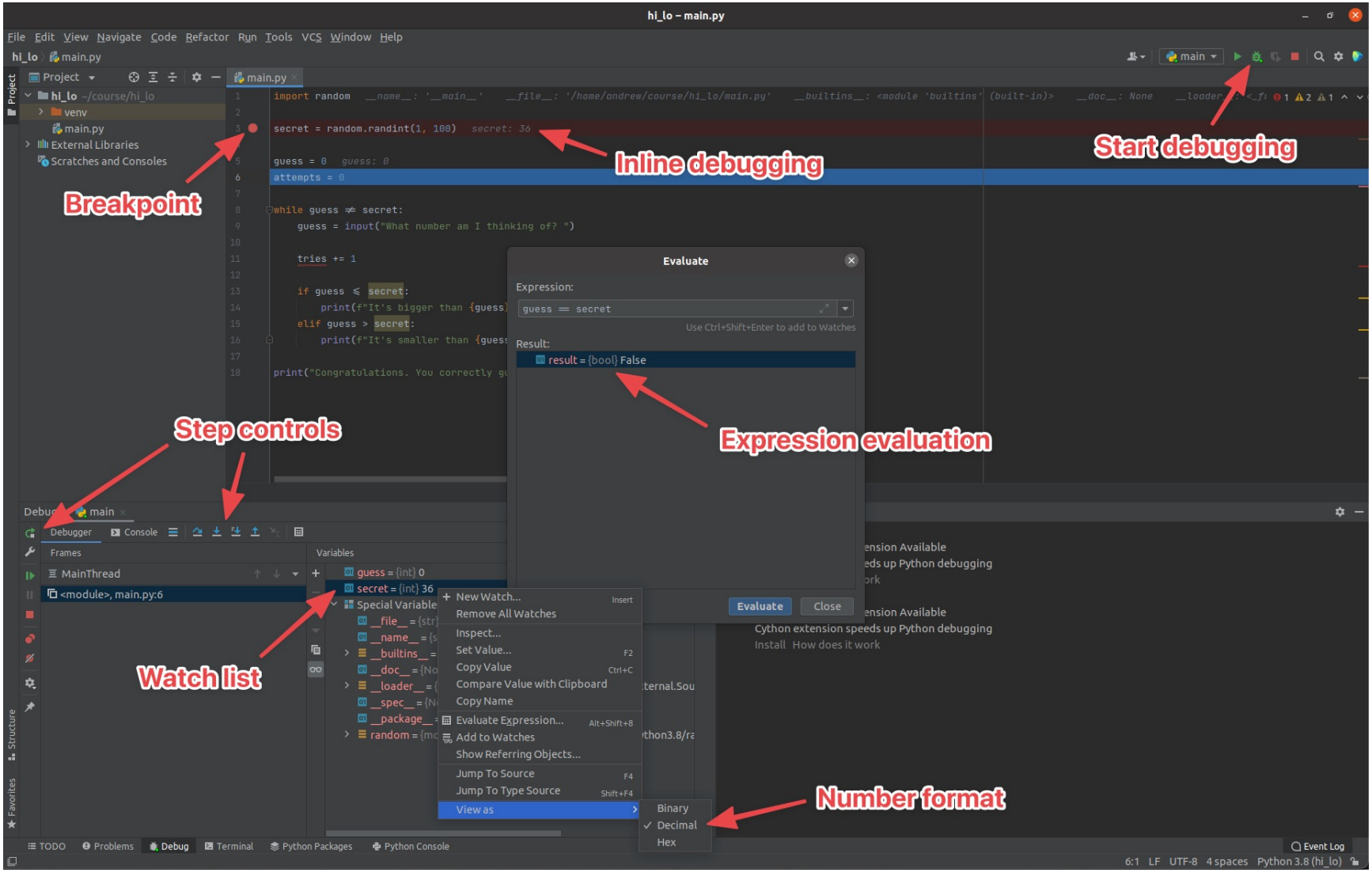
LEARNING TREE™
INTERNATIONAL

Contents

- ► **Debuggers**
- ► **Tracing**
- ► **Logging**
- ► **Static analysis**
- ► **Typing**
- ► **Debugging C++ libraries**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

PyCharm debugging



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

pdb

- **pdb is an interactive command-line debugger for Python**
- **Available in standard Python distribution**
- **Useful when you have no IDE available (e.g. server, IoT)**
- **Can be invoked in four different modes**
  - script
  - run
  - trace
  - post-mortem

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

pdb in script mode

```
# Simple mode
python -m pdb main.py

# Run until an exception occurs
python -m pdb -c continue main.py

# Run until line 10
python -m pdb -c 'until 10' main.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

pdb in run mode

```
# In interpreter
>>> import main
>>> import pdb
>>> pdb.run("main.main()")
(Pdb) b main:8
(Pdb) n
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

pdb in post-mortem mode

```
>>> import main
>>> main.main()
>>> # Code has crashed in interpreter
>>> import pdb
>>> pdb.pm()
```

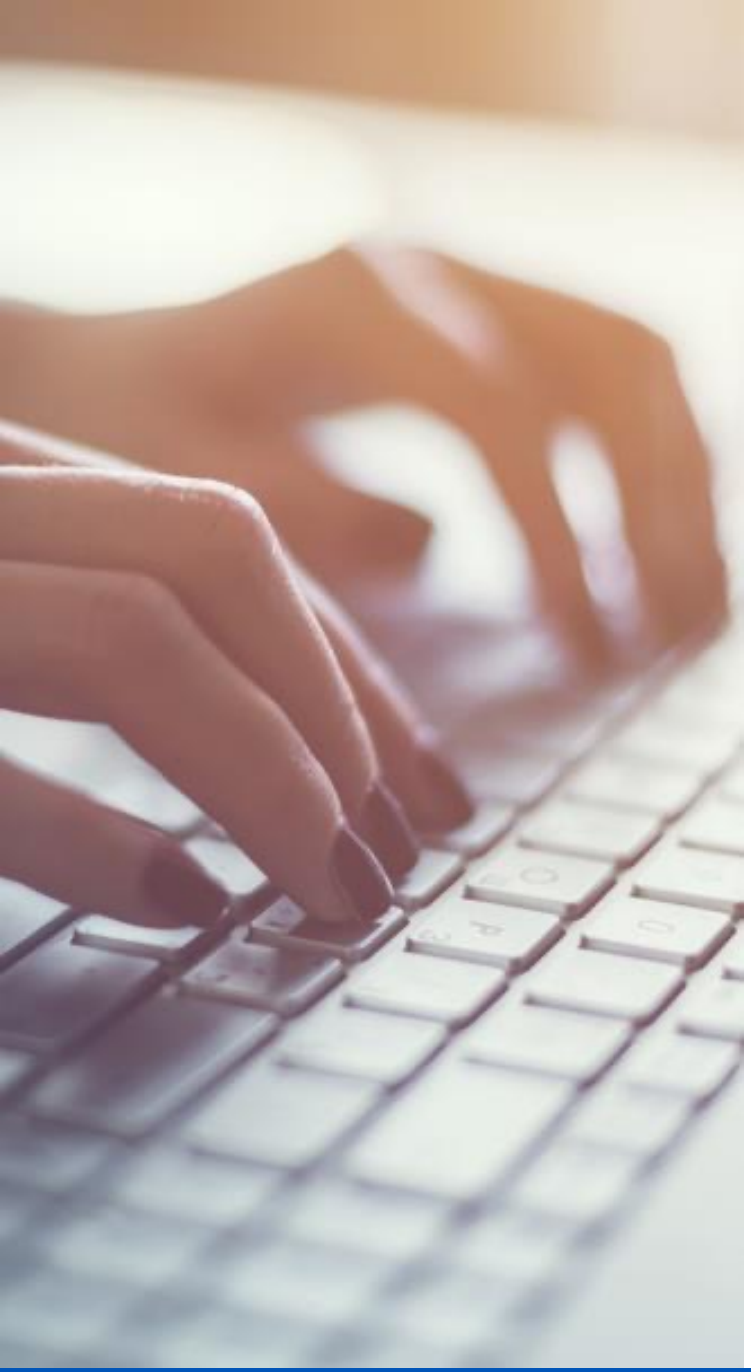
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Common pdb commands

- **`h`**: display help (`h [cmd]` for command-specific help)
- **`b`**: set a breakpoint
- **`c`**: continue debugging until you hit a breakpoint
- **`s`**: step into the code
- **`n`**: go to the next line of code (step over)
- **`l`**: list source code (11 lines by default)
- **`ll`**: longer source code list
- **`u`**: navigate up the call stack
- **`d`**: navigate down the call stack
- **`p`**: print the value of an expression

https://cheatography.com/ralienpp/cheat-sheets/python-pdb/pdf/

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Debug an app using pdb

Try it for yourself

- Use pdb to explore `hi_lo_broken.py`.
- Explore all four modes

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

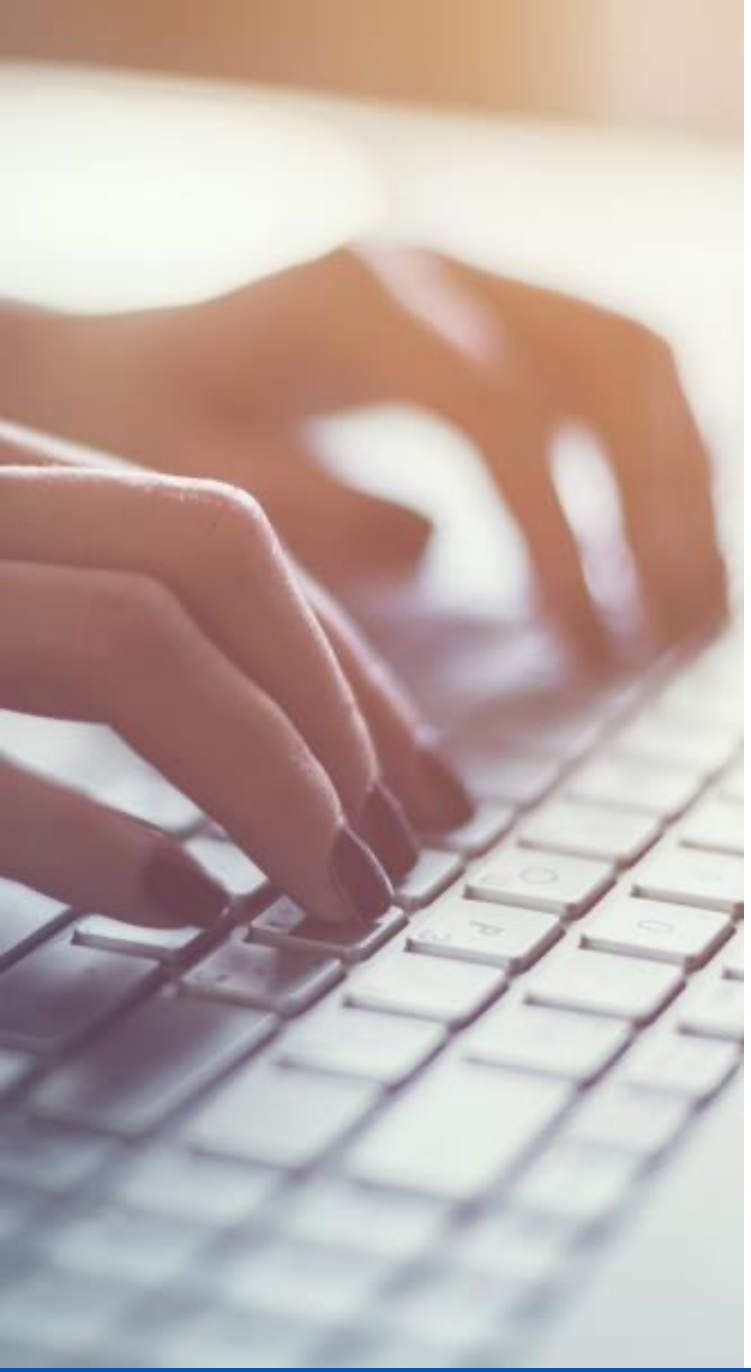
- Debuggers
- **Tracing**
- Logging
- Static analysis
- Typing
- Debugging C++ libraries

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Tracing

```
python -m trace --help

# Print each line as it's executed
python -m trace --trace tracing.py

# Count the number of times each line is executed
# Report is written to tracing.cover
python -m trace --count --coverdir . tracing.py

# Track the functions that are executed
python -m trace --listfuncs tracing.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Trace the execution of an app

Try it for yourself

- **Trace the execution of a Rock, Paper, Scissors, Lizard, Spock game**
  - https://bit.ly/3uIAinD
- **The app is rpsls.py**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Debuggers
- Tracing
- **Logging**
- Static analysis
- Typing
- Debugging C++ libraries

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Logging

```python
import logging

logging.debug("This is a debug message")
logging.info("This is an info message")
logging.warning("This is a warning message")
logging.error("This is a error message")
logging.critical("This is a critical message")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Basic logging configuration

```python
import logging

logging.basicConfig(
    level=logging.DEBUG,
    filename="app.log",
    format="%(asctime)s %(levelname)s %(process)d: %(message)s",
)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Custom loggers

```
import logging

logger = logging.getLogger("appLogger")
logger.warning("This is a warning message")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Logging handlers

```python
import logging

logger = logging.getLogger("appLogger")
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.DEBUG)
console_formatter = logging.Formatter("%(name)s %(levelname)s: %(message)s")
console_handler.setFormatter(console_formatter)
logger.addHandler(console_handler)

logger.warning("This is a warning message")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Runtime configuration files

```python
import logging
import logging.config

logging.config.fileConfig(fname="logging.ini")
logger = logging.getLogger("appLogger")

logger.debug("This is a debug message")
```

```ini
[loggers]
keys=root,appLogger

[logger_root]
level=DEBUG
handlers=consoleHandler

...
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Configuring logging with YAML

```python
import logging
import logging.config
import yaml

with open("logging.yml") as f:
    config = yaml.safe_load(f.read())
    logging.config.dictConfig(config)

logger = logging.getLogger("appLogger")

logger.debug("This is a debug message")
```

```yaml
version: 1
root:
  level: DEBUG
  handlers: [console]
loggers:
  appLogger:
    level: DEBUG
    handlers: [console,file]

...
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Logging custom values

```
logger.debug("This is a debug message")
logger.info(42)
logger.warning({
  "author": "Bram Stoker"
})

username = "andrew"
logger.info(f"{username} logged in")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Logging exceptions

```
a = 1
b = 0

try:
    c = a / b
except ZeroDivisionError:
    logger.exception("Attempted division by zero")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Logging uncaught exceptions

```python
import sys

def handle_exception(exc_type, exc_value, exc_traceback):
    # Allow a console program to exit with ^C
    if issubclass(exc_type, KeyboardInterrupt):
        sys.__excepthook__(exc_type, exc_value, exc_traceback)

        return

    logger.critical(
        "Uncaught exception",
        exc_info=(exc_type, exc_value, exc_traceback)
    )

sys.excepthook = handle_exception
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Rolling log files

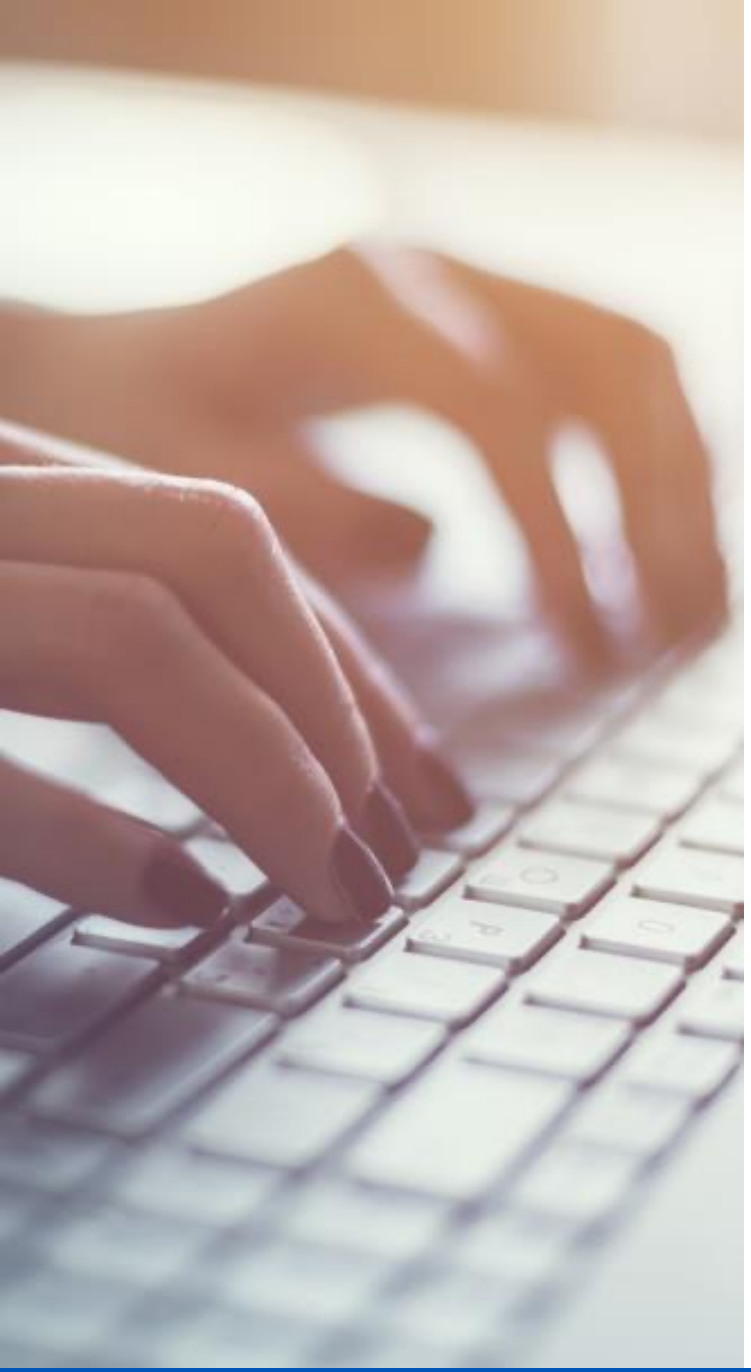
```python
import logging
import logging.handlers as handlers
import time

fileHandler = handlers.TimedRotatingFileHandler(
    "app.log", when="M", interval=1
)
fileHandler.setLevel(logging.DEBUG)
logger.addHandler(fileHandler)

while True:
    time.sleep(1)
    logger.debug("This is a debug message")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

External logging handlers

- google-cloud-logging
- loggly-python-handler
- opencensus-ext-azure
- splunk_handler
- watchtower (AWS Cloudwatch)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Add logging to an app

Try it for yourself

- **Add file logging to hi_lo_broken.py**
  - Use a custom logger
  - Think about the types of things you'd want to know about this app if users reported that it wasn't working
- **Log an exception**
- **Add an unhandled exception handler**
- **Move the configuration out to an external file so that logging can be configured without modifying the app**
- **Solution:**
  - hi_lo_broken_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Debuggers
- Tracing
- Logging
- **Static analysis**
- Typing
- Debugging C++ libraries

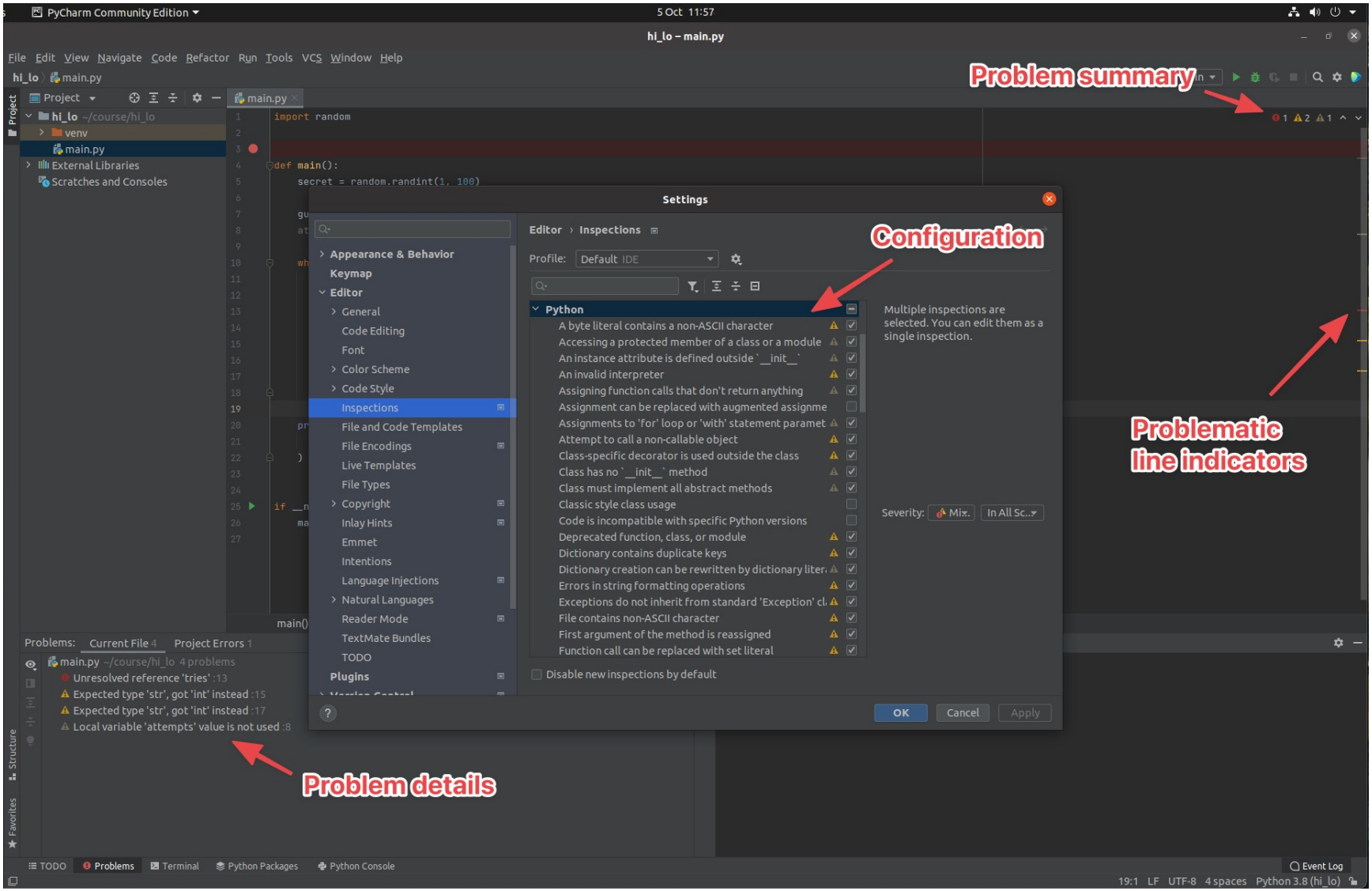
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Static analysis

- **Static analysis checks your code without running it**
- **Linting is static analysis**
- **Advances in computer processing power allowing linted to do done in real time**
- **Static analysis include linting, dependency analysis, coding metrics, etc**
- **Can also be considered a matter of degree**
  - Some types of analysis would be very distracting if included as part of the coding cycle
  - Developers have different levels of tolerance for automated suggestions
  - Need to balance productivity with "purity"

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Static analysis in PyCharm



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Prospector

- **Provides information about**
  - errors
  - potential problems
  - convention violations
  - complexity
- **Brings together a range of static analysis tools**
  - Dodgy
  - McCabe
  - Pep8
  - Pydocstyle/Pep257
  - Pyflakes
  - Pylint
- **Supports others, such as Bandit, as optional extras**
- **Uses sensible defaults to make it easy to get started**
  - Highly configurable

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Static analysis using Prospector

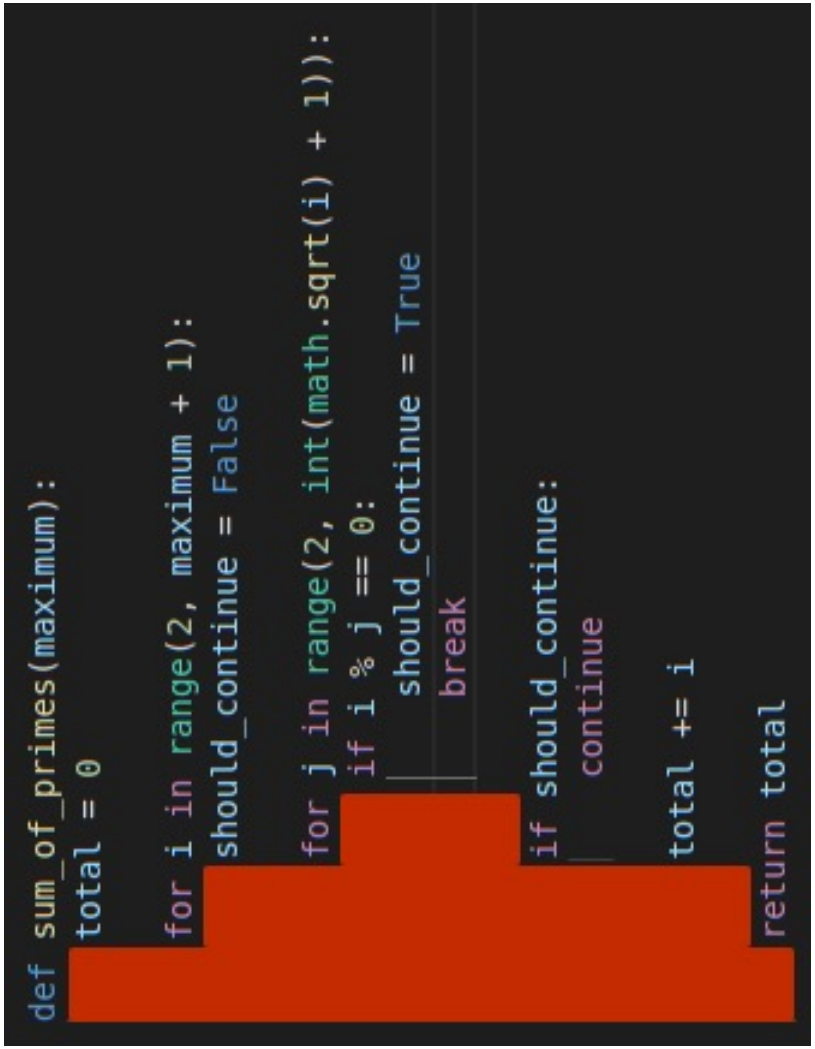
```
pip install prospector

# Decide what to check
prospector

# Check hi_lo_broken.py
prospector --strictness veryhigh hi_lo_broken.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cyclomatic complexity

$$C = E - N + 2$$

E is the number of edges (relationships)
N is the number of nodes (instructions)
C is the cyclomatic complexity



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Calculating cyclomatic complexity using McCabe

```
python -m mccabe --min 2 sum_of_primes.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Conducting a security audit with Bandit

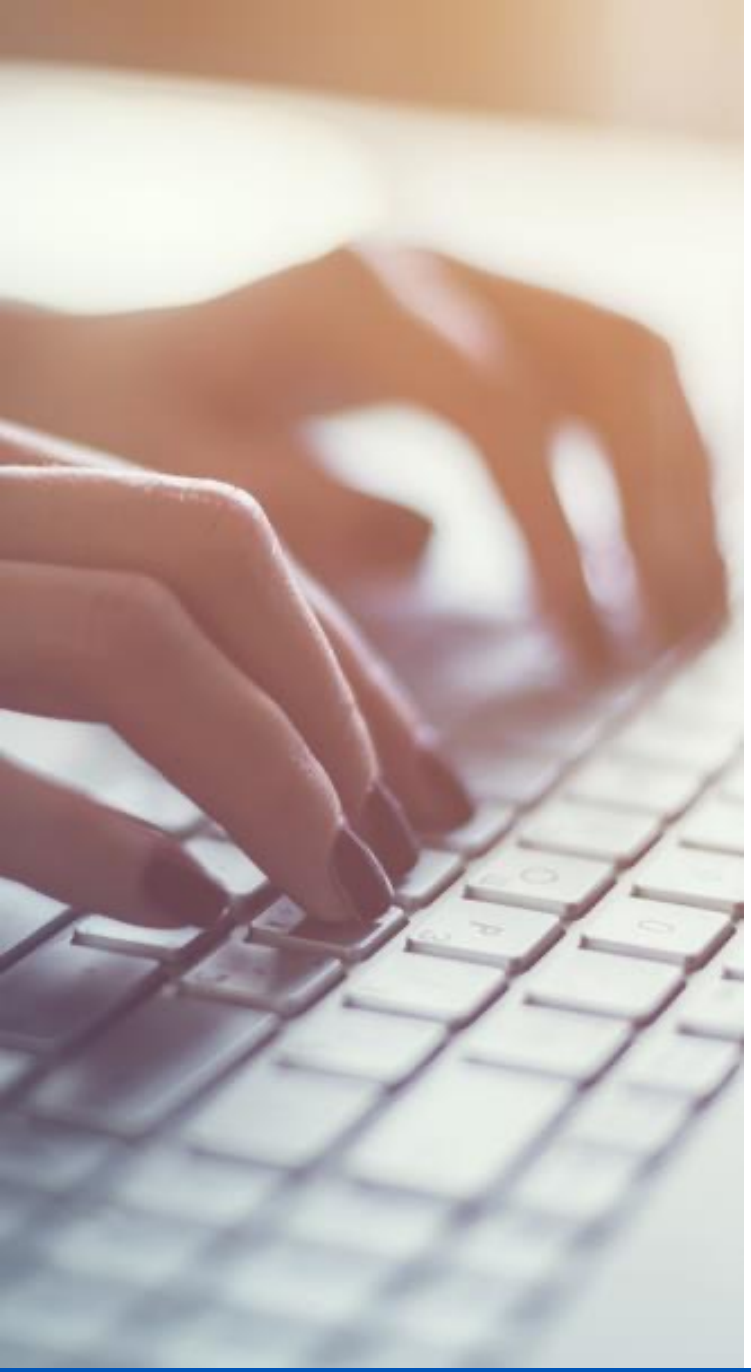
```
pip install prospector[with_bandit]
prospector --with-tool bandit unsafe_yaml_load.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Prospector profiles

```yaml
output-format: json

strictness: medium
test-warnings: true

ignore-paths:
  - docs

ignore-patterns:
  - (^|/)skip(this)?(/|$)

pep8:
  disable:
    - W602
  enable:
    - W601

mccabe:
  run: false
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Refactor to reduce complexity

Try it for yourself

- **Rector the code in sum_of_primes.py to have a lower complexity score**
- **Solution:**
  - sum_of_primes_simple.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Debuggers
- Tracing
- Logging
- Static analysis
- **Typing**
- Debugging C++ libraries

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Typing

- **Python is dynamically typed**
  - Interpreter checks types as the code runs
  - The type of a variable can change over its lifetime
- **Type hints are supported in Python (since Python 3.5)**
  - Uses structural (duck) typing, rather than nominal typing
- **Type hints have no effect at runtime**
  - A program with inconsistent types will still run
- **Static type checkers are used to verify typed code**
  - Mypy is the most popular
- **Types hints also**
  - Allow IDEs and linters to provide better support
  - Provide formal documentation of intended behaviour
  - Enhance autogenerated documentation (e.g. OpenAPI)
  - Improve design by encouraging developers to be consistent
  - Deliver more accurate refactoring

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Should I use type hints?

- **Adds another level of complexity for inexperienced developers**
  - Most resources show untyped code
  - Learning curve can be frustrating
- **Gradual typing is supported**
  - Untyped code is ignored by the type checker
- **Decreases readability**
- **Unnecessary in short, throw-away scripts**
- **Beneficial in large codebases**
- **Helps keep code maintainable over time**
- **Very beneficial for code that will be used by third parties**
- **If it's worth writing unit tests, it may be worth using type hints**
  - Both help the developer write better code
- **There's a *small* hit in start-up time**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Annotations (functions)

```
def func(arg: arg_type, kwarg: kwarg_type=default) -> return_type:
    pass
```

```
def to_fahrenheit(deg_c: float) -> float:
    return deg_c * 9 / 5 + 32
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Variable annotations

```
freezing_point_deg_f: float = 32.
feeling: str = "Brrrr..."
boiling_point_deg_f: float = 212.

# Variable doesn't exist---only type info
absolute_zero_deg_c: float
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Mypy

```python
def to_fahrenheit(deg_c: float) -> float:
    return deg_c * 9.0 / 5.0 + 32.0


deg_c = input("What temperature do you wish to convert? ")
deg_f = to_fahrenheit(deg_c)

print(f"{deg_c} degC is {deg_f} degF.")
```

```
mypy temperature_broken.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Revealing Mypy types

```python
def to_fahrenheit(deg_c: float) -> float:
    return deg_c * 9.0 / 5.0 + 32.0


deg_c = input("What temperature do you wish to convert? ")
deg_f = to_fahrenheit(deg_c)

print(f"{deg_c} degC is {deg_f} degF.")

# These are specific to Mypy and will fail in the interpreter
reveal_type(to_fahrenheit)
reveal_locals()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Complex types

```python
from typing import Dict, List, Tuple

riders: List[str] = ["Rossi", "Marquez", "Quartararo"]

bike_numbers: List[Tuple[str, int]] = [
  ("Rossi", 46), ("Marquez", 93), ("Quartararo", 20)
]

BikeNumber = Tuple[str, int]

bike_numbers_2: List[BikeNumber] = [
  ("Rossi", 46), ("Marquez", 93), ("Quartararo", 20)
]

bike_numbers_3: Dict[str, int] = {
  "Rossi": 46,
  "Marquez": 93,
  "Quartararo": 20
}
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Optional types

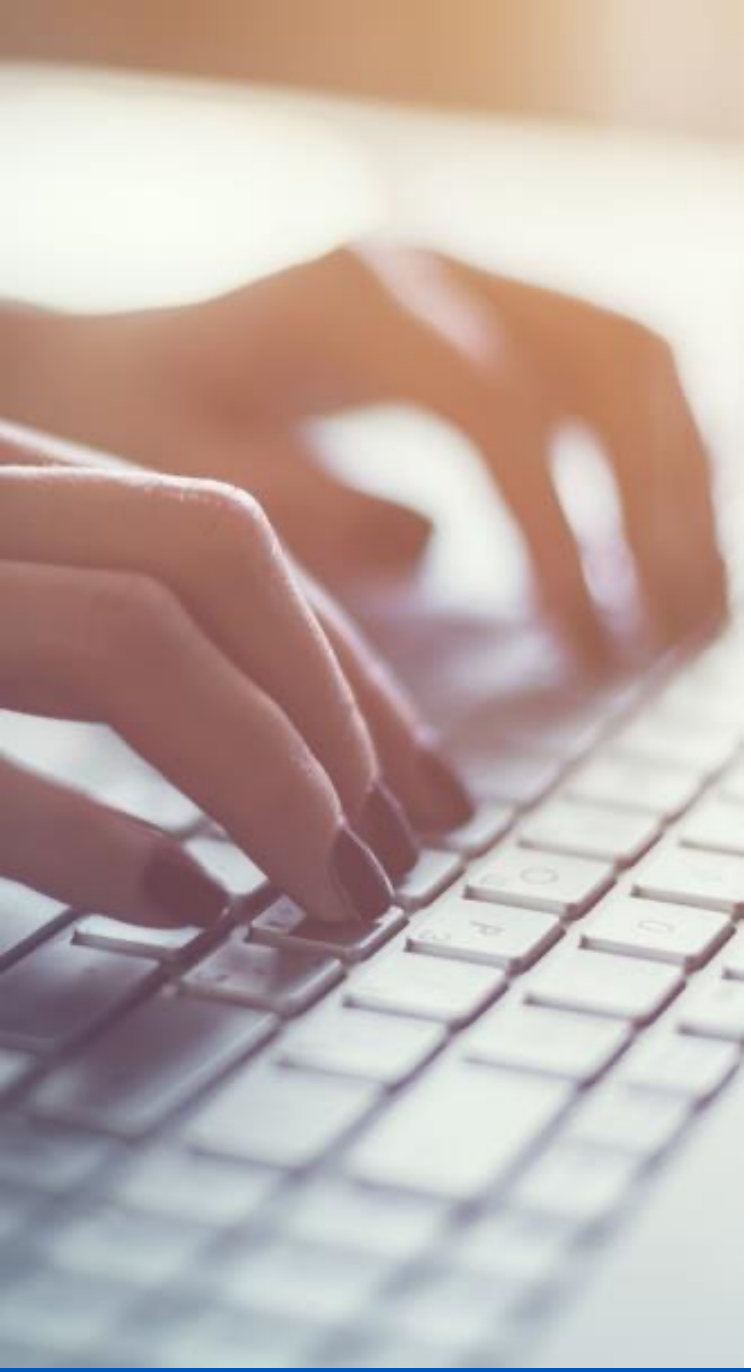
```python
from typing import Optional


def print_address(
    address1: str, address2: Optional[str], town: str, postcode: str
) -> None:
    print(address1)

    if address2:
        print(address2)

    print(town)
    print(postcode)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Functions that never return

```
from typing import NoReturn

def raise_div_by_zero_exception() -> NoReturn:
    raise ZeroDivisionError("How did this happen?!")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a typed function library

Try it for yourself

- **Create some library functions that allow you to work with a deck of cards**
- **You should include functions that**
  - Create a deck of 52 cards
  - Shuffle a deck
  - Deal *N* cards from a deck
- **Make sure your functions are fully typed**
- **Write code to test your functions**
  - Again, make sure it's typed
- **Use Mypy to check your types**
- **Solution:**
  - cards_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Debuggers
- Tracing
- Logging
- Static analysis
- Typing
- **Debugging C++ libraries**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cross-debugging C/C++ Python extensions

- **Debugging mixed Python and C/C++ code isn't supported in PyCharm (or CLion)**
- **Workaround for Python extension modules in *CLion* involves attaching the native debugger to the Python process**
  - Set breakpoints in CLion and make sure they'll be hit *after* attachment
  - Run the Python script
  - Open the **Run | Attach to Process** menu in CLion
  - Select the ID of the Python process from the **Native** section of the list
  - Debug the native code as usual
  - Open the **Run | Stop** menu to detach from the process when finished

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **pdb can be used to debug apps when other options aren't available**
- **Tracing can be used to understand the dynamic behaviour of an app**
- **Built-in logging is powerful and flexible**
- **PyCharm's code inspections cover most cases**
  - Use Prospector for more detailed analysis
- **Type hints can improve the maintainability of a codebase**
  - At a cost (e.g. readability)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Testing

CULTURAL PROBE
DEVELOP PERSONAS
CARD SORTING
CUSTOMER INTERVIEWS
LISTEN IN ON CUSTOMER SERVICE CALLS
FIELD VISITS
RUN A USABILITY TEST
USER SURVEY

Contents

- ► Unit testing
- ► Code coverage
- ► Mocking

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Unit testing in Python

- **There are a number of test runners available for Python**
- **unittest**
  - Batteries included
  - Framework and runner
  - Requires tests to be class methods
  - Uses custom assertion methods
- **nose**
  - Compatible with unittest
  - Extends unittest and makes it easier to work with
  - Can use standard assertions
- **pytest**
  - The de facto standard
  - Compact, but scalable
  - Tests are functions, not classes
  - Uses standard assertions
  - Mature plug-in ecosystem

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Unit testing strategy

Arrange

↓

Act

↓

Assert

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Installing pytest

```
pip install pytest
pytest --help
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating and running tests

test_basic.py

```
def test_always_passes():
    assert True

def test_always_fails():
    assert False
```

```
pytest
```

Convention over configuration

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

pytest output

Root folder

System info

```
(venv) andrew@styria:~/course/testing$ pytest
============================= test session starts ==============================
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: /home/andrew/course/testing
collected 2 items

test_basic.py .F                                                         [100%]

================================= FAILURES =================================
_____________________________ test_always_fails ______________________________

    def test_always_fails():
>       assert False
E       assert False

test_basic.py:6: AssertionError
=========================== short test summary info ============================
FAILED test_basic.py::test_always_fails - assert False
========================== 1 failed, 1 passed in 0.01s ==========================
(venv) andrew@styria:~/course/testing$
```

Status

Failures

. = Pass, F = Fail, E = Unexpected exception

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Simple tests

```python
import math


def test_strip():
    assert "  I'm padded  ".strip() == "I'm padded"


def test_sort():
    assert sorted([5, 1, 3, 2, 4]) == list(range(1, 6))


def test_factorial():
    assert math.factorial(6) == 720


def test_normalise_text():
    # Not a best practice
    assert "  I'm padded  ".strip().upper() == "I'M PADDED"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Arrange, Act, Assert

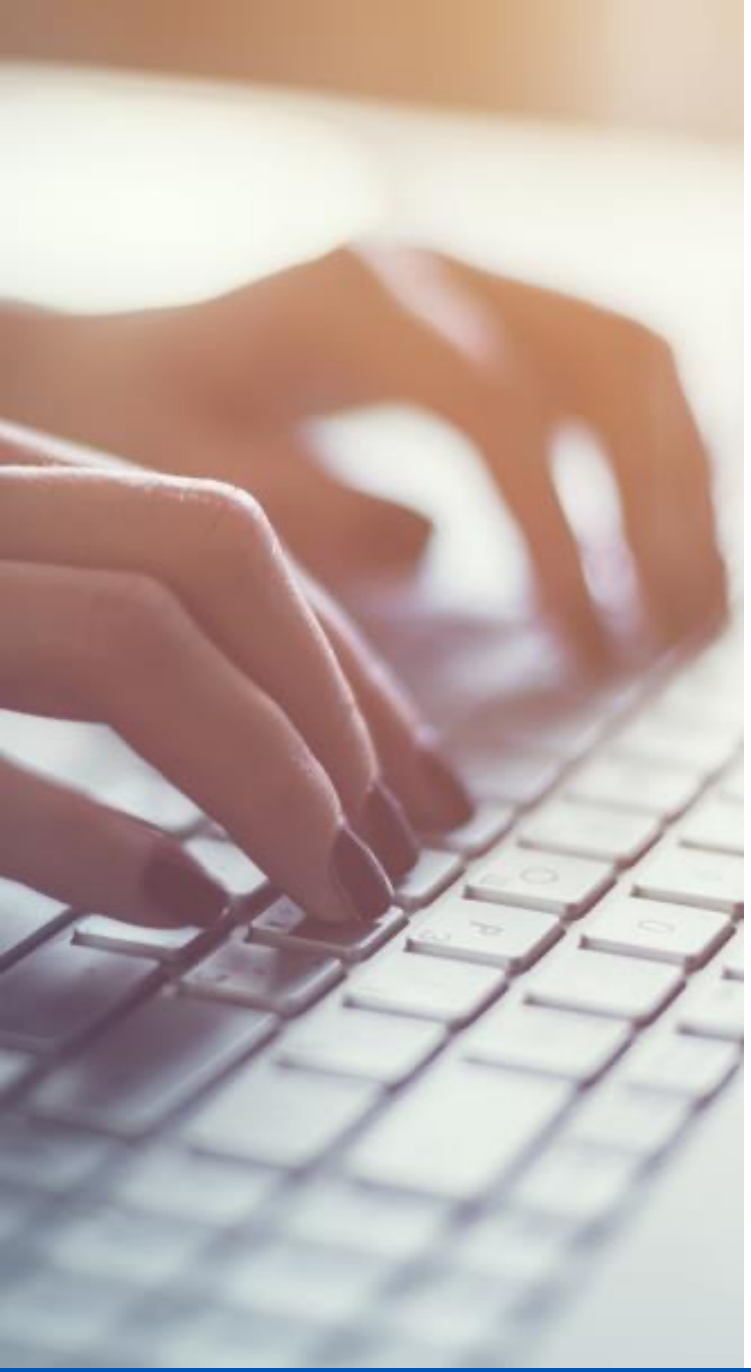
```python
def test_sort():
    # Arrange
    expected = list(range(0, 6))

    # Act (fail---missing zero)
    actual = sorted([5, 1, 3, 2, 4])

    # Assert
    assert actual == expected
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Output from failed test

```
(venv) andrew@styria:~/course/testing$ pytest test_sort_fail.py
================================= test session starts =================================
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: /home/andrew/course/testing
collected 1 item

test_sort_fail.py F                                                          [100%]

====================================== FAILURES =======================================
_________________________________ test_sort __________________________________

    def test_sort():
        # Arrange
        expected = list(range(0, 6))

        # Act (fail---missing zero)
        actual = sorted([5, 1, 3, 2, 4])

        # Assert
>       assert actual == expected
E       assert [1, 2, 3, 4, 5] == [0, 1, 2, 3, 4, 5]
E         At index 0 diff: 1 != 0
E         Right contains one more item: 5
E         Use -v to get the full diff

test_sort_fail.py:9: AssertionError
============================== short test summary info ================================
FAILED test_sort_fail.py::test_sort - assert [1, 2, 3, 4, 5] == [0, 1, 2, 3, 4, 5]
================================= 1 failed in 0.01s ===================================
(venv) andrew@styria:~/course/testing$
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Add tests to an app

Try it for yourself

- **Add tests to the cards library**
- **Solution:**
  - test_cards_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Managing state in tests

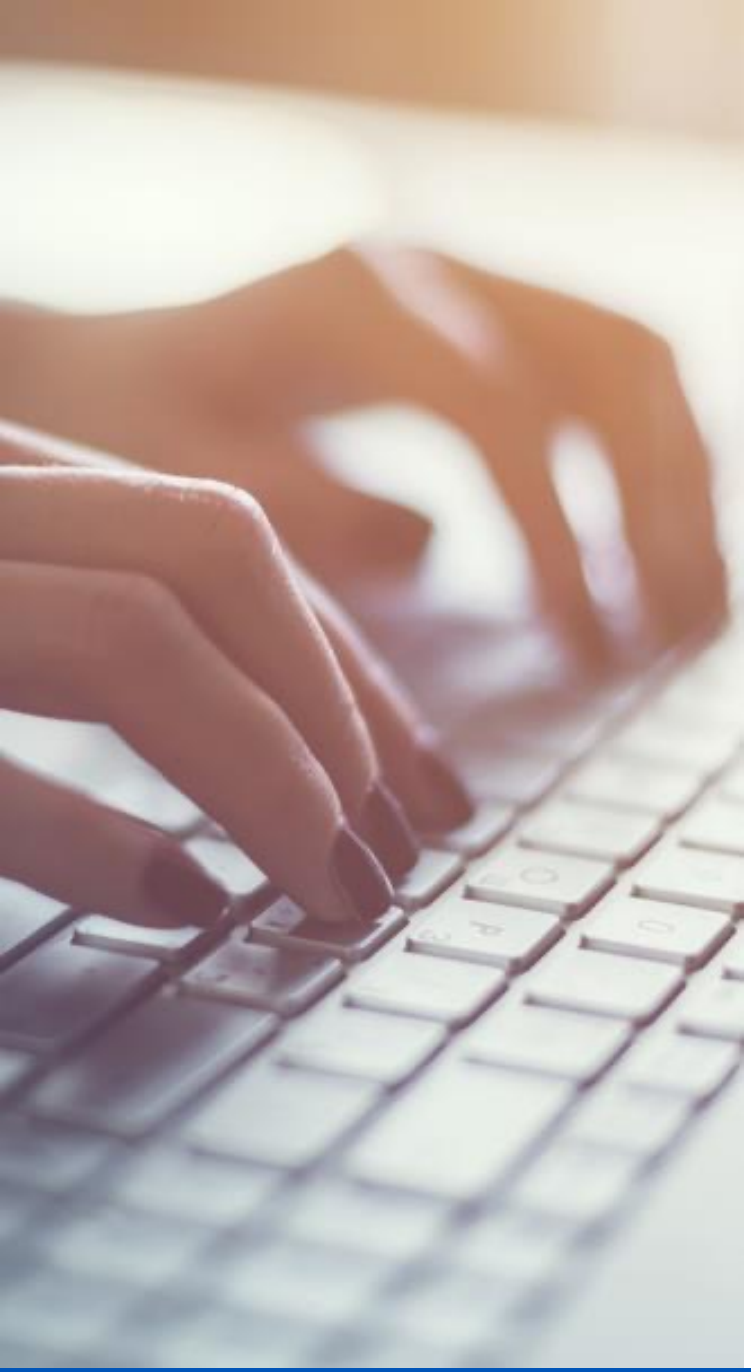
- **Tests should be isolated**
  - They should be able to run in any order
- **Tests often require state to operate**
- **State can lead to a lot of redundant code and implicit dependencies**
- **Often managed in "setup" and "tear down" methods (e.g. unittest)**
- **This can obfuscate the relationship between state and individual tests**
  - Implicit state leads to tests that fail unexpectedly
- **pytest allows state to be formally managed via fixtures**
- **The lightweight philosophy of pytest also makes it easy to management state using standard functions**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Test fixtures

```python
import json
import pytest

def get_captial(capitals, country):
    return capitals[country]

@pytest.fixture
def capitals_fixture():
    with open("capitals.json") as f:
        return {
            capital["country"]: capital["capital"]
            for capital in json.load(f)
        }

def test_get_captial_returns_paris_for_france(capitals_fixture):
    capital = get_captial(capitals_fixture, "France")
    assert capital == "Paris"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Improve tests using fixtures

Try it for yourself

- **In testing our card library, we had to make regular calls to `create_deck`**
  - This gets a little repetitive
- **Add a fixture to inject a new deck into the tests that require one**
- **Solution:**
  - text_cards_fixture_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Alternative to test fixtures

```
import json

def get_captial(capitals, country):
    return capitals[country]

def create_capitals():
    with open("capitals.json") as f:
        return {
          capital["country"]: capital["capital"]
          for capital in json.load(f)
        }

def test_get_captial_returns_paris_for_france(capitals=create_capitals()):
    capital = get_captial(capitals, "France")
    assert capital == "Paris"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Fixtures are modular

```
@pytest.fixture
def large_country_capitals_fixture(capitals_fixture):
    return {
        key: value
        for key, value in capitals_fixture.items()
        if key in ["Russia", "Canada", "United States", "China", "Brazil"]
    }
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Testing exceptions

```python
import pytest

def test_dividing_by_zero_raises_zero_division_error():
    with pytest.raises(ZeroDivisionError) as exc_info:
        result = 1 / 0

    # Optional
    assert str(exc_info.value) == "division by zero"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Skipping tests

```python
import pytest
import time

def long_running_task():
    time.sleep(3)

@pytest.mark.skip(reason="too slow")
def test_long_running_task():
    long_running_task()

    assert True

# "pytest -rs" to show skipped tests
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Skipping all tests in a module

```python
import pytest
import sys

# Place at start of test module
if not sys.platform.startswith("win"):
    pytest.skip("skipping windows-only tests", allow_module_level=True)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Expected to fail tests

```python
import pytest

def is_prime(n):
    # To be done
    return False

@pytest.mark.xfail
def test_is_prime_returns_true_for_2():
    assert is_prime(2)

# "pytest -rxX" to show XFAIL/XPASS tests
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Defining custom marks

pytest.ini

```
[pytest]
markers =
    slow: marks tests as slow (deselect with '-m "not slow"')
    serial: tests cannot be executed in parallel
```

```
import pytest
import time

def long_running_task():
    time.sleep(3)

@pytest.mark.slow
def test_long_running_task():
    long_running_task()

    assert True
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Identifying slow tests

```
pytest --durations=3
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Parameterised tests

```python
import pytest

def c_to_f(deg_c):
    return deg_c * 9 / 5 + 32

@pytest.mark.parametrize(
    "deg_c, expected_deg_f", [(-10, 14), (0.0, 32.0), (100.0, 212.0)]
)
def test_c_to_f_returns_correct_temperature(deg_c, expected_deg_f):
    actual_deg_f = c_to_f(deg_c)
    assert round(actual_deg_f) == expected_deg_f
```

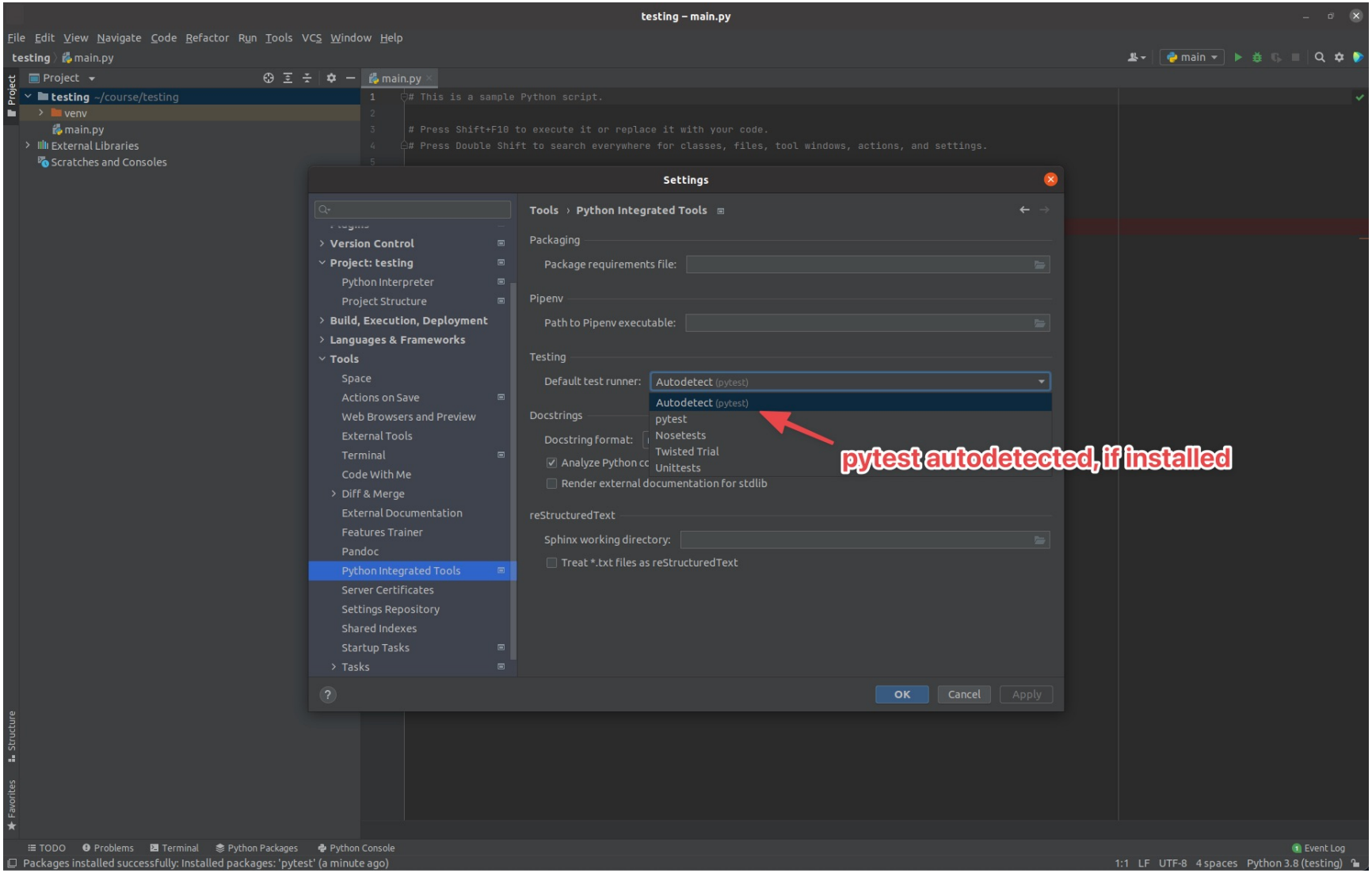
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Useful pytest plug-ins

- **pytest-bdd**
  - Facilitates behavioural driven development
- **pytest-cov**
  - Measures the coverage of your tests
- **pytest-instafail**
  - Show failures immediately, instead of waiting until all tests have run
- **pytest-picked**
  - Only tests files that have changed since your last commit
- **pytest-randomly**
  - Helps ensure independent tests by running them in random order
- **pytest-sugar**
  - Prettier output
- **pytest-xdist**
  - Runs tests in parallel

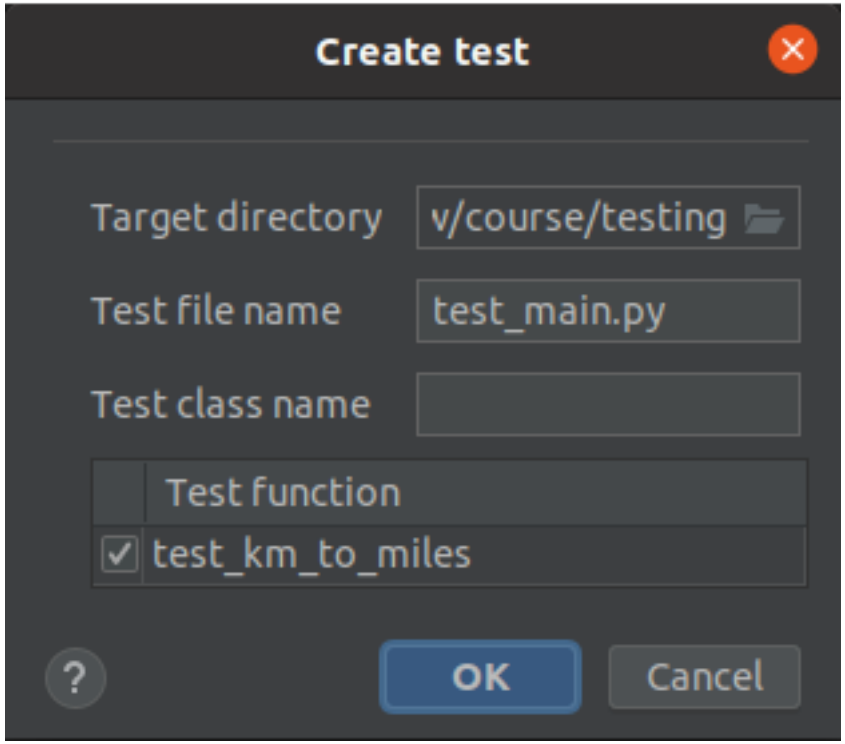
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

pytest in PyCharm



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Create tests in PyCharm



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Running tests in PyCharm

testing – test_main.py

File Edit View Navigate Code Refactor Run Tools VCS Window Help

testing 〉 test_main.py

Python tests in testing

Project

testing

.pyt

venv

mai

test

Extern

Scratcl

New
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+Shift+7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Inspect Code...
Refactor
Clean Python Compiled Files
Add to Favorites
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Override File Type
Run 'Python tests in test...' Ctrl+Shift+F10
Debug 'Python tests in test...'
Modify Run Configuration...
Open In
Local History
Reload from Disk
Compare With... Ctrl+D
Mark Directory as

main.py | test_main.py

```
import main
import pytest


@pytest.mark.parametrize(
    "deg_c, expected_deg_f", [(-10, 14), (0.0, 32.0), (100.0, 212.0)]
)
def test_celsius_to_fahrenheit(deg_c, expected_deg_f):
    actual_deg_f = main.celsius_to_fahrenheit(deg_c)
    assert round(actual_deg_f) == expected_deg_f


def test_km_to_miles():
    assert round(main.km_to_miles(5), 2) == 3.11
```

Run individual test

Run all tests in selected folder

test_km_to_miles()

Run: Python tests in testing

Tests passed: 4 of 4 tests – 2 ms

Test Results 2 ms

```
/home/andrew/course/testing/venv/bin/python /snap/pycharm-community/252/plugins/python-ce/helpers/pycharm/_jb_pytest_runner.py --path /home/andrew/course/testing
Testing started at 12:16 ...
Launching pytest with arguments /home/andrew/course/testing --no-header --no-summary -q in /home/andrew/course/testing

============================= test session starts ==============================
collecting ... collected 4 items

test_main.py::test_celsius_to_fahrenheit[-10-14] PASSED                  [ 25%]
test_main.py::test_celsius_to_fahrenheit[0.0-32.0] PASSED                [ 50%]
test_main.py::test_celsius_to_fahrenheit[100.0-212.0] PASSED             [ 75%]
test_main.py::test_km_to_miles PASSED                                    [100%]

============================== 4 passed in 0.01s ===============================
```

Auto-run tests

Test report

Run TODO Problems Terminal Python Packages Python Console Event Log

Tests passed: 4 (a minute ago)

14:41 LF UTF-8 4 spaces Python 3.8 (testing)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Unit testing best practices

- **KISS**
- **Test one thing per test**
- **Give your tests descriptive names**
  - Adopt a naming convention
- **Isolate tests**
- **Don't place implementation logic in tests**
- **Ensure tests are deterministic**
- **Don't have external dependencies**
- **Make them part of the build process**
- **Minimise the number of slow tests**
- **Always be testing**
  - At least test prior to commits

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Test layouts

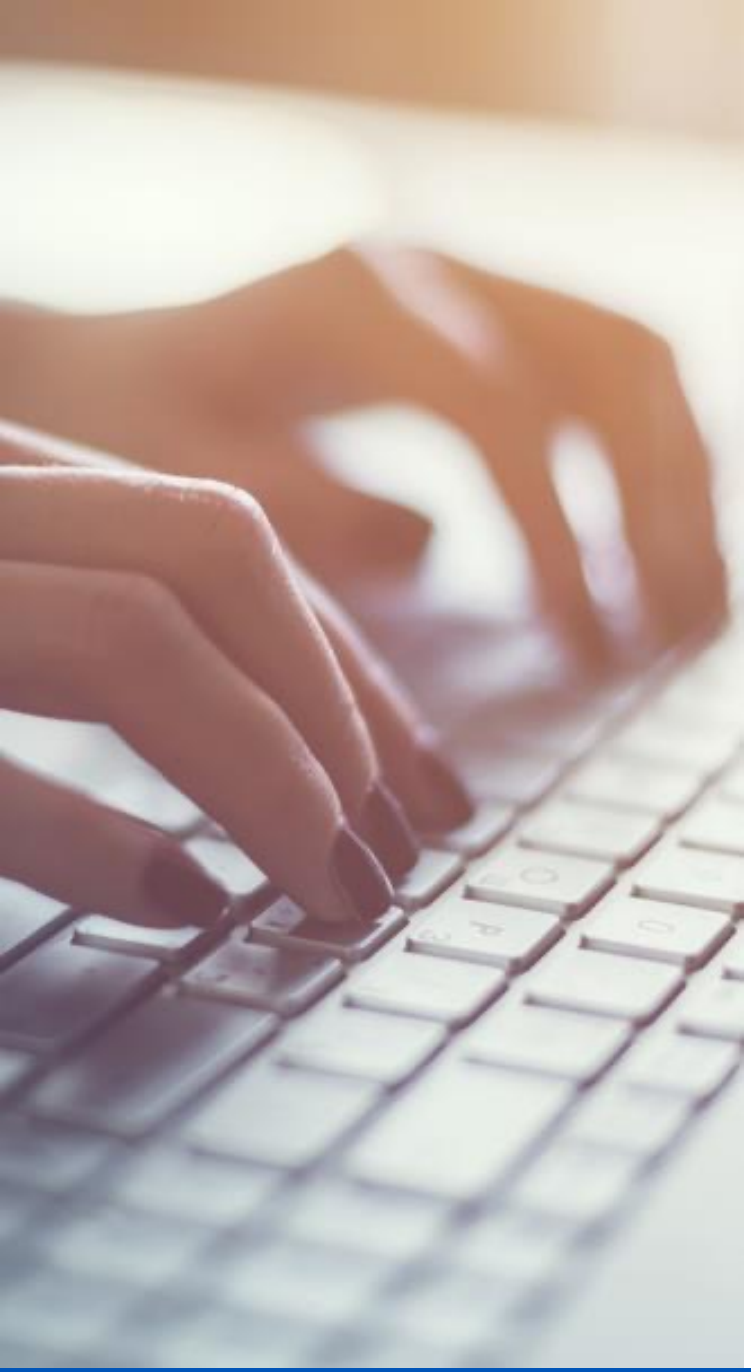
```
project/
    app/
        main.py
        …
    tests/
        test_main.py
        …
    …
```

- Tests are separate from app
- May need to set PYTHONPATH to find modules
- Allows testing structure to be different from app structure

```
project/
    app/
        main.py
        tests/
            test_main.py
            …
        …
    …
```

- Tests are embedded within app
- Easier to import modules
- Tests tend to get distributed with application

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Unit test bank account library

Try it for yourself

- Test the functions in bank_account.py
- Use fixtures to inject accounts
- Test deposits and withdrawals with parameterised tests
- Test that a `BankAccountError` is thrown when withdrawing more than the balance
- Add a test to check that special offers are returned by fetch_special_offers
- Time the tests
- Add a mark to skip any lengthy tests
- Solution:
  - test_bank_account_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Unit testing
- **Code coverage**
- Mocking

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Code coverage

- **Having all tests pass is clearly an important goals**
  - However, it's a necessary, but no sufficient check on code quality
- **How do we know that all the code that needs to be tested, is being tested?**
- **Code coverage lets us measure how much of our code is under test**
- **Some development teams like to have 100% of their code covered by tests**
  - It's important to consider the requirements of your own project
  - Chasing 100% code coverage just for the sake of it can be wasteful
- **Coverage.py is popular tool for measuring code coverage in Python apps**
  - Can be used via the pytest-cov plug-in, which adds additional features

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Measuring code coverage with Coverage.py

Run coverage analysis

```
(venv) andrew@styria:~/course/testing$ coverage run -m pytest test_bank_account_solution.py
========================================= test session starts =========================================
platform linux -- Python 3.8.10, pytest-6.2.5, py-1.10.0, pluggy-1.0.0
rootdir: /home/andrew/course/testing
collected 9 items

test_bank_account_solution.py ........s                                                      [100%]

================================= 8 passed, 1 skipped in 0.02s ================================
(venv) andrew@styria:~/course/testing$ coverage report --show-missing
Name                                Stmts   Miss  Cover   Missing
-----------------------------------------------------------------
bank_account.py                        32     10    69%   41-43, 47-53, 57
test_bank_account_solution.py          28      5    82%   52-60
-----------------------------------------------------------------
TOTAL                                  60     15    75%
(venv) andrew@styria:~/course/testing$
```

Generate report

Missing line #

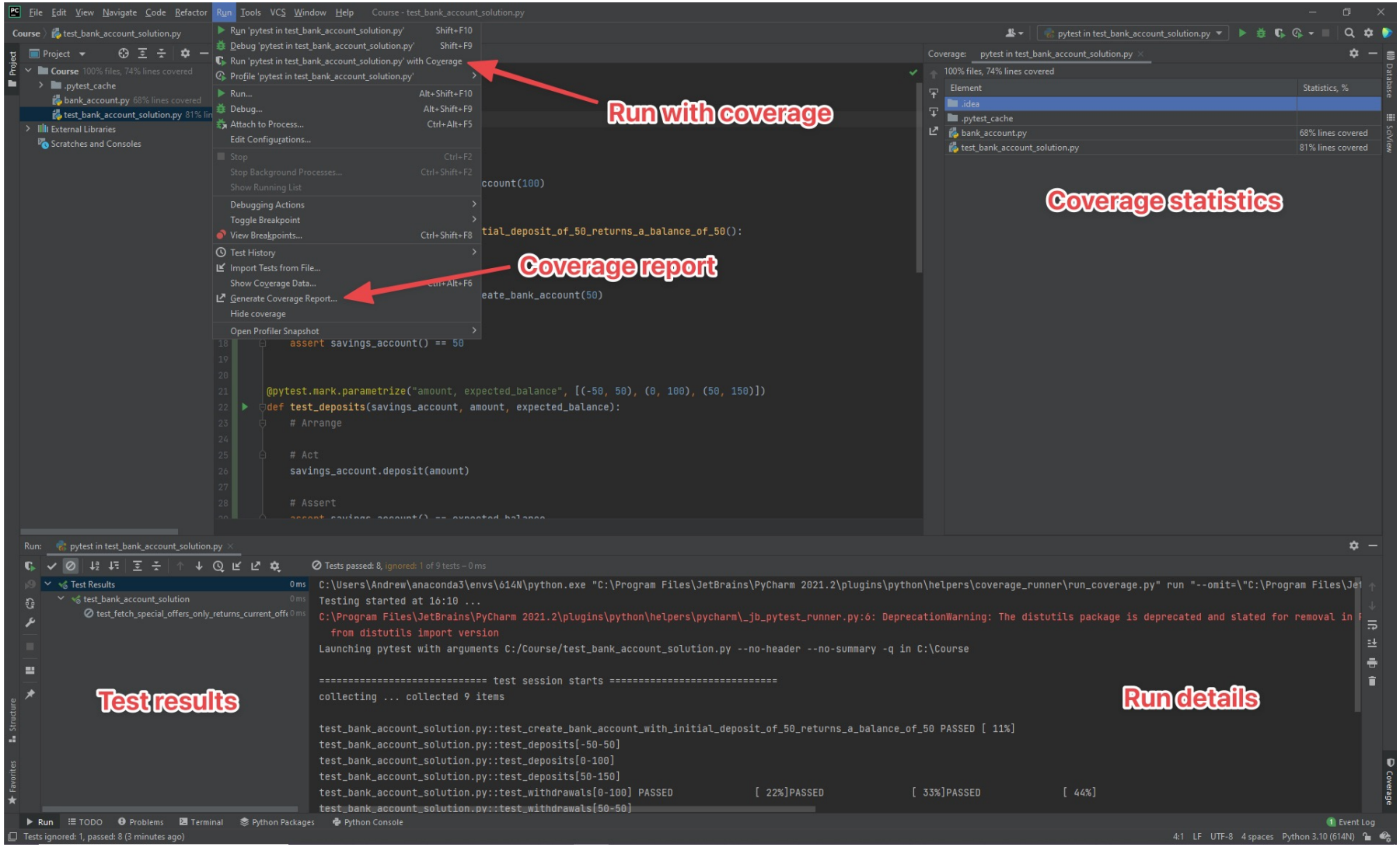
Executable statements

Missed statements

((E - M) / E) * 100

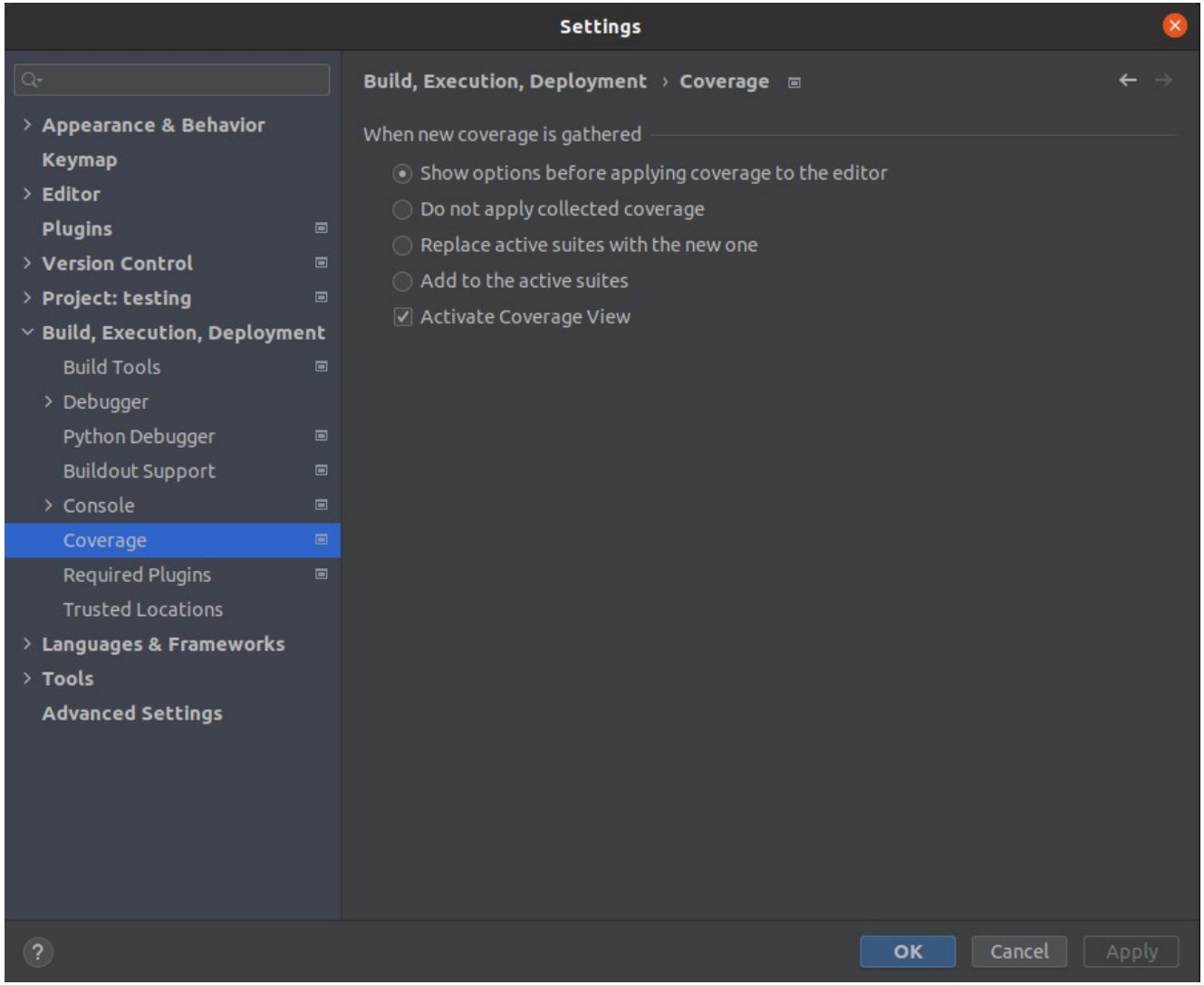
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Coverage with PyCharm (Professional)

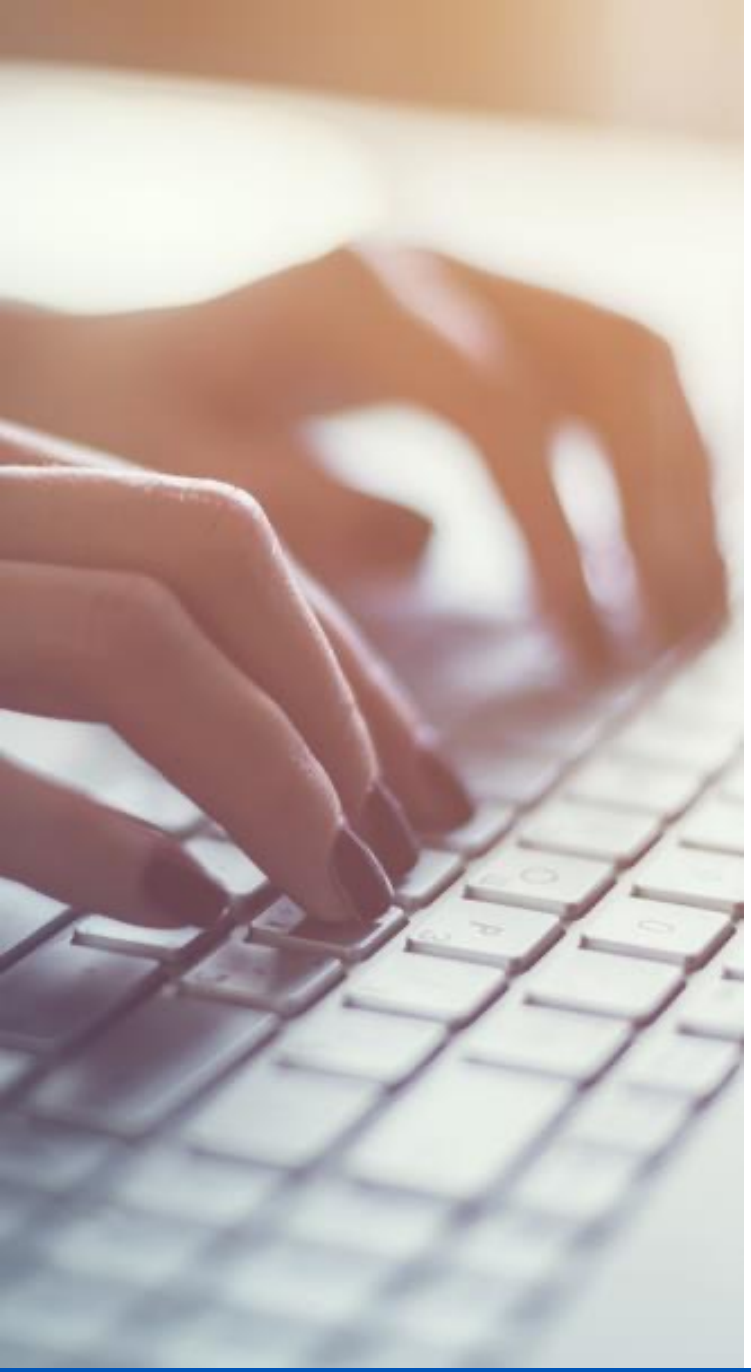


© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Coverage settings in PyCharm



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Achieve 100% test coverage

Try it for yourself

- **Modify bank_account.py and/or the tests developed earlier to achieve 100% test coverage**
- **Was it worth the effort?**
- **Solution:**
  - test_bank_account_coverage_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Unit testing
- Code coverage
- **Mocking**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Mocking

- **Mocking is used to break dependencies in code**
  - Behaviours can then be tested in isolation
- **Mocking replaces implementation code with "mocked" code during testing**
  - Slow operations can be replaced with faster mocks
  - Stochastic data can be replaced with deterministic mocks
- **Mocks can be used to**
  - Whether functions are called
  - How often functions are called
  - What parameters are used in the calls
- **Monkey patching makes mocking relatively easy in Python**
- **Python has a "batteries included" mocking library (unittest.mock)**
  - Third-party mocking libraries (e.g. pytest-mock) build on this base
- **Code that is written to support unit testing can inject mocks from outside**
  - Monkey patching must be used when code is tightly coupled
  - Legacy code often exhibits tight coupling

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Tightly coupled code

```python
import time

def fetch_orders():
    time.sleep(3)

    return [
        {"customer": "ACME", "value": 100.0},
        {"customer": "MegaCorp", "value": 275.0},
    ]

def get_revenue():
    orders = fetch_orders()

    return sum((order["value"] for order in orders))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Mocking using explicit monkey patching

```python
import orders

def test_get_revenue():
    # Arrange
    orders.fetch_orders = lambda: [
        {"customer": "ACME", "value": 100.0},
        {"customer": "MegaCorp", "value": 275.0},
    ]

    # Act
    revenue = orders.get_revenue()

    # Assert

    assert revenue == 375.0
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Decoupling code using dependency injection

```python
def get_revenue(fetch_orders):
    orders = fetch_orders()

    return sum((order["value"] for order in orders))

def test_fetch_orders():
    fetch_orders_mock = lambda: [
        {"customer": "ACME", "value": 100.0},
        {"customer": "MegaCorp", "value": 275.0},
    ]

    revenue = get_revenue(fetch_orders_mock)
    assert revenue == 375.0
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Mocking with pytest-mock

```python
import orders

def test_get_revenue(mocker):
    # Arrange
    mocker.patch(
        "orders.fetch_orders",
        return_value=[
            {"customer": "ACME", "value": 100.0},
            {"customer": "MegaCorp", "value": 275.0},
        ],
    )

    # Act
    revenue = orders.get_revenue()

    # Assert

    assert revenue == 375.0
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Checking arguments passed to mocked functions

```python
import os

def delete_database(file_path):
    os.remove(file_path)

def test_delete_database(mocker):
    # Arrange
    mocker.patch("os.remove")

    # Act
    delete_database("users.db")

    # Assert
    os.remove.assert_called_once_with("users.db")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Triggering error conditions using mocks

```python
import pytest
import random

def divide_by_random(n):
    return n / random.randint(0, 100)

def test_divide_by_random(mocker):
    # Arrange
    mocker.patch("random.randint", return_value=0)

    # Act
    with pytest.raises(ZeroDivisionError) as exc_info:
        result = divide_by_random(1)

    # Assert
    assert str(exc_info.value) == "division by zero"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Raising exceptions in mocks

```python
import json

def load_capitals(file_path):
    with open(file_path) as f:
        return json.load(f)

def test_load_capitals(mocker):
    mocker.patch(
        "json.load",
        side_effect=json.decoder.JSONDecodeError("bad document", "[]", 0)
    )

    with pytest.raises(json.decoder.JSONDecodeError) as exc_info:
        load_capitals("capitals.json")

    assert str(exc_info.value).startswith("bad document")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Testing callbacks with stubs

```python
def do_task(on_complete):
    on_complete("success", 123)

def test_stub(mocker):
    # Arrange

    # Stubs accept any arguments
    # Optional name helps with debugging
    stub = mocker.stub(name="on_complete_stub")

    # Act
    do_task(stub)

    # Assert
    stub.assert_called_once_with("success", 123)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Tracking execution statistics

```python
import math

def combinations(n, r):
    return math.factorial(n) / (math.factorial(r) * math.factorial(n - r))

def test_combinations(mocker):
    # Arrange
    spy = mocker.spy(math, "factorial")

    # Act
    actual_combinations = combinations(15, 4)

    # Assert
    assert actual_combinations == 1365
    assert spy.call_count == 3
    assert spy.spy_return == math.factorial(15 - 4)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Mocking API calls with requests-mock

```python
# requests_mock is a fixture from the requests-mock library
# services.fetch_todos uses requests.get
# services.TODOS_URL = "http://jsonplaceholder.typicode.com/todos"
def test_fetch_todos_with_404_returns_service_error(requests_mock):
    # Arrange
    requests_mock.get(services.TODOS_URL, status_code=404, json=[])

    # Act
    with pytest.raises(services.ServiceError) as exc_info:
        services.fetch_todos()

    # Assert
    assert exc_info.value.status_code == 404
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Mock out-of-memory errors

Try it for yourself

- **It's difficult to test certain scenarios in real-world use**
  - Mocking allows us to test these edge cases
- **Review the `load_data` function in data_store.py**
- **Test that `load_data` returns an empty list when json.load raises a `MemoryError`**
- **Test that other errors, such as `FileNotFound`, are not trapped**
- **Solution:**
  - test_data_store_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Unit testing is ubiquitous in modern software development**
  - pytest is a popular unit testing framework
  - It's simple, yet powerful
- **Coverage.py can be used to calculate test coverage**
  - Can be used with pytest-cov plug-in
  - 100% coverage isn't mandatory
- **Mocking breaks dependencies in tightly coupled code**
  - pytest-mock wraps Python's built-in mocking capabilities to provide a friendlier API
  - requests-mock provides support for mocking API requests

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Packages

LEARNING TREE™
INTERNATIONAL

Modularisation

- **Modular programming is a best practice that breaks large, unwieldy app into distinct, smaller, more manageable units or *modules***
- **Advantages include**
  - Simplicity
  - Reusability
  - Maintainability
  - Namespacing
- **Python supports modular programming through**
  - functions
  - modules
  - packages
- **Modules can be written in Python or C (or C++, Rust, etc)**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- ► **Creating packages**
- ► **Preparing packages for publication**
- ► **Publishing packages to PyPI**
- ► **Using a personal PyPI**
- ► **Freezing applications**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Packages

- **Packages are Python's highest level of modularisation**
- **Primary mechanism for distributing code to third-parties**
- **Collections of modules**
- **Hierarchical structure**
  - Uses dot notation
- **Built on top of the operating system's file structure**
  - Lightweight system
  - Packages are folders of modules
  - Namespaces reflect the folder structure

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Package structure

```
.
├── betting_markets
│   ├── basketball
│   │   ├── __init__.py
│   │   └── markets.py
│   ├── dog_racing
│   │   └── markets.py
│   ├── horse_racing
│   │   └── markets.py
│   ├── ice_hockey
│   │   └── markets.py
│   ├── motorsport
│   │   └── markets.py
│   └── soccer
│       ├── england
│       │   └── markets.py
│       ├── germany
│       │   └── markets.py
│       ├── italy
│       │   └── markets.py
│       ├── markets.py
│       └── spain
│           └── markets.py
└── main.py
```

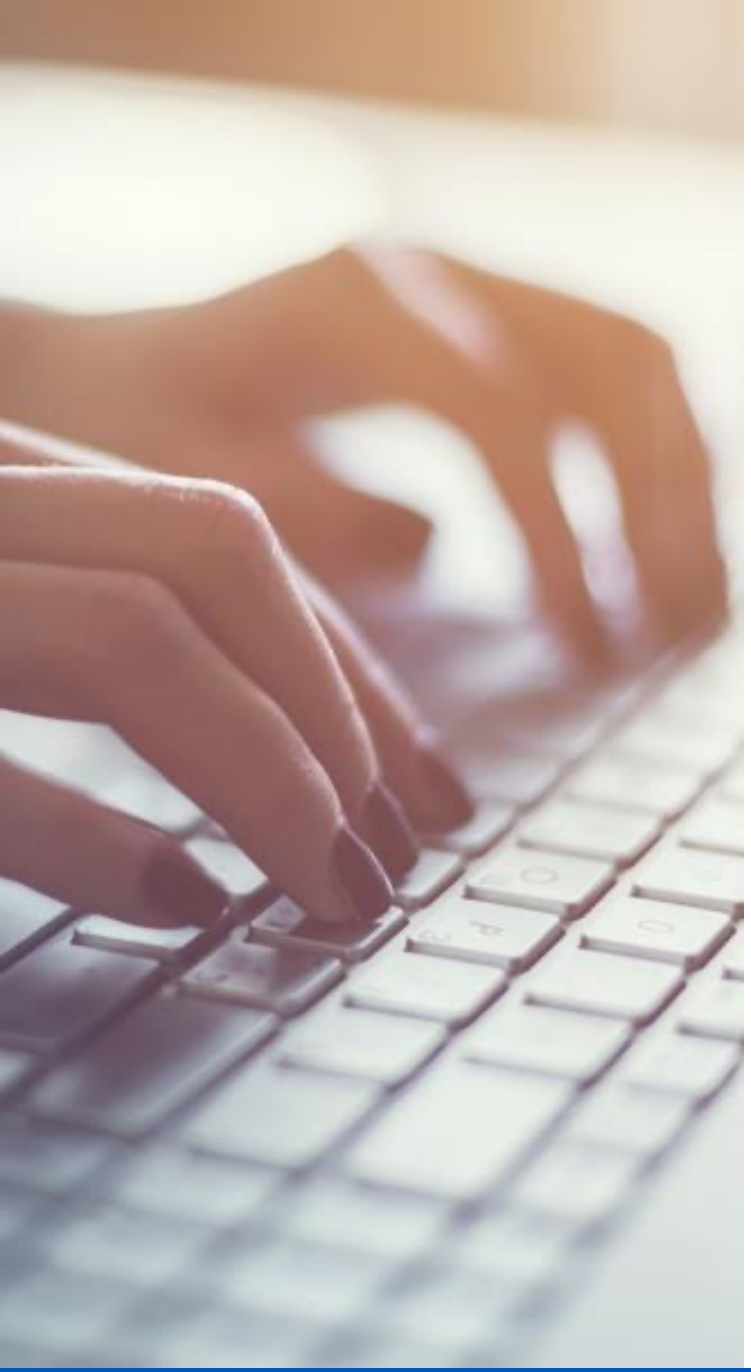
```
import betting_markets.motorsport.markets
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Module resolution strategy

- **When a module is imported, the interpreter needs to find it**
- **The following sources are searched, in order**
  - Folder from which the script, or current folder if running interactively in the interpreter
  - Folders listed in the PYTHONPATH environment variable
  - Installation-dependent folders that are configure when Python is installed
  - Built-in modules
- **If two modules have the same name, then the first one found is used**
- **The module search path can be accessed via `sys.path`**
  - `sys.path` can be updated to produce a dynamic resolution strategy

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a package

Try it for yourself

- **Create a `conversion` package that has the following modules**
  - distance
  - weight
  - temperature
- **Add a suitable function to each module**
- **Use package in simple app by importing the package's modules and calling the functions**
- **Solution:**
  - conversion_app_solution/

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Initialising packages

```python
print("initialising betting_markets.soccer package")

# Default import
import betting_markets.soccer.england.markets

# Wildcard import
__all__ = ["england", "germany", "italy", "spain"]
```

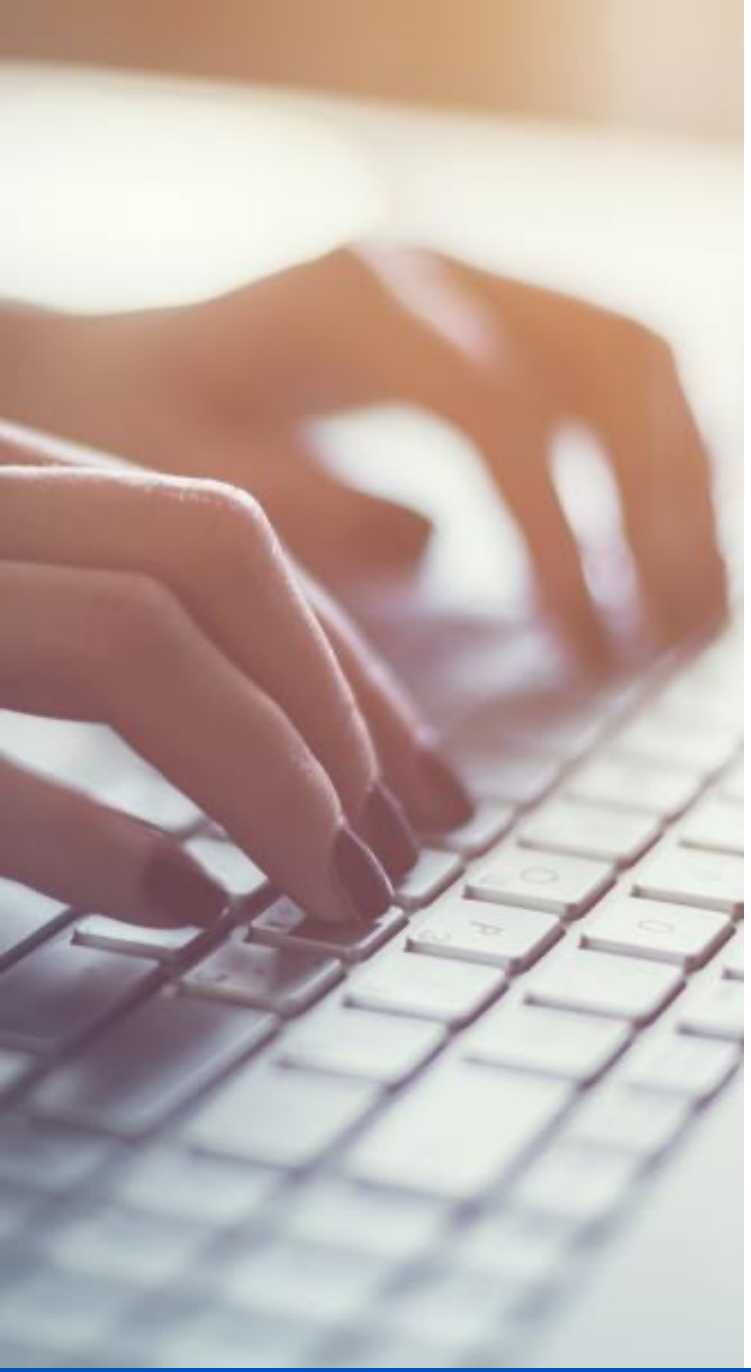
```python
print("initialising betting_markets.soccer.england package")

__all__ = ["markets"]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Executable packages

- **Python can execute scripts using the module name, rather than the file name**
- **The following are equivalent**
  - `python main.py`
  - `python -m main`
- **Built-in modules can be executed**
  - `python –m venv venv`
  - `python -m antigravity`
- **Packages can also be executed in this manner**
  - When a package is executed, code in the __main__.py script in its root is executed
  - An error occurs if there's no __main__.py script (e.g. `python -m json`)
- **Packages can be libraries, executables or hybrids**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Automatically load package modules

Try it for yourself

- **When the `conversion` package is imported, automatically load the distance module**
- **Import all three package modules when a package wildcard import is used**
  - from conversion import *
- **Solution:**
  - package_init_solution/

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Creating packages
- **Preparing packages for publication**
- Publishing packages to PyPI
- Using a personal PyPI
- Freezing applications

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Preparing packages for publication

- Run unit tests
- Naming
- Configuring
- Documenting
- Versioning
- Adding non-code files

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Run unit tests

- Any code that is to be distributed should be tested
- Make sure you code is tested comprehensively before you embark on your publication journey
- If the code is poor, get it into shape before adding more complications
- It's much more stressful to fix code when others are complaining to you about it

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Naming your package

- **"There are only two hard things in Computer Science: cache invalidation and naming things." – Phil Karlton**
- **This is the name that will be visible on PyPI**
  - `pip install betting`
- **Use the PyPI package search feature to see if your name is available**
- **The name used for importing can be different from the PyPI name**
  - Although it easier if they are the same…or similar
- **PEP8 says**
  - Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability.
  - Python packages should also have short, all-lowercase names, although the use of underscores is discouraged

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Configuring your package

pyproject.toml

```
[build-system]
requires = ["setuptools", "wheel"]
build-backend = "setuptools.build_meta"
```

setup.py

```
from setuptools import setup

setup(
    name="betting",
    version="0.0.1",
    packages=["betting"],
)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Documenting your package

- **Documentation is an essential part of a quality package**
- **Can be a basic README file all the way up to a web site**
  - A README file should be a minimum requirement
- **README file should**
  - Briefly describe the project
  - Explain how to install and use the package
- **The README file can be used as the long description in your setup**
- **reStructuredText format has been the norm for Python documentation, but Markdown is now supported**
  - Markdown is is increasingly popular
  - Use the `long_description_content_type` setting to specify the MIME type of your README file (e.g. `text/markdown`)
- **Read the Docs or GitHub can be used for enhanced documentation**
  - The README file can also serve as GitHub documentation

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Versioning your package

- **PyPI restricts you to only one upload of a given *version* of your package**
  - This helps consumers of the package avoid breaking changes
  - Ensures reproducibility
- **A versioning strategy is required to track versions**
- **Semantic versioning is a popular versioning scheme**
  - MAJOR.MINOR.PATCH (e.g. 1.2.7)
  - Increment the MAJOR version when you introduce breaking changes
  - Increment the MINOR version when you add new features that don't alter the API
  - Increment the PATCH version when you fix bugs
- **Version numbers make be place in various locations**
  - e.g. setup.py, __init__.py
  - Bumpversion can be used to update them consistently

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Bumpversion

```
pip install bumpversion
bumpversion --current-version 0.0.1 patch setup.py betting/__init__.py
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Adding non-code files

- **`setup()` creates a manifest file with source code and README files**
  - Usually don't have to worry about it
- **There may be non-source code files that are part of the project**
  - Configuration files
  - SQLite databases
  - CSV files
  - JSON files
  - Images
- **These files need to be explicitly included via a manifest template (`MANIFEST.in`)**
  - Contain rules about what to `include` and `exclude`
  - e.g. `include mypackage/*.png`
- **Tell `setup()` to copy these files**
  - `include_package_data=True`

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Creating packages
- Preparing packages for publication
- **Publishing packages to PyPI**
- Using a personal PyPI
- Freezing applications

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Python Package Index (PyPI)

Most downloaded PyPI packages

Most downloaded past **day**.			Most downloaded past **week**.			Most downloaded past **month**.		
1	botocore	7,046,311	1	botocore	46,218,552	1	botocore	192,746,140
2	urllib3	6,885,489	2	urllib3	43,017,492	2	urllib3	184,170,962
3	six	6,254,148	3	boto3	37,210,438	3	setuptools	160,151,877
4	boto3	6,089,152	4	six	36,810,941	4	idna	158,560,651
5	setuptools	5,784,593	5	s3transfer	36,475,726	5	six	155,579,667
6	s3transfer	5,662,723	6	setuptools	34,674,041	6	boto3	154,859,529
7	idna	5,238,629	7	idna	34,116,262	7	s3transfer	153,697,015
8	requests	4,969,272	8	requests	31,150,281	8	requests	134,743,039
9	python-dateutil	4,925,463	9	python-dateutil	30,853,299	9	python-dateutil	132,397,447
10	awscli	4,630,213	10	awscli	29,582,786	10	awscli	127,390,826
11	pyyaml	4,459,929	11	typing-extensions	27,960,896	11	certifi	118,358,028
12	typing-extensions	4,426,042	12	pyyaml	27,784,357	12	pyyaml	117,765,582
13	certifi	4,412,525	13	certifi	27,424,049	13	typing-extensions	115,020,237
14	wheel	4,272,960	14	colorama	23,767,623	14	colorama	100,296,692
15	click	4,115,318	15	wheel	23,364,536	15	click	97,053,992
16	pandas	3,969,104	16	click	23,169,939	16	wheel	97,044,027
17	markupsafe	3,884,643	17	markupsafe	22,604,356	17	markupsafe	95,205,095
18	jinja2	3,826,349	18	pandas	21,580,829	18	pyasn1	90,590,359
19	numpy	3,732,303	19	jinja2	21,569,881	19	rsa	89,567,783
20	colorama	3,653,842	20	pyasn1	21,329,432	20	attrs	89,240,415

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

TestPyPI

- **Separate instance of PyPI**
- **Allows developers to test out the publication and distribution process without have to do it all on the "live" system**
- **Requires a separate account (i.e. PyPI account details don't work)**
- **Use TestPyPI to ensure everything is working as expected**
  - Don't debug your packaging process on PyPI
- **Process for publishing on TestPyPI is almost identical to the process used on PyPI**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Building your package

- **Packages must be bundled for distribution**
- **Two popular formats for distribution packages**
  - Source distributions
  - Wheels
- **Source distributions are tar archives that contain source code (and other) files**
  - setup.py is executed on the host upon installation
  - Creates a potential security risk
  - Difficult to maintain across multiple platforms/versions
- **Wheels are built distributions**
  - Package is pre-interpreted, so no need to run setup.py post-installation
  - Wheels are smaller and faster than source distributions
  - Pre-built files are distributed as a zip archive
  - pip will use Wheels when they are available
  - Name format is {dist}-{version}|-{build}|-{python}-{abi}-{platform}.whl

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating distributable archives

```
pip install wheel
python setup.py sdist bdist_wheel
```

```
dist
├── betting-0.0.4-py3-none-any.whl
└── betting-0.0.4.tar.gz
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review the contents of your package archive

```
tar tzf dist/betting-0.0.1.tar.gz
```

Confirm all the files are included

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Check you package description

```
pip install twine
twine check dist/*
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Upload your package to TestPyPI

```
pip install twine
twine upload --repository testpypi dist/*
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Install your package from TestPyPI

```
pip install --index-url https://test.pypi.org/simple/ betting
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Publish your package to PyPI

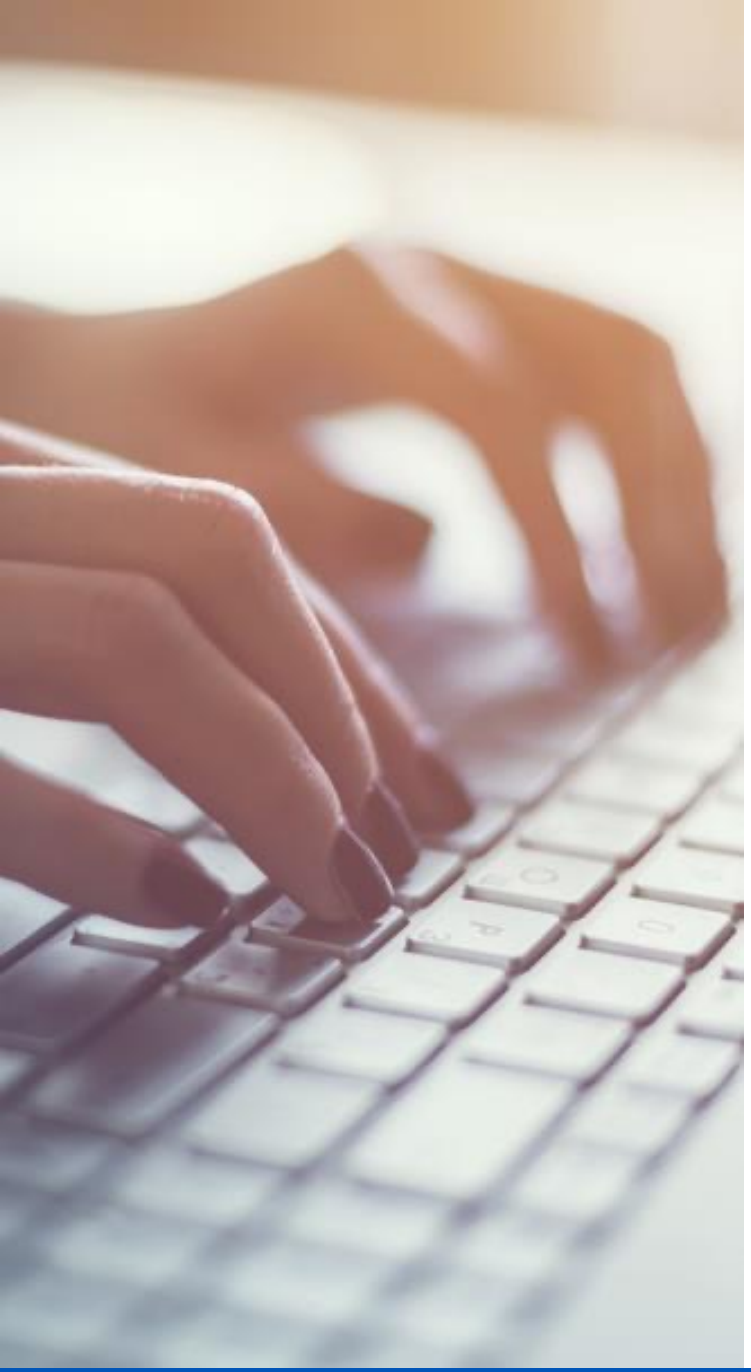
```
pip install twine
twine upload dist/*
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Alternative package management tools

- **Python packaging has evolved over the years**
  - It's always been a bit messy
- **Technologies have come and go**
  - e.g. Eggs have been replaced by Wheels
- **Various tools have appeared that attempt to simplify the process**
- **Popular packaging tools include**
  - Flit
  - Poetry

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Publish a Python package

Try it for yourself

- Publish a package using the instructions in publish.ipynb

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Creating packages
- Preparing packages for publication
- Publishing packages to PyPI
- **Using a personal PyPI**
- Freezing applications

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Personal PyPI

- **Some organisations may not wish to host packages publicly on PyPI**
  - There may be commercial or security implications to making their code publicly available
- **A simple HTTP server can host packages located in its home directory**
- **These packages can be installed with pip**
  - `pip install --extra-index-url=http://127.0.0.1:9000/lt614nstudent`
- **pypiserver is a minimal PyPI compatible server**
  - pypiserver can automatically retrieve and updated selected packages from PyPI
  - This allows it to host a vetted list of public packages as well as private packages
- **Amazon S3 buckets can also be used as private PyPI servers**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Creating packages
- Preparing packages for publication
- Publishing packages to PyPI
- Using a personal PyPI
- **Freezing applications**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Distributing apps

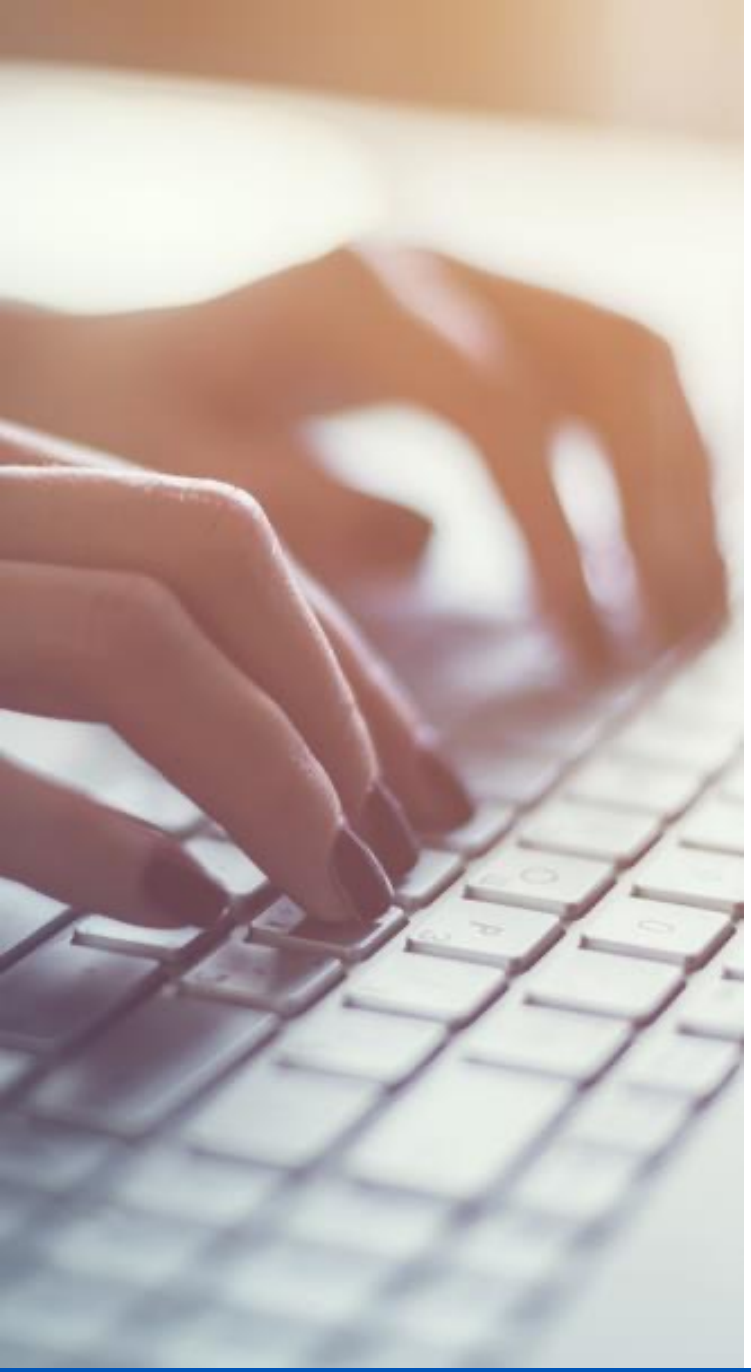
- **PyPI is great for distributing code libraries**
  - It can also be used for distributing applications
- **Installing through pip isn't ideal for those who aren't Python developers**
- **One approach to distributing apps is to create Linux distro packages**
  - Only works on Linux
  - Supporting different distros in a hassle
- **Freezing code creates a single-file executable**
- **The executable contains all the files required to run your application**
  - Including an embedded Python interpreter (much like a virtualenv)
  - Executable is usually silently unpacked to a folder before running, adding a delay to the initial launch time
- **The apps can also be packaged without being bundled into a single file**
  - This allows the app to be packaged by external installers, such as InnoSetup

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

PyInstaller

```
pip install pyinstaller
pyinstaller --onefile main.py

# Examine size of app
ls -l dist/main

# Run app
dist/main
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Freeze a Python app

Try it for yourself

- Create a simple CLI app
- Freeze it as a single-file executable using PyInstaller
- Examine the size of the distributable file
- Run the frozen app

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Freezing tools

Tool	OS	Python 3?	One-file?	Licence
bbFreeze	W/L/M			MIT
cx_Freeze	W/L/M	✓		PSF
py2app	M	✓		MIT
py2exe	W	✓	✓	MIT
PyInstaller	W/L/M	✓	✓	GPL

https://docs.python-guide.org/shipping/freezing/#freezing-your-code-ref

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Packages are the highest level of modularisation in Python---and the primary code distribution vehicle**
- **PyPI is used to distribute public packages**
  - TestPyPI provides a playground for testing before deployment
  - Packages can be hosted on private servers
- **Freezing tools, such as PyInstaller, app Python apps to be bundled into a single distribution file**
  - A Python interpreter is included, so the end-user doesn't have to have one installed
  - The downside is package size

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Concurrency

LEARNING TREE™
INTERNATIONAL

Contents

- ► **Concurrency in Python**
- ► **threading**
- ► **asyncio**
- ► **multiprocessing**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Concurrency

- **Simultaneous execution**
  - Threads
  - Tasks
  - Processes
- **Often only an illusion of simultaneity**
  - Application only uses one processor core
  - Rapid "turn taking"
  - Work is interleaved
  - Pre-emptive multitasking requires expensive context switching
  - Cooperative multitasking requires effort from developer
- **Doing the laundry while coding**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Parallelism

- **True simultaneous execution**
  - Instructions are executing at the same time
- **Requires appropriate hardware**
  - Makes full use of multicore/multithreaded CPUs/GPUs
  - GPUs get their performance through massive parallelism (10,000+ cores)
- **Getting your partner to help with painting a room**

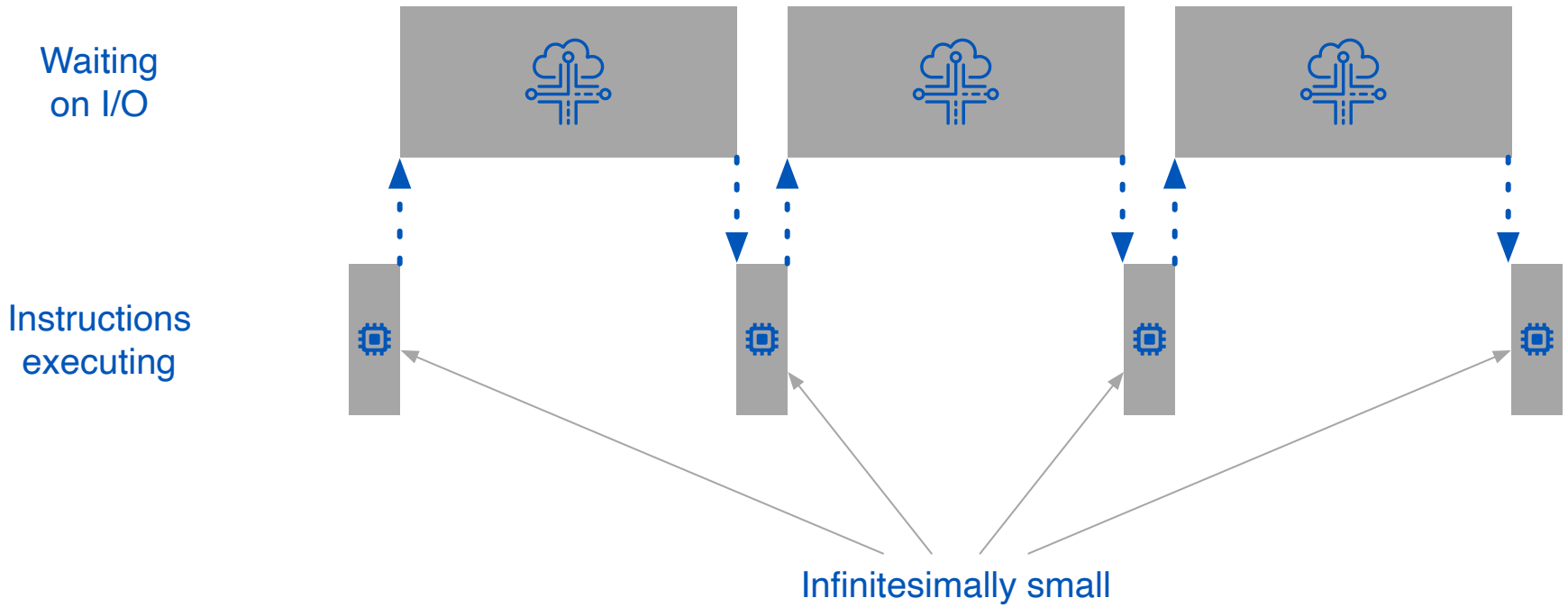
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Types of concurrency

Concurrency type	Switching mechanism	# processors utilised
Pre-emptive multitasking	Controlled by platform	1
Cooperative multitasking	Controlled by application	1
Parallel	No switching	Many

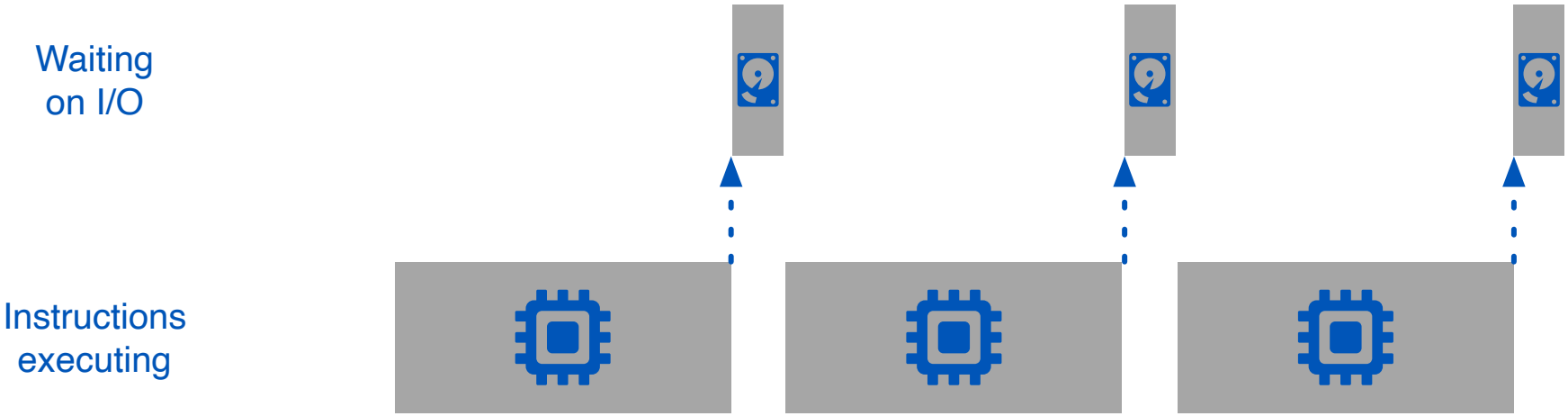
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

I/O-bound execution



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

CPU-bound execution



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Global Interpreter Lock (GIL)

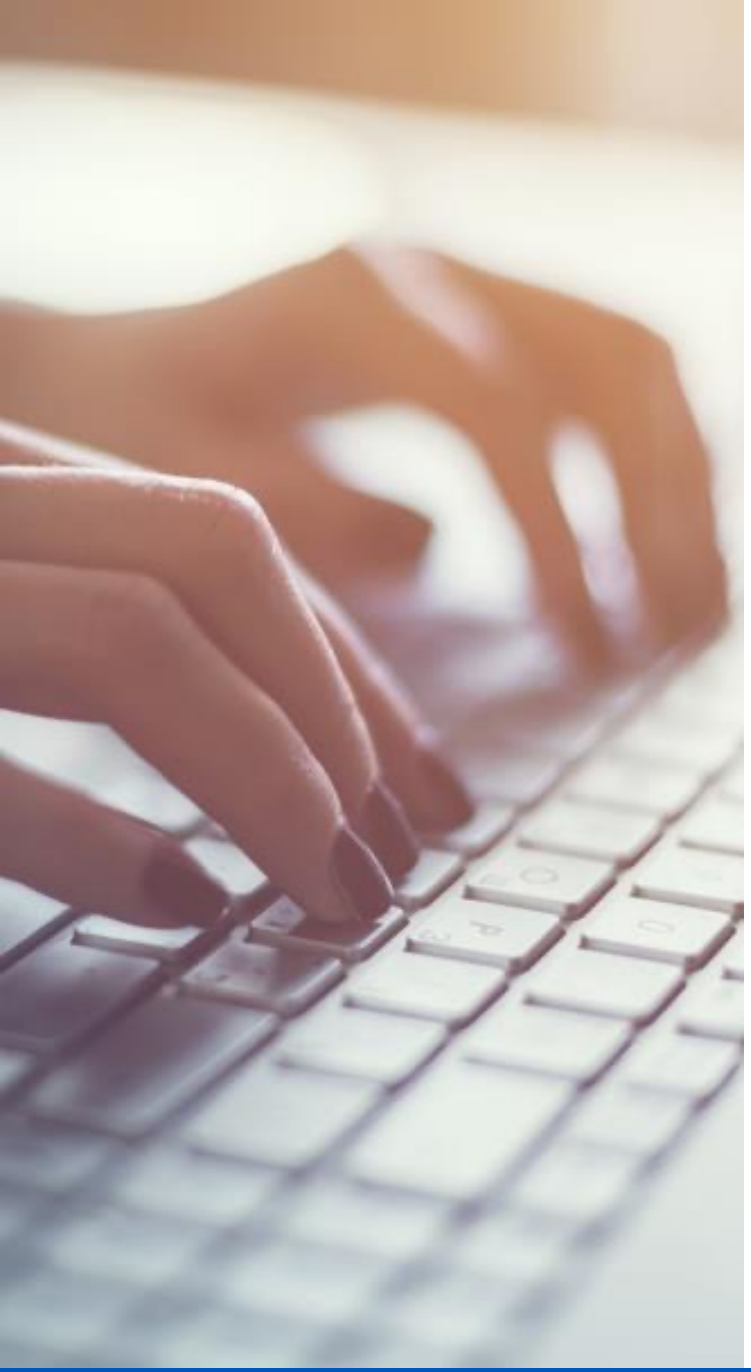
- **Python uses reference counting for memory management**
  - Along with periodic checks for cyclical references
- **Locking ensures the integrity of reference counts**
- **GIL is a single lock on the Python interpreter**
  - Infamously, execution of Python bytecode requires acquiring the GIL
- **GIL has performance benefits, at the cost of making CPU-bound apps single-threaded**
  - Removing the GIL would make Python slower (!)
  - "I'd welcome a set of patches into Py3k only if the performance for a single-threaded program (and for a multi-threaded but I/O-bound program) does not decrease" – Guido van Rossum
- **Made it easy for Python to implement C extensions using C libraries that required a thread-safe environment**
  - It's been argued that this was instrumental to Python's success

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Synchronous execution

```python
import time

def do_something(n):
    print(f"{n + 1}: Zzz...")
    time.sleep(1)

start_time = time.perf_counter()

for _ in range(10):
    do_something()

duration = time.perf_counter() - start_time

print(f"Took {duration:.2f} seconds")

# Took 10.01 seconds
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Explore an I/O bound app

Try it for yourself

- **Use the script in synchronous_quotes.py to display an inspirational quote**
  - `python synchronous_quotes.py 1`
  - How long did it take?
- **Download 3, 10 and 100 quotes**
  - How long did they take?
  - Make a note of the time take to download 100 for future reference
- **Examine the code**
  - Pay particular attention to the `fetch_quotes` and `fetch_quote` functions
- **Where is the time being spent in this app?**
- **Do you feel inspired?**

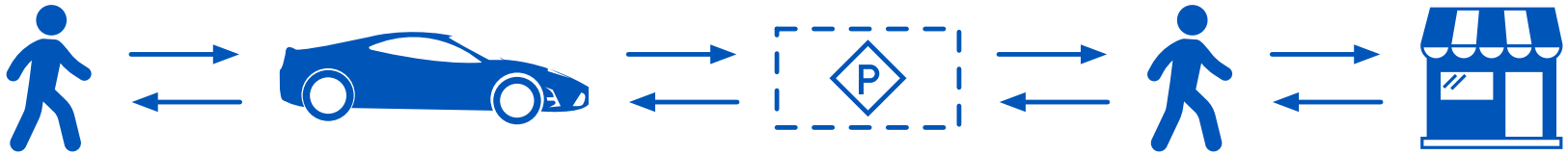
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Benefits of synchronous code

- **There's a reason synchronous code is the norm**
- **Statements execute in a linear, predictable order**
- **It's much easier to reason about when**
  - Writing
  - Debugging
  - Optimising
  - Maintaining
- **Asynchronous code can exhibit strange, "random" behaviour that is difficult to pin down**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Less haste, more speed?



vs

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Concurrency in Python
- **threading**
- asyncio
- multiprocessing

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

threading

- **Python distributions ship with a threading module as standard**
  - Was optional until Python 3.7
- **Provides pre-emptive multitasking**
  - Threads are switched in and out
  - Overhead in setting up and switching threads
- **App has no control over switching**
  - Which can result in difficult to track errors
- **Threads *may* run on different processers, but only one will be running at a time**
- **Speeds up I/O-bound apps**
  - No use for CPU-bound apps
- **Implementation other than CPython may have different models**
- **Threads written in C can release the GIL and run threads in parallel**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Starting threads

```python
import threading
import time

def do_something(n):
    print(f"{n + 1}: Zzz...")
    time.sleep(1)

thread = threading.Thread(target=do_something, args=(0,))

print("Starting thread")
thread.start()
print("Thread running")
thread.join()
print("Finished")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Identifying threads

```python
import threading
import time

def do_something():
    print(f"Running thread with ID {threading.get_ident()}")
    print(f"Running thread {threading.current_thread().getName()}")
    time.sleep(1)

thread = threading.Thread(target=do_something)
dave_thread = threading.Thread(name="Dave", target=do_something)

thread.start()
dave_thread.start()

# Non-daemonic threads are joined automatically on shutdown
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Explicitly waiting for threads to complete

```python
import threading
import time

def do_something():
    print(f"Running thread with ID {threading.get_ident()}")
    print(f"Running thread {threading.current_thread().getName()}")
    time.sleep(1)

thread = threading.Thread(target=do_something)
dave_thread = threading.Thread(name="Dave", target=do_something)

thread.start()
dave_thread.start()

thread.join()
dave_thread.join()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Daemonic threads

```python
import threading
import time

def do_something():
    print(f"Running thread with ID {threading.get_ident()}")
    print(f"Running thread {threading.current_thread().getName()}")
    time.sleep(3600)

thread = threading.Thread(target=do_something, daemon=True)

thread.start()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Spawning multiple threads

```python
import threading
import time

def do_something(n):
    time.sleep(1)
    print(f"{n + 1}: Did something")

threads = []

for i in range(10):
    thread = threading.Thread(target=do_something, args=(i,))
    threads.append(thread)
    thread.start()

for i, thread in enumerate(threads):
    thread.join()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

concurrent.futures

- `concurrent.futures` provides a high-level API for managing concurrency
- Part of the standard library
- Supports multithreading
  - Via `ThreadPoolExecutor`
- Support multiprocessing
  - Via `ProcessPoolExecutor`
- Uses pools of threads (or process) workers
- APIs are consistent across multithreading and multiprocessing
- Executors manage pools of workers
- Futures are used to manage results computed by workers
- Executors launch tasks in the pool and return futures
- High-level APIs mean futures don't have to be managed directly

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Managing groups of threads

```python
import concurrent.futures
import time

def do_something(n):
    print(f"{n + 1}: Zzz...")
    time.sleep(1)

start_time = time.perf_counter()

with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    executor.map(do_something, range(10))

duration = time.perf_counter() - start_time

print(f"Took {duration:.2f} seconds")

# Took 2.00 seconds
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Debugging ThreadPoolExecutor problems

- `ThreadPoolExecutor` will swallow exceptions
- If you pass the wrong number of parameters to your mapped function, the program will end with no output
- Will confuse you every time you return to working with it
- Start with a simple function invocation and build it out

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Returning results from threads

```
import concurrent.futures
import time

def cube(n):
    time.sleep(1)

    return n ** 3

with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    cubes = executor.map(cube, range(1, 11))

print(list(cubes))
# [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Launching individual threads

```
import concurrent.futures
import time

def cube(n):
    time.sleep(1)

    return n ** 3

with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    wait_for = [executor.submit(cube, i + 1) for i in range(10)]

print([f.result() for f in concurrent.futures.as_completed(wait_for)])
# [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Race conditions

```python
import threading
import time

total = 0

def add(n):
    global total
    new_total = total + n
    time.sleep(1)
    total = new_total

thread1 = threading.Thread(target=add, args=(10,))
thread2 = threading.Thread(target=add, args=(20,))
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print(total) # 20
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Locking threads

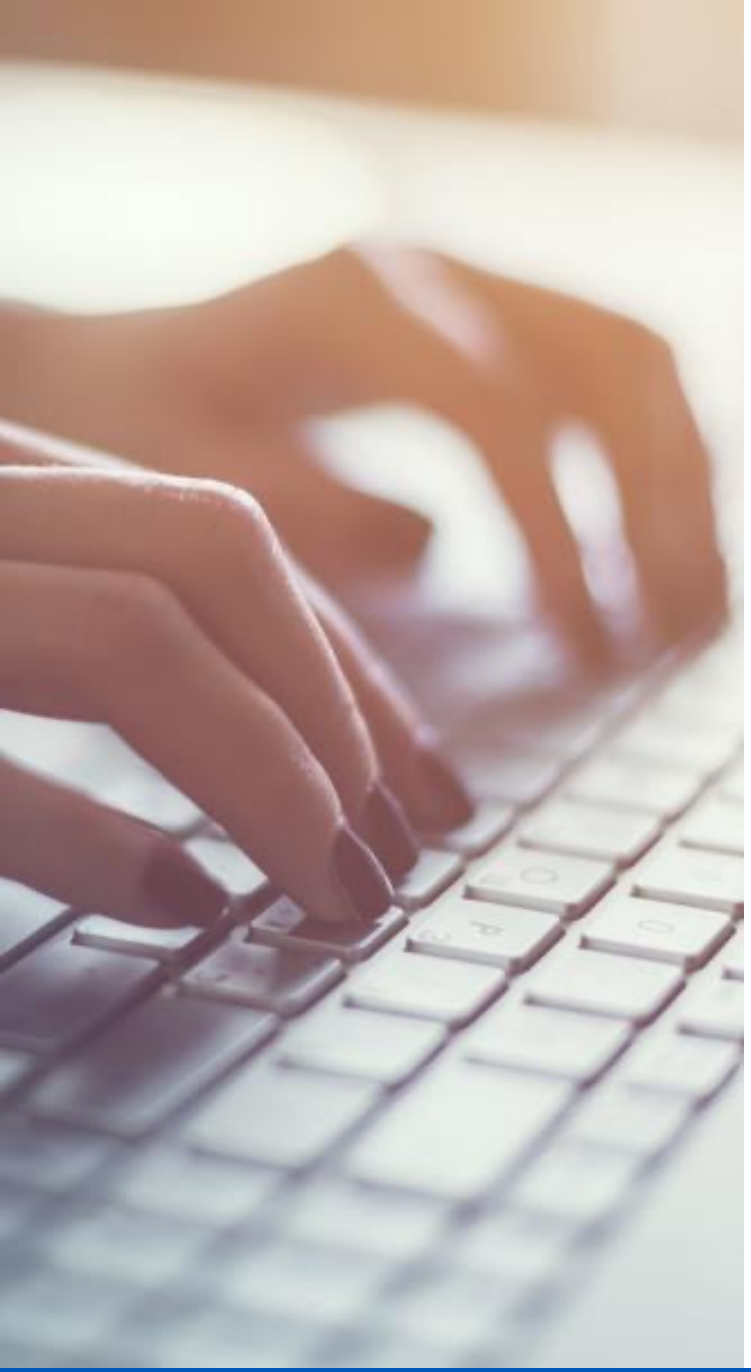
```
lock = threading.Lock()

def add(n):
    global total

    lock.acquire()

    new_total = total + n
    time.sleep(1)
    total = new_total

    lock.release()
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Local thread state

```python
import requests
import threading

thread_local = threading.local()

def get_session():
    """Create a session for each thread as session isn't thread-safe."""
    if not hasattr(thread_local, "session"):
        thread_local.session = requests.Session()
    return thread_local.session
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Create a multithreaded app

Try it for yourself

- **Update the script in multithreaded_quotes.py to fetch quotes in threads**
  - Use a ThreadPoolExecutor
- **Download 100 quotes**
  - How long did it take?
  - Make a note of this value for future reference
- **How does this compare to the time taken in the synchronous version**
  - If there's a difference, what's the explanation?
- **Solution;**
  - multithreaded_quotes_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Concurrency in Python
- threading
- **asyncio**
- multiprocessing

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Async IO

- **Cooperative multitasking**
  - The app controls when tasks are run
- **Execution isn't rudely interrupted, so we can use non-thread-safe code**
- **Developers have to define how the tasks cooperate**
  - When a task will cede to another task
  - Addition complexity at the code level
- **An event loop monitors the tasks**
  - Tasks are in control until they hand control back to the event loop
  - A waiting task is then scheduled and that keeps control until *it's* ready to hand back to the event loop
- **Rogue tasks can undermine concurrency by never relinquishing control**
- **Effective in improving I/O-bound apps**
  - Offers little value to CPU-bound apps
- **Implementation has matured over the last few years**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Coroutines

- **Coroutines are at the heart of async IO**
- **Specialised generator functions that can suspend their execution before they return**
- **Standard library provides the asyncio package**
  - Contains many features that support async IO development
- **`async`/`await` keywords added to Python 3.5**
  - Deprecated techniques still around
- **`async def` is used to define a coroutine**
- **`await` signifies that control can be handed back to the event loop at the point it's used**
- **You can only await "awaitable" functions**
  - These are usually other coroutines
  - Need to use libraries that are async IO "aware"

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating coroutines

```python
import asyncio
import time

async def do_something(n):
    print(f"{n + 1}: Zzz...")
    await asyncio.sleep(1)

async def main():
    start_time = time.perf_counter()

    await asyncio.wait([do_something(i) for i in range(10)])

    duration = time.perf_counter() - start_time

    print(f"Took {duration:.2f} seconds")
    # Took 1.01 seconds

asyncio.run(main())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Gathering results from coroutines

```python
import asyncio

async def cube(n):
    await asyncio.sleep(1)

    return n ** 3

async def main():
    cubes = await asyncio.gather(*(cube(i + 1) for i in range(10)))

    print(cubes)
    # [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]

asyncio.run(main())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Executing coroutines without awaiting

```python
import asyncio

async def do_something(n):
    print(f"{n + 1}: Zzz...")
    await asyncio.sleep(1)
    print(f"{n + 1}: Sleeping Beauty never awoke")

async def main():
    # Fire and forget
    for i in range(10):
        asyncio.ensure_future(do_something(i))

if __name__ == "__main__":
    asyncio.run(main())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Handling exceptions in coroutines

```
import asyncio
import math

async def sqrt(n):
    await asyncio.sleep(1)

    return math.sqrt(n)

async def main():
    try:
        sqrts = await asyncio.gather(
            *(sqrt(i) for i in range(5, -6, -1)), return_exceptions=False
        )

        print(sqrts)
    except (ValueError) as exc_info:
        print(exc_info)

asyncio.run(main())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

async with/for

- **for loops and context managers follow special protocols in Python**
- **for loops**
  - Require iterables
  - Raise a StopIteration exception when there are no more items
- **context managers (with)**
  - Expect `__enter__` and `__exit__` methods
- **These protocols don't play nicely with async features**
- **`async for` allows iteration over an async source**
  - It does *not* automatically make the iteration concurrent
  - It's just a for loop that works with async methods
- **`async with` allows a runtime context to be created from an async source**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

aiohttp

- **requests is the standard library for making HTTP requests in Python**
  - Unfortunately, it doesn't have async IO capabilities
- **aiohttp is a package that provides an HTTP client/server for asyncio**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

aiohttp ClientSession

```python
import aiohttp
import asyncio

INSPIRATIONAL_QUOTES_URL = (
    "https://inspirational-quotes.azurewebsites.net/api/InspirationalQuotes"
)

async def fetch_quote():
    async with aiohttp.ClientSession() as session:
        async with session.get(INSPIRATIONAL_QUOTES_URL) as response:
            return await response.text()

async def main():
    quote = await fetch_quote()
    print(quote)

if __name__ == "__main__":
    asyncio.run(main())
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Popular async IO-ready packages

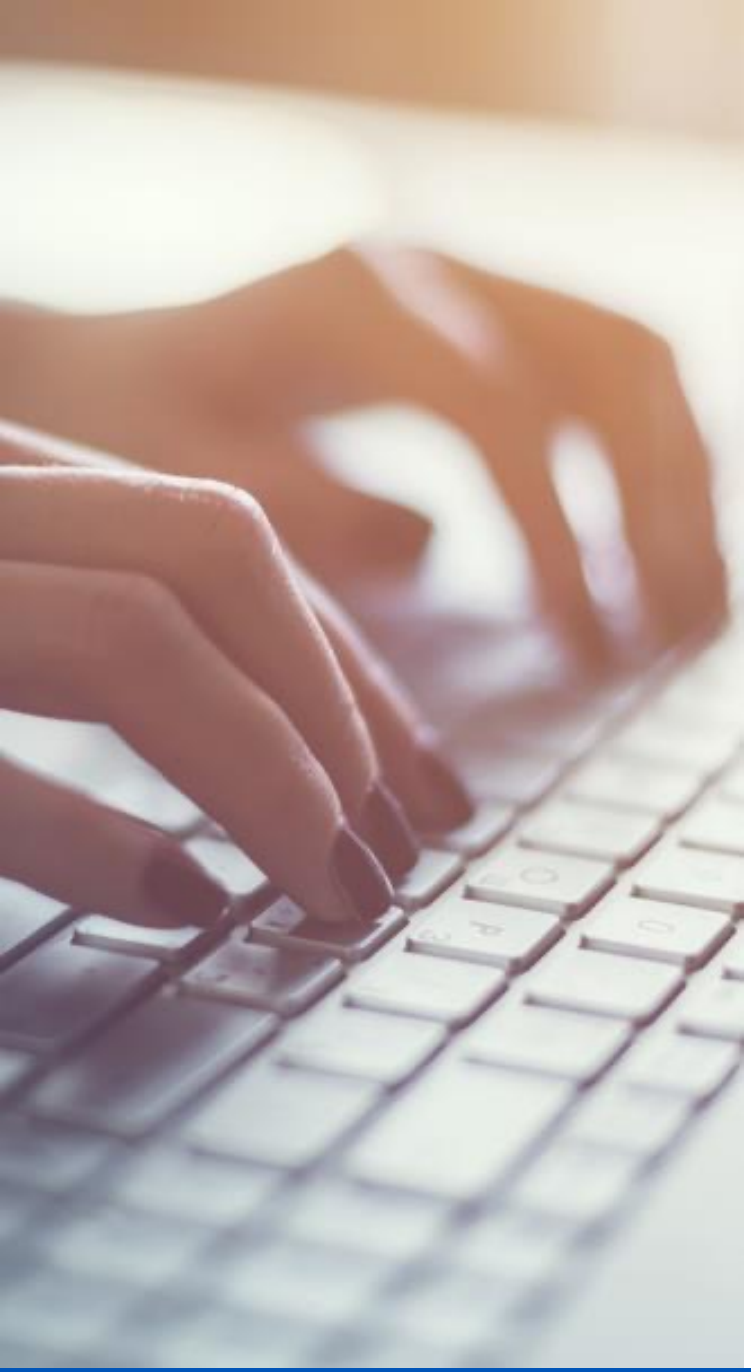
- **aioodbc**
  - Async IO access to ODBC databases
- **aiofiles**
  - Async IO file management
- **aiopg**
  - Async IO access to PostgreSQL databases
- **FastAPI**
  - High performance REST web service framework
- **motor**
  - Async IO driver for MongoDB
- **pytest-asyncio**
  - pytest support for asyncio
- **uvicorn**
  - ASGI server

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Testing async IO functions

```python
# pip install pytest-asyncio
import asyncio
import pytest

async def cube(n):
    await asyncio.sleep(1)
    return n ** 3

@pytest.mark.asyncio
async def test_some_asyncio_code():
    three_cubed = await cube(3)
    assert three_cubed == 27
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Use async IO to enhance performance

Try it for yourself

- **Update the script in asyncio_quotes.py to fetch quotes using coroutines**
  - `gather` the results
- **Download 100 quotes**
  - How long did it take?
  - Make a note of this value for future reference
- **How does this compare to the time taken in the synchronous version**
  - If there's a difference, what's the explanation?
- **Compare with the threaded solution**
- **Solution;**
  - asyncio_quotes_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Concurrency in Python
- threading
- asyncio
- **multiprocessing**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Multiprocessing

- **Multiprocessing gives us true parallelism**
- **Separate processes are run on all the computers cores (or as many as are specified) simultaneous**
- **The only sensible concurrency solution for CPU-bound apps**
- **Each process is given it own Python interpreter**
  - Runs independently
- **Coordinating processes presents challenges**
- **If processes wish to communicate with each other they need to use pipes**
  - Shared storage can also be used
- **Processes are heavyweight, so there's a considerable overhead in initialising the app**
- **GPUs support multiprocessing**
  - Tend to be utilised by higher-level Python libraries (e.g. ML libraries)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

How many CPUs do I have?

```
import multiprocessing

print(f"This machine has {multiprocessing.cpu_count()} core(s)")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

CPU-bound processing

```python
import math
import time

def is_prime(n):
    for i in range(2, int(math.sqrt(n) + 1)):
        if n % i == 0:
            return False

    return True

def find_primes(min, max):
    return [i for i in range(min, max + 1) if is_prime(i)]

start_time = time.perf_counter()
total = sum(find_primes(2, 2000000))
duration = time.perf_counter() - start_time
print(total)
print(f"Calculation took {duration:.2f} seconds")
# Calculation took 8.92 seconds
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

multiprocessing pool

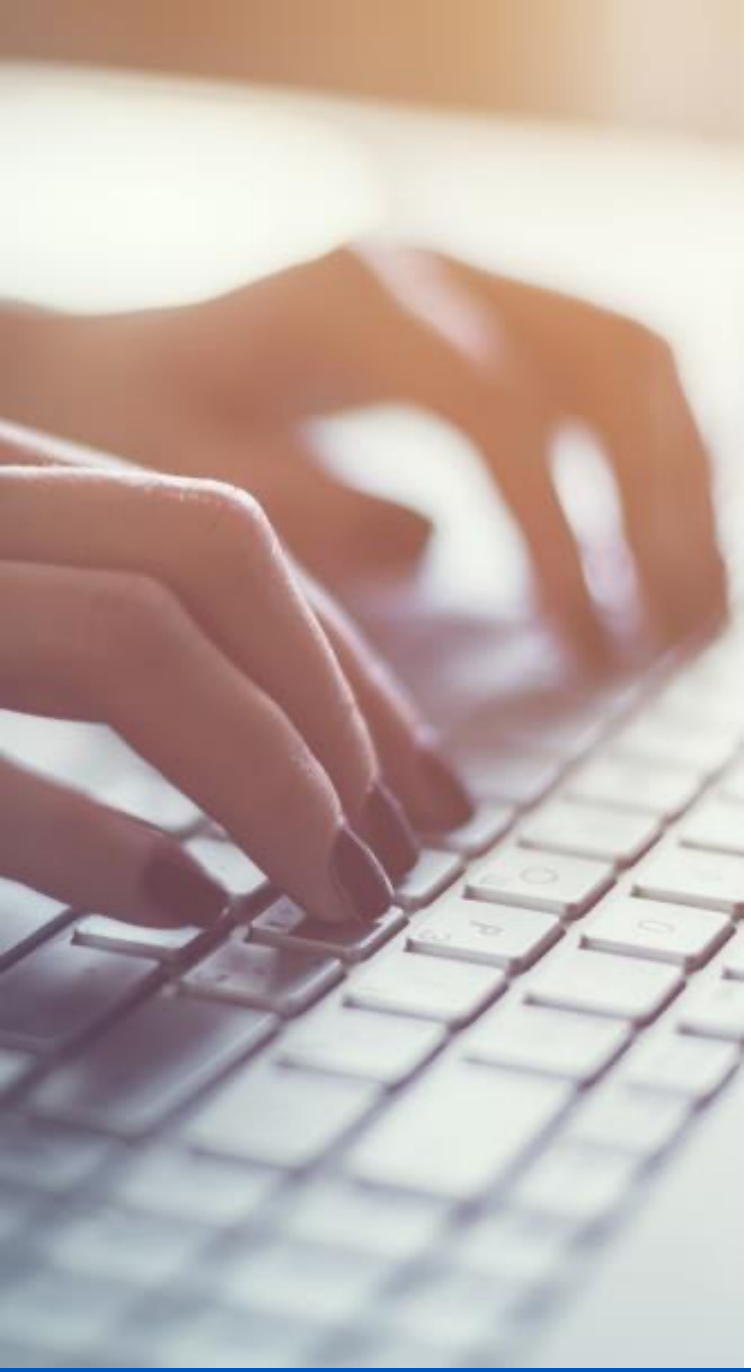
```python
def sum_primes(x):
    return sum(find_primes(x[0], x[1]))

start_time = time.perf_counter()

with multiprocessing.Pool() as pool:
    values = [
        (2, 500_000),
        (500_000, 1_000_000),
        (1_000_000, 1_500_000),
        (1_500_000, 2_000_000),
    ]
    primes = pool.map(sum_primes, values)

total = sum(primes)
duration = time.perf_counter() - start_time
print(total)
print(f"Calculation took {duration:.2f} seconds")
# Calculation took 5.46 seconds (2 cores)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

ProcessPoolExecutor

```python
def sum_primes(x):
    return sum(find_primes(x[0], x[1]))

start_time = time.perf_counter()

with concurrent.futures.ProcessPoolExecutor() as executor:
    values = [
        (2, 500_000),
        (500_000, 1_000_000),
        (1_000_000, 1_500_000),
        (1_500_000, 2_000_000),
    ]
    primes = executor.map(sum_primes, values)

total = sum(primes)
duration = time.perf_counter() - start_time
print(total)
print(f"Calculation took {duration:.2f} seconds")
# Calculation took 5.45 seconds (2 cores)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Fetch quotes using multiprocessing

Try it for yourself

- **Review the script in multiprocessing_quotes.py**
- **It uses a ProcessPoolExecutor to implement concurrency**
  - Compare it with multithreaded code
- **Download 100 quotes**
  - How long did it take?
  - Make a note of this value for future reference
- **How does this compare to the time taken in the multithreaded and async IO versions**
  - If there's a difference, what's the explanation?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Concurrent execution can make a big difference to the performance of your apps**
- **threading and asyncio can speed up I/O-bound apps efficiently**
  - Async IO is generally preferable to multithreading
- **multiprocessing is more suited to CPU-bound apps**
- **Don't use concurrency unless you need it**
  - It adds an additional burden in development and maintenance
- **multiprocessing can be combined with async IO to utilise CPUs**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Profiling

LEARNING TREE™
INTERNATIONAL

Profiling

- **It's easy to write poorly performing code**
- **This didn't used to matter**
  - Clock-cycles are cheap when it's an occasional execution on an end-user machine
- **However, the rise of cloud computing means that we have less redundancy in our systems**
- **The value of optimisation is increasing**
- **It's important to optimise on the critical path**
  - Cheap trainers aren't holding back my marathon running ambitions
- **Profilers help us pinpoint the areas where we need to optimise**
- **It's always worth running a profiler against production code**
  - Small savings mount up

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- ► **Measuring execution time**
- ► **cProfile**
- ► **py-spy**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Timing execution

```
import time

NUMBER = 997

start_time = time.time()
print(is_prime(NUMBER))
duration = time.time() - start_time
print(f"time recorded {duration} second(s)")
# time recorded 3.528594970703125e-05 seconds(s)

start_time = time.perf_counter()
print(is_prime(NUMBER))
duration = time.perf_counter() - start_time
print(f"pref_counter recorded {duration} second(s)")
# pref_counter recorded 4.598000032274285e-06 seconds(s)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

time.perf_counter()

- **`time.perf_counter()` returns the value (in fractional seconds) of a performance counter**
  - i.e. a clock with the highest available resolution to measure a short duration
- **Includes time elapsed during sleep and is system-wide**
- **Reference point of the returned value is undefined**
  - Only the difference between the results of two calls is valid
- **Uses a CPU counter which is monotonically increased at a frequency related to the CPU clock**
- **Very hardware dependent**
  - No clear relationship to real-world time
  - Reset when computer is powered up
- **`time.perf_counter_ns()` returns time as an integer (nanoseconds)**
  - Avoids the precision loss from using the `float` type

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

time.sleep(secs)

- **`time.sleep(secs)` is often used to simulate long running operations**
  - Use `asyncio.sleep(secs)` when working with async IO code
- **Suspend execution of the calling thread for the given number of seconds**
  - Can supply a floating point number to indicate a more precise sleep time
- **Sleeps at *least* `secs`**
- **Don't rely on it being exact**

```python
import time

start_time = time.perf_counter()
time.sleep(1)
duration = time.perf_counter() - start_time
print(f"Slept for {duration} second(s)")
# Slept for 1.002169474999846 second(s)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Timing script execution from terminal

```
time python nth_prime.py 100000
```

```
The 100000th prime is 1299709

real    0m4.491s
user    0m4.468s
sys     0m0.004s
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Verbose timing details

```
command time --verbose python nth_prime.py 100000
```

```
The 100000th prime is 1299709
        Command being timed: "python nth_prime.py 100000"
        User time (seconds): 4.55
        System time (seconds): 0.04
        Percent of CPU this job got: 98%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:04.68
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 8928
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 841
        Voluntary context switches: 1
        Involuntary context switches: 514
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

timeit

- **timeit measures execution time of small code snippets**
- **Has a CLI interface as well as the Python API**
- **Repeats task multiple times to average out variations**
  - Produces a statistical result
- **Disables garbage collection**
- **Picks the most accurate timer for your OS**
  - e.g. `time.perf_counter()`
- **Supported in Jupyter notebooks**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

timeit CLI

```
python -m timeit --setup="import nth_prime" "nth_prime.main(100000)"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

timeit options

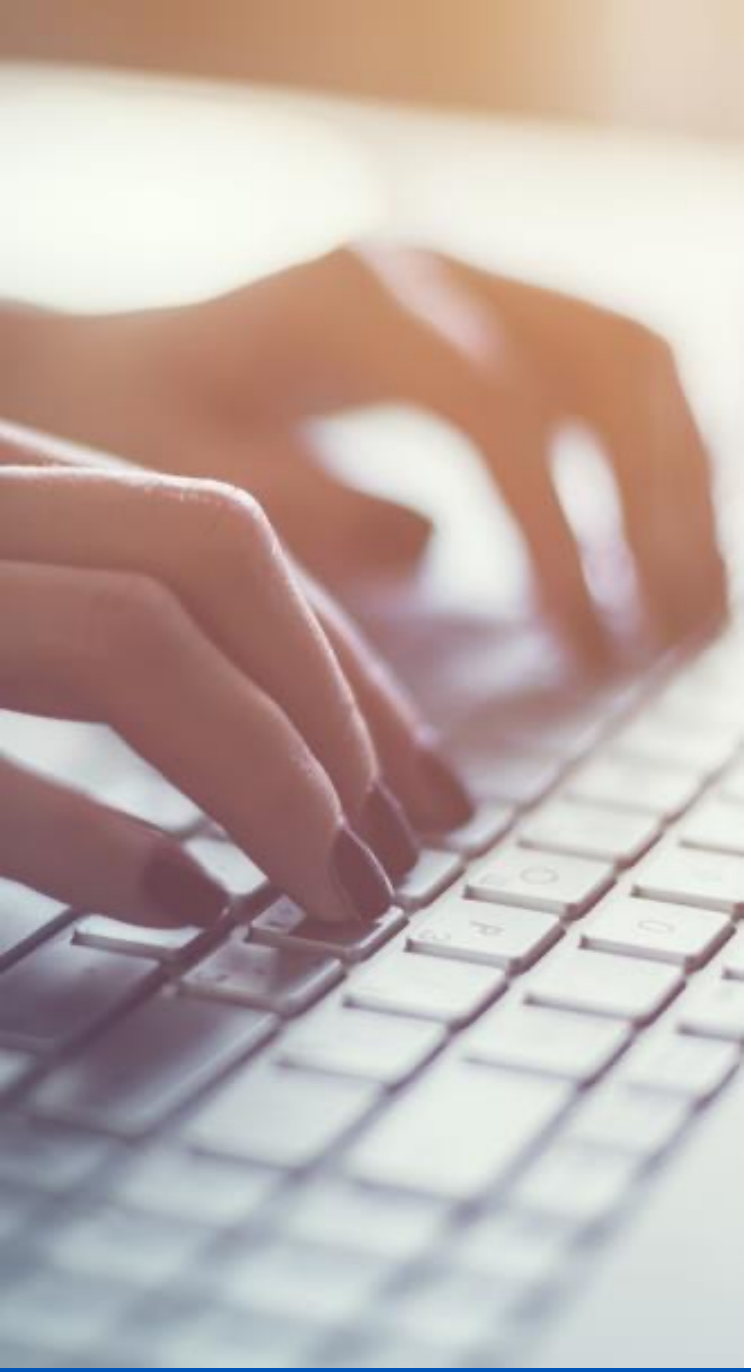
--number=1	Execute statement 1 time
--repeat=5	Repeat the timer 5 times
--setup="pass"	Statement to be executed once initially
--process	Use `time.process_time()` instead of `time.perf_counter()`
--unit=sec	Time unit (nsec, usec, msec, sec)
--verbose	Print raw timing results
--help	Display usage information

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

timeit API

```python
import statistics
import timeit

N = 100_000
t = timeit.repeat(stmt=lambda: main(N), repeat=3, number=1)
print(t)
print(f"Mean execution time is {statistics.mean(t):.2f}")
# [4.5360433330006344, 4.648870472999988, 4.670502273000238]
# Mean execution time is 4.62
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Measure execution time

Try it for yourself

- **Measure the difference in execution time between testing for membership or a large list versus an equivalent set**
- **Use**
  - `time.perf_counter()`
  - timeit CLI
  - timeit API
- **How do the results compare?**
- **Is there variation between timeit repeats?**
- **What code snippets might vary more?**
- **Solution:**
  - execution_time_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Measuring execution time
- **cProfile**
- py-spy

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

cProfile

- **Profiler included in the standard library**
- **Deterministic profiler**
  - Traces every function call in the program
- **Measures how much time elapsed during the function run**
  - This means it can pick up problems like sleeping and lock contention
- **C extension, with reasonable overhead, so can be used to profile long-running code**
- **Reports**
  - **ncalls**: Number of calls made
  - **tottime**: Total time taken (excluding sub-functions)
  - **percall**: Total time taken / Number of calls made
  - **cumtime**: Total time taken (including sub-functions)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

cProfile CLI

```python
# Output stats table
python -m cProfile --sort=tottime nth_prime.py 100000
# Write stats to file
python -m cProfile --sort=tottime --out stats.out nth_prime.py 100000
```

```
The 100000th prime is 1299709
         2599476 function calls in 4.708 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  1299708    4.453    0.000    4.526    0.000 nth_prime.py:5(is_prime)
        1    0.182    0.182    4.707    4.707 nth_prime.py:26(main)
  1299708    0.072    0.000    0.072    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 {built-in method _imp.create_builtin}
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:986(_find_and_load)
        1    0.000    0.000    4.708    4.708 nth_prime.py:1(<module>)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:157(_get_module_lock)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:650(_load_unlocked)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:477(_init_module_attrs)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:406(spec_from_loader)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:890(_find_spec)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:78(acquire)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:103(release)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:956(_find_and_load_unlocked)
```

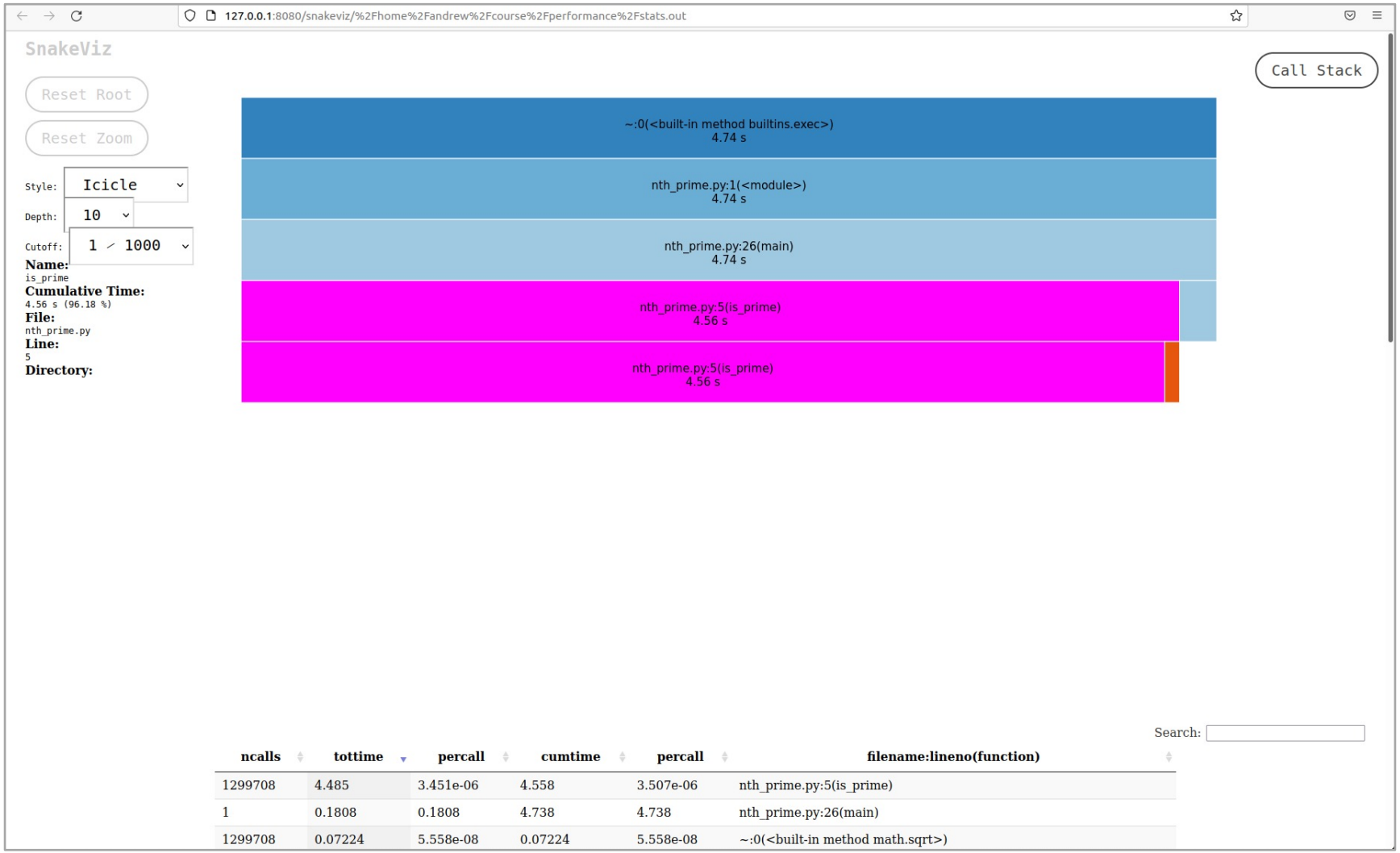
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Reviewing cProfile statistics (using pstats)

```
(venv) andrew@styria:~/course/performance$ python -m pstats
Welcome to the profile statistics browser.
% help

Documented commands (type help <topic>):
========================================
EOF  add  callees  callers  help  quit  read  reverse  sort  stats  strip

% read stats.out
stats.out% sort tottime
stats.out% stats
Tue Oct 12 15:49:37 2021    stats.out

         2599476 function calls in 4.739 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  1299708    4.485    0.000    4.558    0.000 nth_prime.py:5(is_prime)
        1    0.181    0.181    4.738    4.738 nth_prime.py:26(main)
  1299708    0.072    0.000    0.072    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {built-in method _imp.create_builtin}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:986(_find_and_load)
        1    0.000    0.000    4.739    4.739 nth_prime.py:1(<module>)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:157(_get_module_lock)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:650(_load_unlocked)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:477(_init_module_attrs)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:406(spec_from_loader)
        1    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap>:890(_find_spec)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Visualising cProfile statistics (using snakeviz)



ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
1299708	4.485	3.451e-06	4.558	3.507e-06	nth_prime.py:5(is_prime)
1	0.1808	0.1808	4.738	4.738	nth_prime.py:26(main)
1299708	0.07224	5.558e-08	0.07224	5.558e-08	~:0(<built-in method math.sqrt>)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

cProfile API

```python
import cProfile
import pstats

cProfile.run(f"find_nth_prime({100_000})", "stats")
profile_statistics = pstats.Stats("stats")
profile_statistics.sort_stats(pstats.SortKey.TIME).print_stats()
profile_statistics.dump_stats("stats.out")
```

```
The 100000th prime is 1299709
Tue Oct 12 16:00:32 2021    nth_prime_stats

         2599422 function calls in 4.828 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  1299708    4.577    0.000    4.651    0.000 nth_prime_profile.py:7(is_prime)
        1    0.177    0.177    4.828    4.828 nth_prime_profile.py:28(find_nth_prime)
  1299708    0.074    0.000    0.074    0.000 {built-in method math.sqrt}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
        1    0.000    0.000    4.828    4.828 {built-in method builtins.exec}
        1    0.000    0.000    4.828    4.828 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 nth_prime_profile.py:15(get_number_suffix)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Problems with cProfile

- **Tracing every function call is a lot of work**
  - cProfile "observing" the code can distort the results
  - The more function calls in an area of code, the more the code will be slowed down (resulting in *biased* distortion)
- **Too much overhead to run in production**
  - But performance under production conditions is what needs to be optimised
- **As the reports cover every single function they are very noisy**
  - Signal gets drowned out
- **Performance is measured for functions only**
  - But functions may behave differently based on their arguments
  - A function's behaviour may not be consistent across the range of inputs it receives

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Measuring execution time
- cProfile
- py-spy

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

py-spy

- **Third-party profiler**
  - Has to be installed
- **Sampling profiler**
  - Checks in on the running application periodically to collect samples of the execution state (call stack snapshots)
  - Uses process_vm_readv (Linux), ReadProcessMemory (Windows) or vm_read (Mac)
  - Samples are analysed to determine execution time of parts of the app
- **Call stack samples allow the performance of a function to be viewed in context**
- **Can profile C extensions**
- **Extremely-low overhead**
- **Can be used in production environments**
- **Written in Rust, for speed**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

py-spy commands

```
(venv) andrew@styria:~/course/profiling$ py-spy help
py-spy 0.3.10
Sampling profiler for Python programs

USAGE:
    py-spy <SUBCOMMAND>

OPTIONS:
    -h, --help       Prints help information
    -V, --version    Prints version information

SUBCOMMANDS:
    record    Records stack trace information to a flamegraph, speedscope or raw file
    top       Displays a top like view of functions consuming CPU
    dump      Dumps stack traces for a target program to stdout
    help      Prints this message or the help of the given subcommand(s)
(venv) andrew@styria:~/course/profiling$ py-spy help record
py-spy-record
Records stack trace information to a flamegraph, speedscope or raw file

USAGE:
    py-spy record [OPTIONS] --pid <pid> [python_program]...

OPTIONS:
    -p, --pid <pid>              PID of a running python program to spy on
        --full-filenames         Show full Python filenames, instead of shortening to show only the package part
    -o, --output <filename>      Output filename
    -f, --format <format>        Output file format [default: flamegraph]  [possible values: flamegraph, raw, speedscope]
    -d, --duration <duration>    The number of seconds to sample for [default: unlimited]
    -r, --rate <rate>            The number of samples to collect per second [default: 100]
    -s, --subprocesses           Profile subprocesses of the original process
    -F, --function               Aggregate samples by function's first line number, instead of current line number
        --nolineno               Do not show line numbers
    -t, --threads                Show thread ids in the output
    -g, --gil                    Only include traces that are holding on to the GIL
    -i, --idle                   Include stack traces for idle threads
    -n, --native                 Collect stack traces from native extensions written in Cython, C or C++
        --nonblocking            Don't pause the python process when collecting samples. Setting this option will reduce the perfomance impact of sampling, but may lead to inaccurate
results
    -h, --help                   Prints help information
    -V, --version                Prints version information

ARGS:
    <python_program>...    commandline of a python program to run
(venv) andrew@styria:~/course/profiling$
```

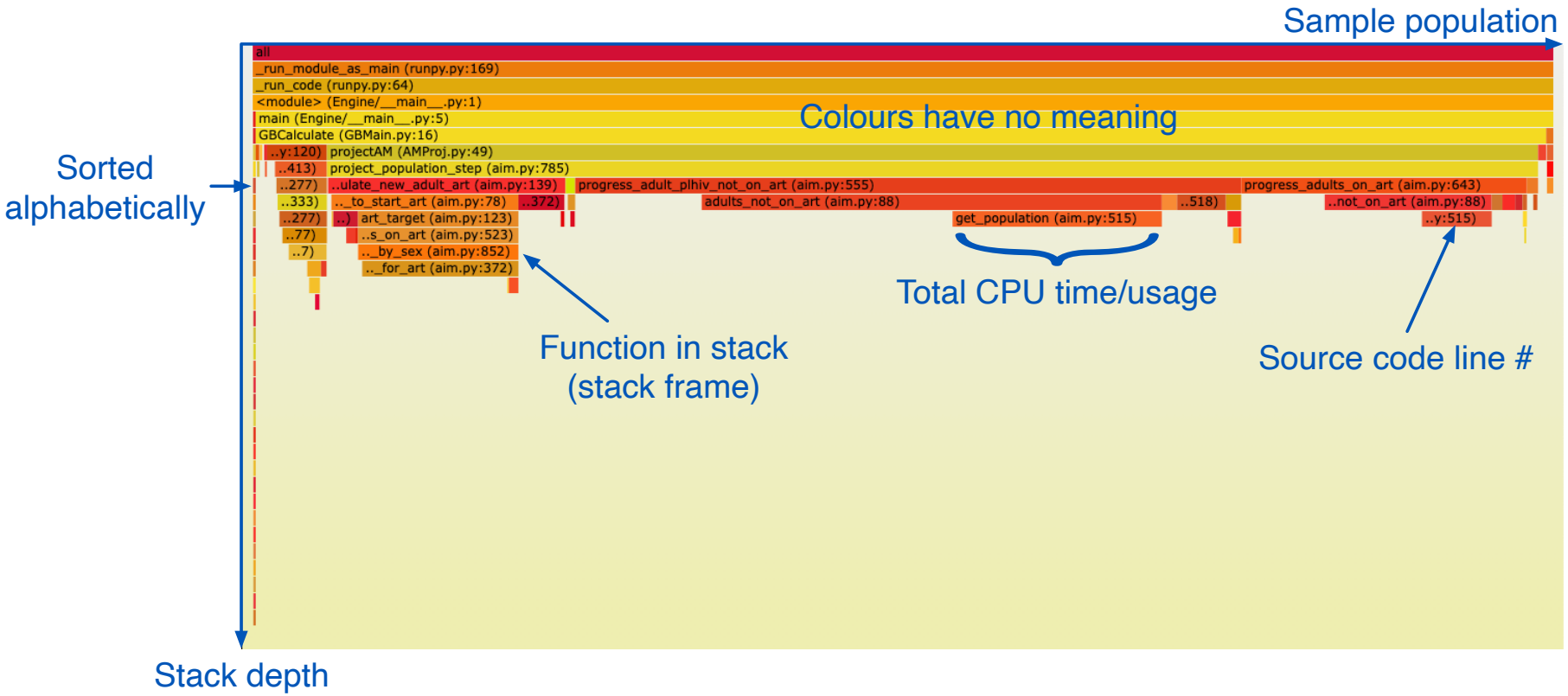
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Profiling with py-spy

```
pip install py-spy
py-spy record -F -o profile.svg -- python nth_prime.py 100000
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Interpreting flame graphs (icicle charts)



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Monitoring profiling

```
py-spy top -- python nth_prime.py 1000000
```

```
Collecting samples from 'python nth_prime.py 1000000' (python v3.8.10)
Total Samples 12700
GIL: 100.00%, Active: 100.00%, Threads: 1

  %Own   %Total  OwnTime  TotalTime  Function (filename:line)
 70.00%  70.00%   91.65s    91.65s   is_prime (nth_prime.py:7)
 30.00%  30.00%   33.76s    33.76s   is_prime (nth_prime.py:6)
  0.00% 100.00%   0.790s    126.5s   main (nth_prime.py:34)
  0.00%   0.00%   0.310s    0.310s   main (nth_prime.py:32)
  0.00%   0.00%   0.210s    0.210s   is_prime (nth_prime.py:8)
  0.00%   0.00%   0.190s    0.190s   main (nth_prime.py:31)
  0.00%   0.00%   0.050s    0.050s   is_prime (nth_prime.py:5)
  0.00%   0.00%   0.030s    0.030s   main (nth_prime.py:35)
  0.00%   0.00%   0.010s    0.010s   is_prime (nth_prime.py:10)
  0.00% 100.00%   0.000s    127.0s   <module> (nth_prime.py:41)


Press Control-C to quit, or ? for help.
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Viewing the current call stack

```
python nth_prime.py 1000000 &
ps -a
# Python process ID was 14115

sudo env "PATH=$PATH" py-spy dump -l -p 14115
```

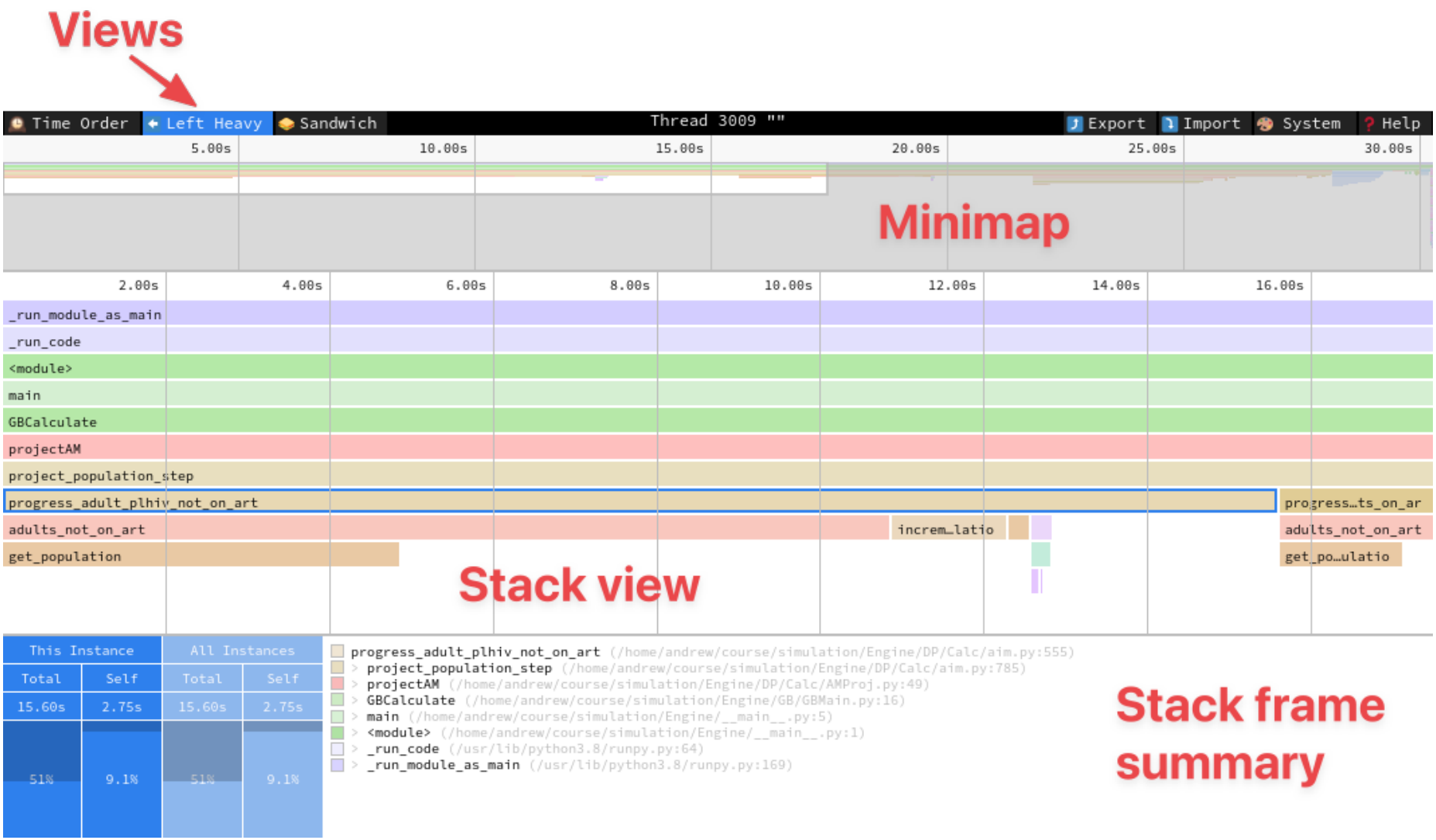
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

speedscope

```
py-spy record -F -f=speedscope -o profile.json -- python nth_prime.py 100000

sudo npm install -g speedscope
speedscope profile.json
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Using speedscope



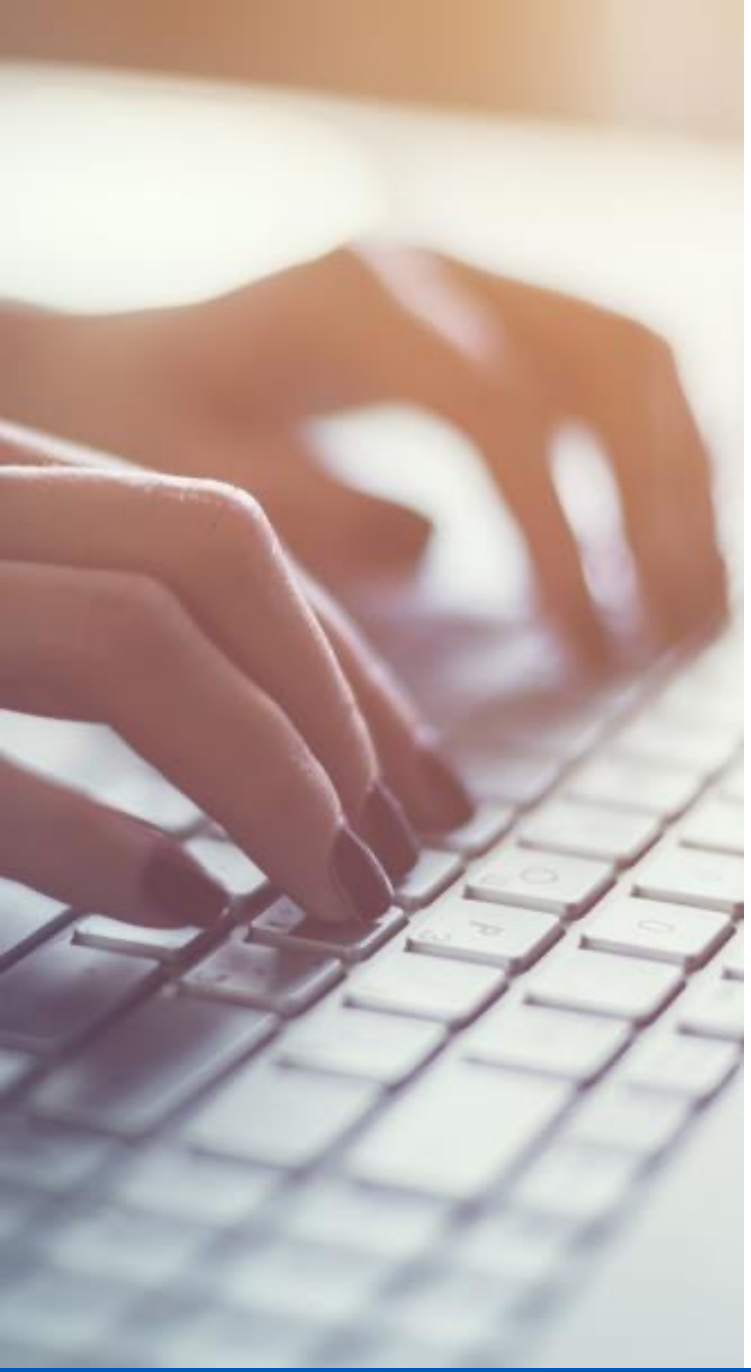
https://github.com/jlfwong/speedscope#usage

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Additional profilers

- **Profile**
  - Part of the standard library
  - Written in Python
  - High overhead
- **Pyinstrument**
  - A sampling profiler
- **yappi**
  - Concurrent execution profiler
  - Coroutine aware
- **Pympler**
  - Memory profiler
- **Eliot**
  - A logging library that logs not only *what* happened, but *why* it happened

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Profile an app

Try it for yourself

- **estimate_pi.py contains an app that uses a statistical approach to estimate π**
- **Profile the app, using py-spy, to see where the bottlenecks are**
- **Optimise the app to improve its performance**
- **Solution:**
  - estimate_pi_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **It's important to use an accurate timer when benchmarking code that runs for a short time**
  - Longer-running code tends not to require the same level of accuracy
- **Inefficient code can be costly**
  - So can over-optimising
- **Optimisation efforts need to be targeted**
  - Profiling identifies those targets
- **Python comes bundled with cProfile**
- **Third-party statistical profilers, such as py-spy, tend to provide better insights**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Improving Performance

LEARNING TREE™ INTERNATIONAL

Contents

- ► **Basic optimisations**
- ► **NumPy**
- ► **Numba**
- ► **JAX**
- ► **PyPy**
- ► **Cython**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

1. Use built-ins

```python
total = 0

for i in range(1_000_000):
    total += i

# Better to...

sum(range(1_000_000))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

2. Don't assume that libraries are fast

```
numbers = list(range(1_000_000))

from statistics import mean
mean(numbers)

# Better to...

sum(numbers) / len(numbers)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

3. Don't manually search lists

```
numbers = list(range(1_000_000))

numbers.index(500_000) != 0

# Better to...

500_000 in numbers
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

4. Make more use of sets

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
fibonacci = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

in_both = []

for p in primes:
    for f in fibonacci:
        if p == f:
            in_both.append(p)

# Better to...

in_both = set(primes) & set(fibonacci)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

5. Use comprehensions

```
odd_cubes = []
for n in range(1_000_000):
    if n % 2 == 1:
        odd_cubes.append(n ** 3)

# Better to...

odd_cubes = [n ** 3 for n in range(1_000_000) if n % 2 == 1]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

6. Use generators

```
numbers = [n ** 3 for n in range(1_000_000) if n % 2 == 1]
sum(numbers[:10])

# Better to...

import itertools
numbers = (n ** 3 for n in range(1_000_000) if n % 2 == 1)
sum(itertools.islice(numbers, 10))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

7. Concatenate strings using join

```
words = [
    "the", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"
]

sentence = ""

for word in words:
    if sentence:
        sentence += " "

    sentence += word

# Better to...

sentence = " ".join(words)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

8. Only load modules when you need 'em

```
if True:
    import math

    print(math.sqrt(9))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

9. Use caching

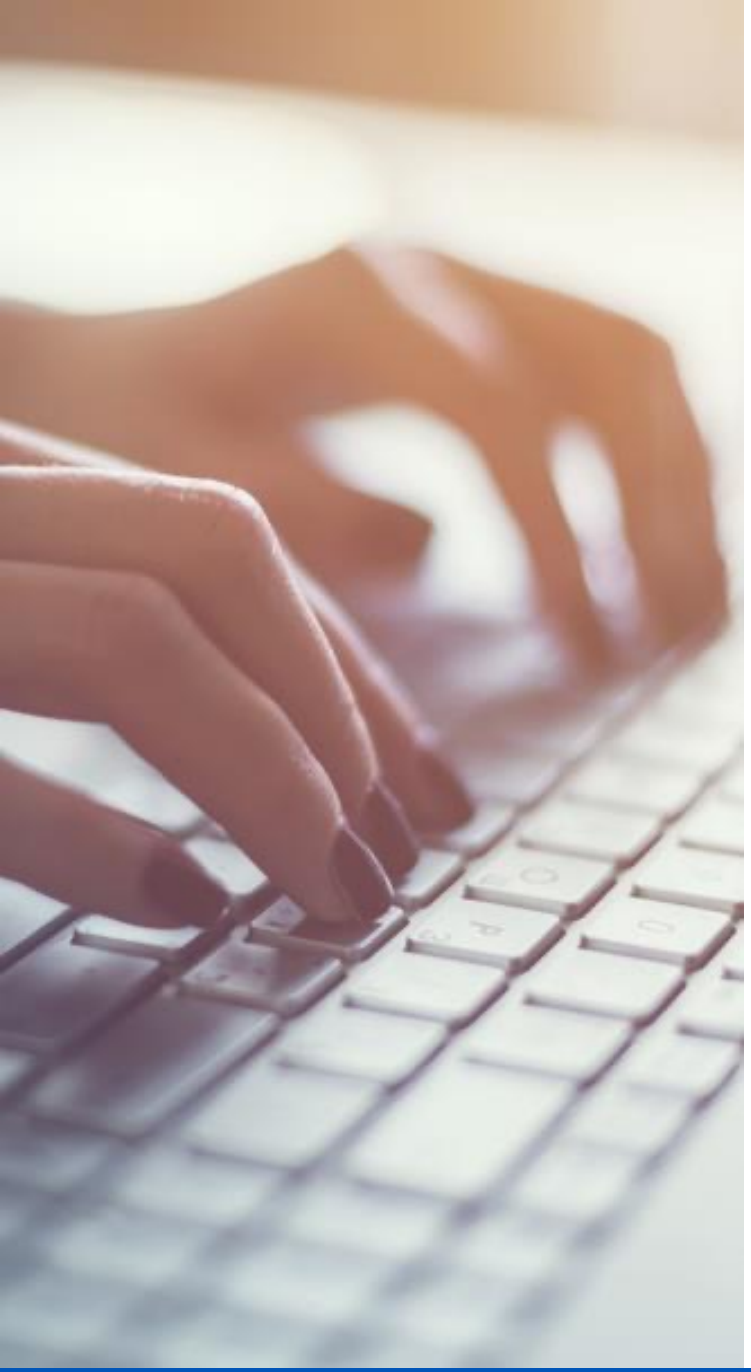
```
import functools

@functools.lru_cache(maxsize=16)
def find_cached_nth_prime(n):
    return find_nth_prime(n)

find_cached_nth_prime(1_000)
find_cached_nth_prime(1_000)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

10. Get to know itertools

```
numbers = list(range(100))

cum_sum = []
running_total = 0

for n in numbers:
    cum_sum.append(running_total + n)
    running_total += n

# Better to...

import itertools
cum_sum = list(itertools.accumulate(numbers))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Evaluate basic optimisations

Try it for yourself

- Work through the basic optimisations notebook (basic_optimisations.ipynb) comparing the execution times of different approaches to problems
- Make changes to the code cells and confirm that the resulting performance is as you expected

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Basic optimisations
- **NumPy**
- Numba
- JAX
- PyPy
- Cython

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

NumPy

- **One of Python's major strengths is the rich collection types that are part of the language**
- **However, pure Python code is slow**
  - Python compensates for this by calling into libraries written in C/C++
- **NumPy is a fixed-length multidimensional arrays/matrices library**
  - These kinds of data structures are the foundation of the types of scientific/analytical tasks that Python excels at
- **NumPy is fast---much faster than working with lists**
  - Highly-optimised C code
- **Many headlining packages in the Python ecosystem build on NumPy**
  - pandas
  - PyStan
  - SciPy
  - Seaborn
  - statsmodels

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Installing NumPy

```
pip install numpy
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating NumPy arrays

```python
import numpy as np

primes = np.array([2.0, 3.0, 5.0, 7.0, 11.0])
print(primes.shape)
# (5,)

tic_tac_toe = np.array(
    [
        [1.0, 0.0, 0.0],
        [0.0, 1.0, 1.0],
        [1.0, 0.0, 1.0],
    ]
)
print(tic_tac_toe.shape)
# (3. 3)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

NumPy arrays have homogenous types

```
import numpy as np

np.array([2, 3, 5, 7, 11]).dtype
# dtype('int64')

np.array([2.0, 3.0, 5.0, 7.0, 11.0]).dtype
# dtype('float64')

np.array(["h", "e", "l", "l", "o"]).dtype
# dtype('<U1')

np.array([2, 3.0, 5, 7, 11]).dtype
# dtype('float64')
```

https://numpy.org/devdocs/user/basics.types.html

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Initialise arrays to the required dimensions

```
import numpy as np

np.zeros((2, 2))
# array([[0., 0.],
#        [0., 0.]])

np.ones((2, 2), dtype=np.int64)
# array([[1, 1],
#        [1, 1]])

np.empty((2, 2))
# array([[4.67226476e-310, 0.00000000e+000],
#        [6.93529876e-310, 6.93537372e-310]])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Initialise arrays using sequences

```
import numpy as np

np.arange(0, 20, 3)
# array([ 0,  3,  6,  9, 12, 15, 18])

np.linspace(0, 10, 4)
# array([ 0.        ,  3.33333333,  6.66666667, 10.        ])

np.fromfunction(lambda i, j: (i + 1) ** (j + 1), (3, 3))
# array([[ 1.,  1.,  1.],
#        [ 2.,  4.,  8.],
#        [ 3.,  9., 27.]])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Initialising arrays using repetition

```
import numpy as np

np.repeat(1, 3)
# array([1, 1, 1])

np.repeat([1, 2], 3)
# array([1, 1, 1, 2, 2, 2])

np.repeat([
    [1, 2],
    [3, 4]
], 3, axis=1)
array([[1, 1, 1, 2, 2, 2],
       [3, 3, 3, 4, 4, 4]])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Interpreting array output

- **All but the last axis are printed from top-to-bottom**
  - Each axis is separated from the next by an empty line
- **Last axis is printed left-to-right**

```python
import numpy as np

def to_id(i, j, k):
    return (i + 1) * 100 + (j + 1) * 10 + k + 1

a = np.fromfunction(to_id, (2, 3, 4), dtype=np.int64)
print(a)
```

```
[[[111 112 113 114]
  [121 122 123 124]
  [131 132 133 134]]

 [[211 212 213 214]
  [221 222 223 224]
  [231 232 233 234]]]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Reshaping arrays

```
import numpy as np

a = np.arange(1, 25).reshape(2, -1, 4)
a
# array([[[ 1,  2,  3,  4],
#         [ 5,  6,  7,  8],
#         [ 9, 10, 11, 12]],
#
#        [[13, 14, 15, 16],
#         [17, 18, 19, 20],
#         [21, 22, 23, 24]]])

a.ravel()
# array([ 1,  2,  3,  4,  5,  6,  7,  8,
#         9, 10, 11, 12, 13, 14, 15, 16,
#        17, 18, 19, 20, 21, 22, 23, 24])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Combining arrays

```
import numpy as np

a = np.arange(1, 5).reshape((2, 2))
b = np.arange(5, 9).reshape((2, 2))

np.hstack((a, b))
# array([[1, 2, 5, 6],
#        [3, 4, 7, 8]])

np.vstack((a, b))
# array([[1, 2],
#        [3, 4],
#        [5, 6],
#        [7, 8]])

np.concatenate((a, b), axis=1)
# array([[1, 2, 5, 6],
#        [3, 4, 7, 8]])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Slicing

```python
import numpy as np

a = np.arange(1, 25).reshape(2, 3, -1)
b = a[1:2, 1:2, 1:2]
print(b)
# [[[18]]]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Indexing

```
import numpy as np

a = np.arange(1, 10, dtype=np.float64).reshape(3, -1)

i = (a >= 2.5) & (a <= 6.5)
print(i)
# [[False False  True]
#  [ True  True  True]
#  [False False False]]

a[i] = np.nan
print(a)
# [[ 1.  2. nan]
# [nan nan nan]
# [ 7.  8.  9.]]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Views

```
import numpy as np

a = np.arange(1, 25, dtype=np.float64).reshape(2, 3, -1)
b = a.view()

b is a # False
b.base is a.base # True

c = a[0:2, 0:2, 0:2]
c[:] = np.nan
print(a)
# [[[nan nan  3.  4.]
#   [nan nan  7.  8.]
#   [ 9. 10. 11. 12.]]
#
#  [[nan nan 15. 16.]
#   [nan nan 19. 20.]
#   [21. 22. 23. 24.]]]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Deep copies

```
import numpy as np

a = np.arange(1, 25, dtype=np.float64).reshape(2, 3, -1)

c = a[0:2, 0:2, 0:2].copy()
c[:] = np.nan

print(a)
# [[[ 1.  2.  3.  4.]
#   [ 5.  6.  7.  8.]
#   [ 9. 10. 11. 12.]]

#  [[13. 14. 15. 16.]
#   [17. 18. 19. 20.]
#   [21. 22. 23. 24.]]]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Vectorised operations

```
import numpy as np

temperatures = np.array([
    [0.0, 50.0, 100.0],
    [-10.0, -50.0, -100.0],
])
temperatures * 9.0 / 5.0 + 32.0
# array([[  32.,  122.,  212.],
#        [  14.,  -58., -148.]])

radii = np.array([1, 3, 10.0])
np.pi * radii ** 2
# array([  3.14159265,  28.27433388, 314.15926536])

revenue = np.array([1_000_000, 1_100_000, 1_200_000])
costs = np.array([800_000, 850_000, 900_000])
revenue - costs
# array([200000, 250000, 300000])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Broadcasting

- **Allows operations to be performed on array that have different shapes**
- **Smaller arrays are broadcast across larger arrays**
  - The smaller array is "stretched" or repeated to be compatible with the dimensions of the lager array
- **Broadcasting rules**
  - The smaller array with have dimensions of size 1 prepended until it has the same number of dimensions are the larger array
  - The "right-most" dimensions of the arrays must match, if they are not 1
- **Arrays with incompatible dimensions will result in a ValueError**
- **Broadcast leads to efficient vectorised calculations**
  - However, consideration needs to be paid to how the data is organised in memory to optimise performance
  - This is especially true with large arrays (which are the ones where performance is usually an issue)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Broadcasting across an array

```
import numpy as np

a = np.arange(1, 25).reshape(2, 3, -1)
print(a)
# [[[ 1  2  3  4]
#   [ 5  6  7  8]
#   [ 9 10 11 12]]
# ...

b = np.arange(1, 13).reshape(3, -1)
# [[ 1  2  3  4]
# [ 5  6  7  8]
# [ 9 10 11 12]]

print(a * b)
# [[[  1   4   9  16]
#   [ 25  36  49  64]
#   [ 81 100 121 144]]
# ...
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Useful NumPy functions

- **NumPy has many built-in functions that are aggressively optimised**
- **Operate elementwise, returning NumPy arrays**
- **Includes**
  - `all, any, apply_along_axis, argmax, argmin, argsort, average, bincount, ceil, clip, conj, corrcoef, cos, cov, cross, cumprod, cumsum, diff, dot, exp, floor, inner, invert, lexsort, max, maximum, mean, median, min, minimum, nonzero, outer, prod, re, round, sin, sort, std, sum, trace, transpose, var, vdot, vectorize, where`

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Generating random numbers

```
import numpy as np

np.random.uniform(low=0, high=100, size=2)
# array([ 9.39016516, 22.65330985])

np.random.normal(loc=100, scale=16, size=2)
# array([105.42034079,  92.85726491])

np.random.triangular(left=2, mode=5, right=10, size=2)
# array([4.65681506, 7.37800877])

np.random.binomial(n=10, p=0.5, size=5)
# array([7, 4, 6, 5, 5])

np.random.poisson(lam=2.5, size=2)
# array([2, 3])

np.random.beta(a=2, b=5, size=2)
# array([0.10184346, 0.2567474 ])
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Matrix operations

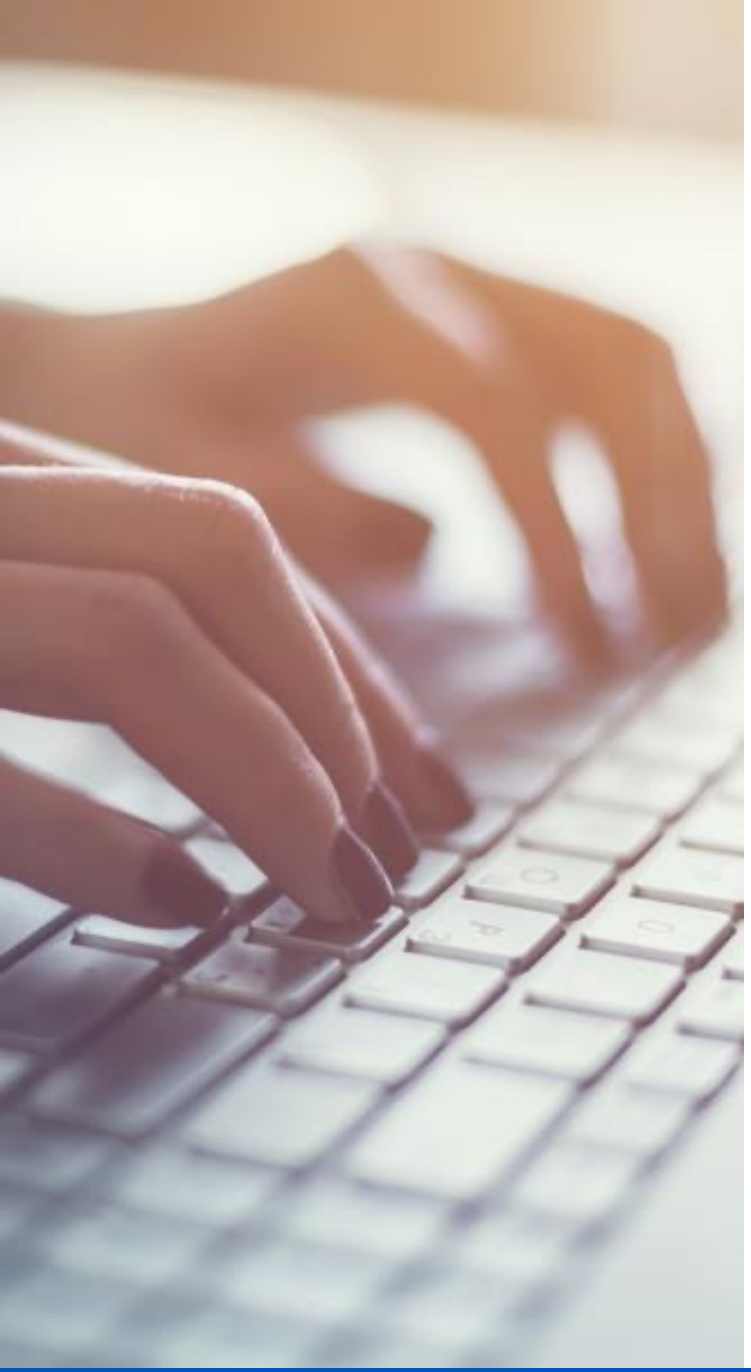
```
import numpy as np

a = np.arange(1, 10).reshape(3, -1)
print(a)
# [[1 2 3]
#  [4 5 6]
#  [7 8 9]]

print(a.T)
# [[1 4 7]
#  [2 5 8]
#  [3 6 9]]

b = np.arange(1, 4)
print(np.matmul(a, b))
[14 32 50]
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Universal functions

```
import numpy as np

def to_fahrenheit(deg_c):
    return deg_c * 9.0 / 5.0 + 32

temperatures = np.array([[0, 50, 100]])

# %timeit is a Jupyter notebook magic function
%timeit to_fahrenheit(temperatures)
# 3.74 µs ± 60.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)

np_to_fahrenheit = np.frompyfunc(to_fahrenheit, 1, 1)

%timeit np_to_fahrenheit(temperatures)
# 1.73 µs ± 27.9 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Perform basic statistical inference using NumPy

Try it for yourself

- **Mensa membership requires scoring in the 98th percentile in a test where the scores are distributed as *N*(100, 16)**
- **Simulate 1m IQ tests and estimate what score is likely to be required if you want to join Mensa**
- **Monte Carlo simulation is a powerful tool for decision-making under uncertainty**
  - NumPy can be used to create complex, efficient simulation models
  - Libraries like PyStan build on NumPy to provide more sophisticated capabilities
- **Solution:**
  - iq_solution.ipynb

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Basic optimisations
- NumPy
- **Numba**
- JAX
- PyPy
- Cython

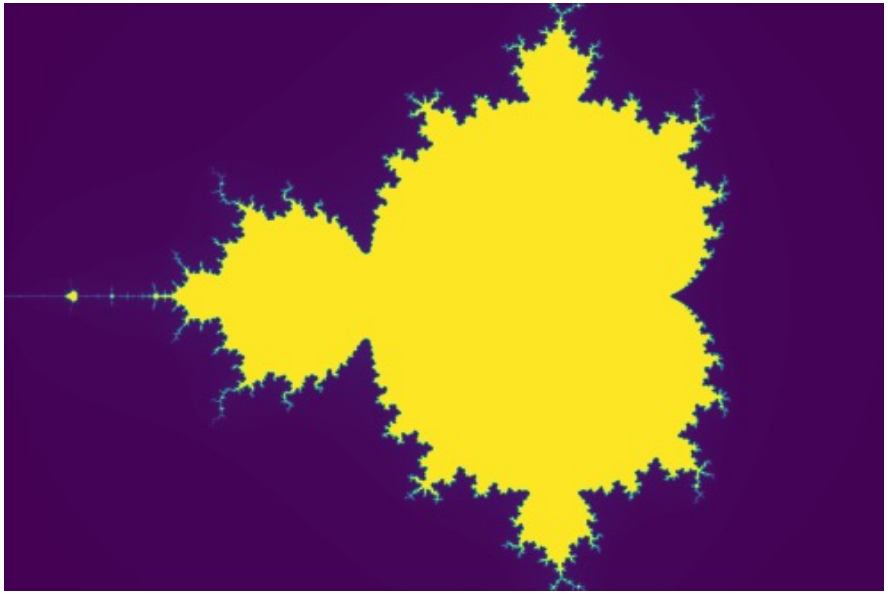
© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Numba

- **JIT compiler for numerical functions in Python**
- **Works best with code that make heavy use of NumPy**
- **Numba is not an *app* compiler**
  - It's a *function* compiler
- **Uses decorators to identify functions to be compiled**
- **Can compile in `nopython` or `object` modes**
  - `nopython` mode is best, but main fail on some functions
- **Supports CPU and GPU compilation**
  - GPU compilation is where the greatest performance opportunities lie
- **pyCUDA tends to produce more efficient GPU code**
  - Requires C programming and code modifications
- **JITing is performed on first call**
  - So only benchmark *subsequent* calls

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Compiling NumPy functions with Numba



```
python mandelbrot.py
# 2nd call to create_fractal executed in 3.11 second(s)

python mandelbrot_jit.py
# 2nd call to create_fractal executed in 0.04 second(s)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Optimising for GPU

```
from numba import vectorize

@vectorize(["int64(int64, int64)"], target="cuda")
def add_gpu(x, y):
    return x + y

add_gpu(a, b)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Basic optimisations
- NumPy
- Numba
- **JAX**
- PyPy
- Cython

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

JAX

- **Google project**
- **JAX is NumPy on the CPU, GPU and TPU**
  - Has support for the automatic differentiation required in ML research
- **Frameworks like Tensorflow and PyTorch are powerful**
  - But display a lack of flexibility
- **Goal is to speed up Python and NumPy functions**
  - JIT
  - GPU
  - Similar objectives to Numba
- **Adopts a functional design**
  - Functions processed by JAX must be pure
  - Mutating updates require special functions
  - Restrictions on Python control flow
- **Uses async computation by default**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Installing JAX

```
pip install "jax[cuda111]"

# Install CPU for development
pip install "jax[cpu]"
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

JAX example

```python
import jax.numpy as jnp
from jax import random
import time

KEY = random.PRNGKey(0)
SIZE = 10000

x = random.normal(KEY, (SIZE, SIZE), dtype=jnp.float64)

start_time = time.perf_counter()

jnp.dot(x, x.T).block_until_ready()

duration = time.perf_counter() - start_time

print(f"JAX took {duration:.2f} second(s) to multiply the matrices")
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Basic optimisations
- NumPy
- Numba
- JAX
- **PyPy**
- Cython

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

PyPy

- **PyPy is an alternative (to CPython) implementation of Python**
  - Highly compatible with existing Python code
- **It's designed with speed in mind**
  - "If you want your code to run faster, you should probably just use PyPy." -- Guido van Rossum
- **Uses a JIT compiler to improve performance**
- **Optimises memory usage**
- **Supports massive concurrency support in Stackless mode**
  - Cheap, lightweight microthreads
- **Versions for Python 2 and Python 3**
- **Can be installed side-by-side with CPython**
- **Performance advantages occur in large, real-world apps**
  - Not just demo code

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Creating PyPy virtual environments

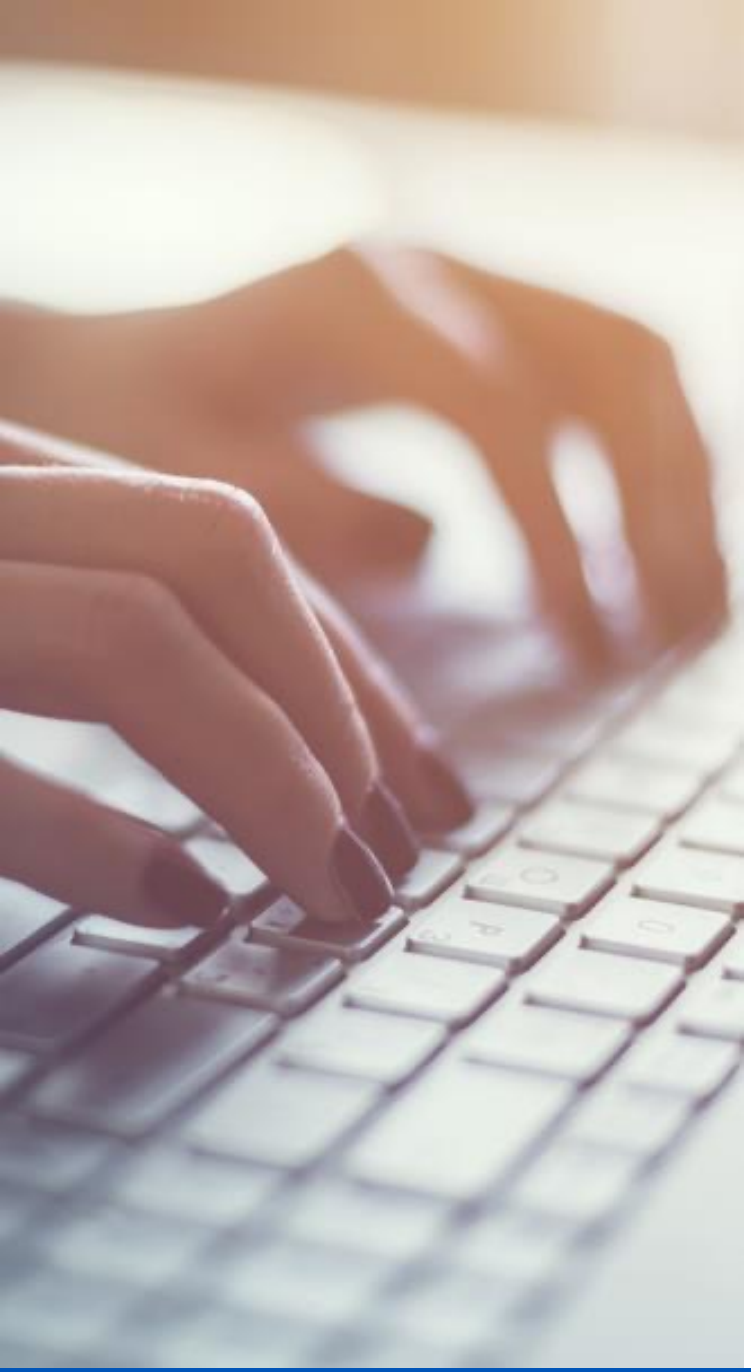
```
virtualenv -p /snap/bin/pypy3 venv
source venv/bin/activate
# deactivate when finished
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

When (not) to use PyPy

- **PyPy offers the most benefit when use with long-running apps that spend time executing Python code**
- **Short-lived scripts will possibly run slower as the JIT compiler won't have time to kick-in**
- **PyPy isn't going to speed up apps that make heavy use of C/C++ libraries such as NumPy**
- **PyPy lags behind CPython in support for new Python features**
  - CPython currently supports Python **3.10**
  - PyPy currently supports Python **3.7**
- **Most packages are developed and tested using CPython**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Compare performance of CPython and PyPy

Try it for yourself

- **Modify the script in nth_prime_benchmarking.py to benchmark `find_nth_prime`**
- **Run the code using CPython**
  - This script doesn't need any packages, so no need for an environment
  - Use python3 <script.py> <args>
- **Run the code using PyPy**
  - Use pypy3 <script.py> <args>
- **Which Python implementation ran the script faster?**
- **Solution:**
  - nth_prime_benchmarking_solution.py

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Contents

- Basic optimisations
- NumPy
- Numba
- JAX
- PyPy
- Cython

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cython

- **Libraries like NumPy and Numba provide performance improvements for special use cases (e.g. numerical programming)**
- **Cython speeds up general-purpose Python code**
  - Turn to it when the special cases don't apply
- **An intermediate step before C and Python**
  - Requires you to annotate your Python code with C type definitions
- **A compilation phase is required to produce the optimised code**
  - Cython code is translated into C and then compiled with LLVM
  - e.g. `cpdef int find_nth_prime(int n):`
- **Compilation is configured via a simple setup.py file**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cython library

primes.pyx

```
import math

cdef int is_prime(int n):
    ...

cpdef int find_nth_prime(int n):
    cdef int count = n

    cdef int candidate = 1

    while count > 0:
        candidate += 1

        if is_prime(candidate):
            count -= 1

    return candidate
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Benchmarking compiled Cython library

```
python nth_prime.py 100000
# Calculated 100000th prime to be 1299709 in 4.52 second(s)

python nth_prime_compiled.py 100000
# Calculated 100000th prime to be 1299709 in 0.38 second(s)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Cython configuration

```
from distutils.core import setup
from Cython.Build import cythonize

setup(ext_modules=cythonize("primes.pyx"))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Compiling Cython libraries

```
pip install cython
python setup.py build_ext --inplace
# Creates a shared library (and .c source)
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Using compiled Cython libraries

setup.py

```
import primes

print(primes.find_nth_prime(100_000))
```

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- It's possible to optimise pure Python code without requiring any third-party assistance
- Huge performance gains are possible by leaning on highly-optimised packages written in C
- GPUs provide an opportunity to further enhance performance of certain classes of apps
- At the expense of increased complexity in development (and support)
- Cython provides an intermediate step between Python and C that can be pre-compiled

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Regular Expressions (optional)

LEARNING TREE™
INTERNATIONAL

Regular expressions

- **A regular expressions (regex) is a pattern (expressed as a string of text) that is matched to input text**
- **Patterns consist of regular characters and metacharacters**
  - Together they create a surprisingly effective grammar for describing and parsing text
- **Regexes can be used for**
  - matching
  - validating
  - replacing
  - splitting
  - extracting
- **They are infamously complex to decipher when they get beyond a certain size**
  - Regexes need rigorous testing

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Basics

. (dot)	match any character (apart from new lines)
^	Match the beginning of the text
$	Match the end of the text
[A-Za-z]	Match any of these characters (including ranges)
[^0-9]	Match any characters except these (including ranges)
cat|dog	Match either of these patterns
(hello)	Match and capture a group of characters
****	Escape a special character

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Character classes

\b	Word boundary
\B	Non-word boundary
\d	Digit
\D	Non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word (alphanumeric and _) character
\W	One non-word character
\A	Start of text (ignores m flag)
\Z	End of text (ignores m flag)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Flags

i	Ignore case
m	^ and $ match start and end of *lines*
s	. (dot) also matches new lines
x	Increase legibility by allowing spaces and comments
L	Make matches locale dependent
u	Match Unicode only
a	Match ASCII only
(?imsxLua)	Set flags within regex

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Special characters

\n	New line
\r	Carriage return
\t	Tab
\xDD	Hexadecimal character code (DD)
\DDD	Octal character code (DDD)

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Quantifiers

*****	Match 0 or more occurrences
+	Match 1 or more occurrences
?	Match 0 or 1 occurrences (i.e. optional)
{n}	Match exactly *n* occurrences
{m,n}	Match from *m* to *n* occurrences
{n,}	Match a minimum of *n* occurrences
{,n}	Match a maximum of *n* occurrences
{m,n}?	Match from *m* to *n* occurrences, but as few as possible

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Groups

(dog)	Match and capture a group of characters
(?P<grp1>dog)	Name a capture group grp1
(?:dog)	Don't capture this group
\3	Match the 3rd captured group
(?P=grp1)	Match the grp1 group
(?#test)	Comment

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Assertions

(?=dog)	Positive lookahead (don't include in match)
(?!dog)	Negative lookahead (don't include in match)
(?<=dog)	Positive lookbehind (don't include in match)
(?<!dog)	Negative lookbehind (don't include in match)
(?(dog)cat|rabbit)	Conditional

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

re module

- **`search(pattern, text)`**
  - Match the first occurrence of `pattern` in `text`
- **`findall(pattern, text)`**
  - Match all occurrences of `pattern` in `text`
- **`sub(pattern, new, text)`**
  - Replace `pattern` with `new` in `text`
- **`split(pattern, text)`**
  - Split `text` into a list using `pattern` as a separator
- **`compile(pattern)`**
  - Create an optimised regex from `pattern` that can be reused efficiently

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

regex101.com

SAVE & SHARE

Save Regex ⌘+s

FLAVOR

- PCRE2 (PHP >=7.3)
- PCRE (PHP <7.3)
- ECMAScript (JavaScript)
- Python
- Golang
- Java 8

FUNCTION

- Match
- Substitution
- List
- Unit Tests

TOOLS

- Code Generator

SPONSOR

Build, deploy and scale apps quickly using App Platform, DigitalOcean's fully managed solution.

Layer0
Jamstack at Scale

REGULAR EXPRESSION

no match

/ insert your regular expression here / gm

TEST STRING

insert your test string here

EXPLANATION

An explanation of your regex will be automatically generated as you type.

MATCH INFORMATION

Detailed match information will be displayed here automatically.

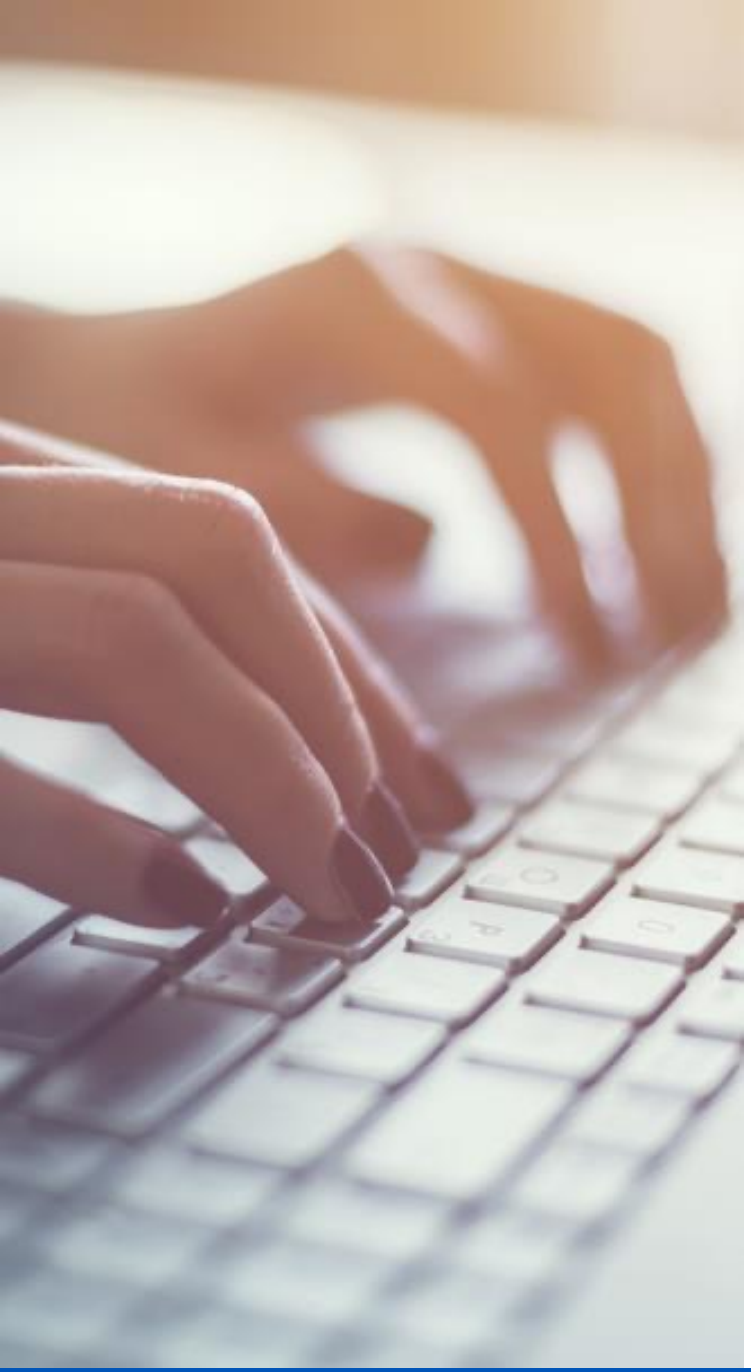
QUICK REFERENCE

Search reference

- All Tokens
- Common Tokens
- General Tokens
- Anchors
- Meta Sequences

A single characte...	[abc]
A character exc...	[^abc]
A character in th...	[a-z]
A character not ...	[^a-z]
A character i...	[a-zA-Z]
Any single character	.

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Manipulate text using regexes

Try it for yourself

- Work through the regular expressions notebook (regular_expressions.ipynb)
- Conduct additional experiments to test your knowledge
- Use regex101.com to test some more ambitions patterns

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Regexes are a very powerful tool for working with freeform text**
- **It's very easy to overdo it and create expressions that are impossible to test or maintain**
- **Start simple and build up**
- **Use tools like regex101.com to develop and test your patterns**
- **Don't be afraid to use standard Python to manipulate your strings**
  - Readable and maintainable trumps concise every time

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Working with Data (optional)

LEARNING TREE INTERNATIONAL

Making decisions with data

- **There are two major steps to communicating with data**
  - Wrangling the data to get the information you need
  - Communicating that information through visualisation

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Data manipulation verbs

Select

Filter

Sort

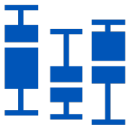
Mutate

Aggregate

Group

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Visualising Relationships



Distribution

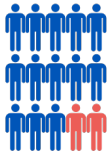
Correlation

Flow

Change over time

Spatial

Ranking



Magnitude

Deviation

Part-to-whole

https://github.com/ft-interactive/chart-doctor/blob/master/visual-vocabulary/Visual-vocabulary.pdf

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Python is the leading platform for data science**
- **Pandas provides powerful capabilities for working with tabular data**
- **Python has a suite of powerful visualisation packages available**
  - Can be difficult to know which package to use

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Dashboards (optional)

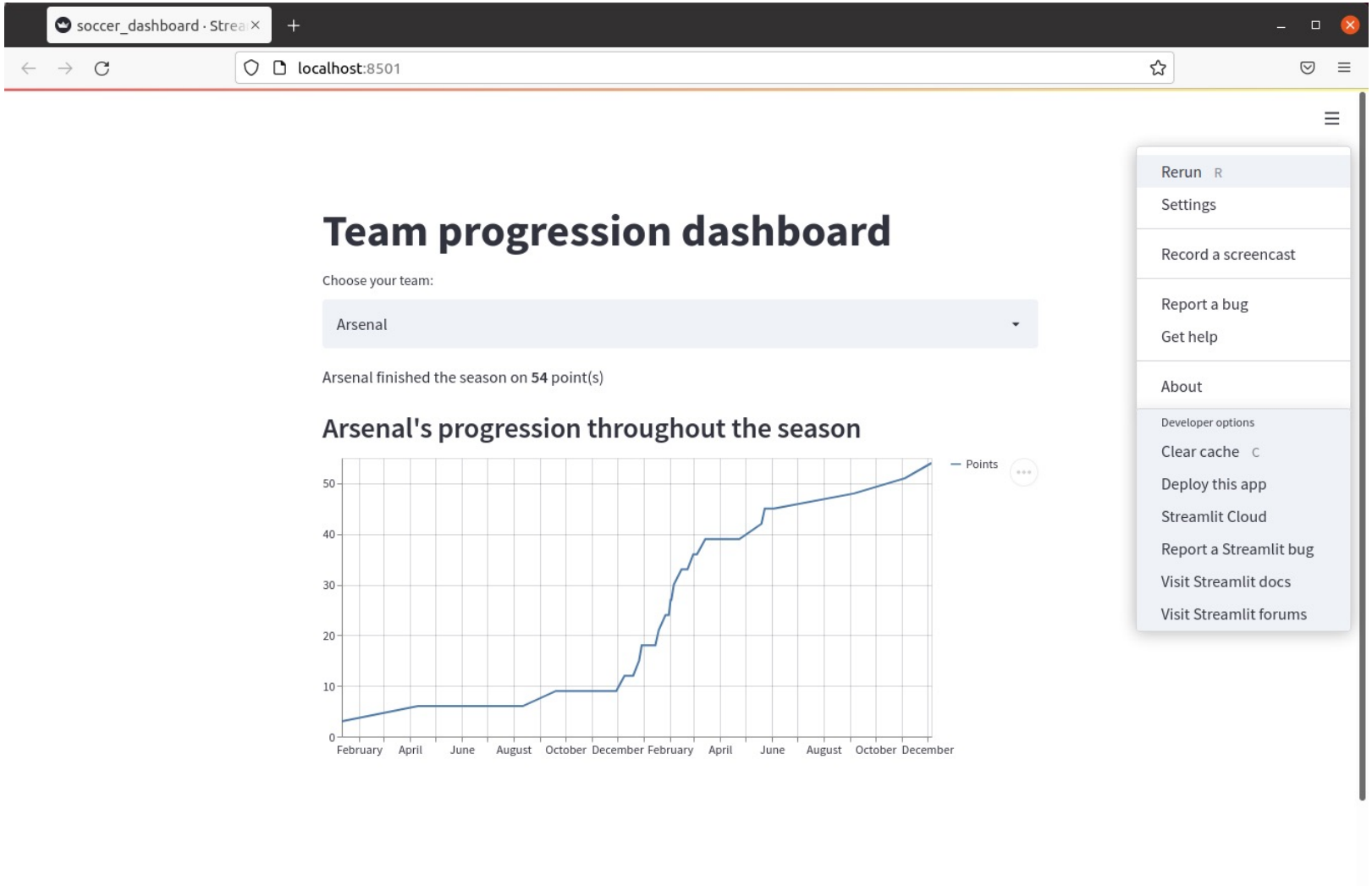
LEARNING TREE™
INTERNATIONAL

Dashboards

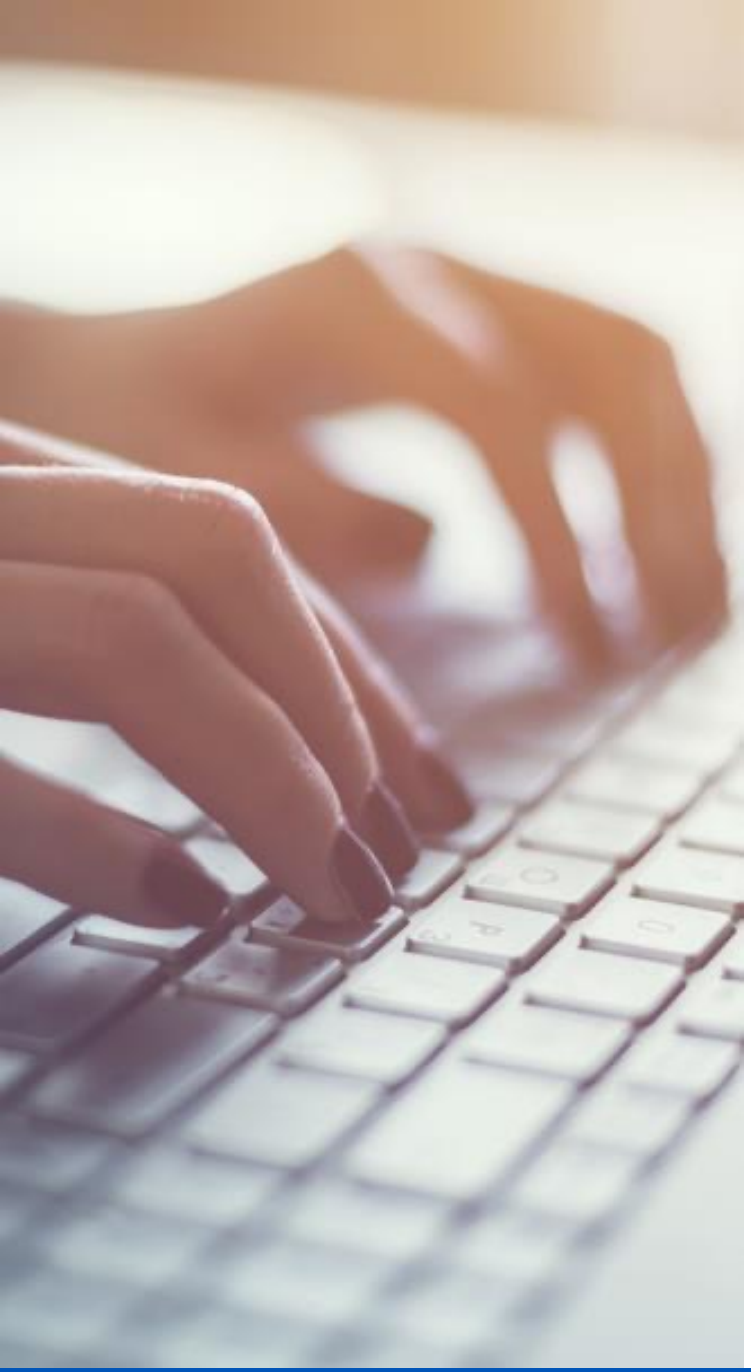
- **Domain expertise is essential to the effective use of data for decision-making**
- **Static reports and charts don't support exploration**
- **Business users don't have the technical skills to work directly with data and models**
- **Streamlit makes it easy to publish interactive apps and dashboards that allow domain experts to contribute to the overall analysis effort**
  - Designed to be lightweight

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Soccer dashboard



© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.



Explore the soccer dashboard

Try it for yourself

- **Launch the dashboard in a web browser**
  - `streamlit run soccer_dashboard.py`
- **Examine the progress of your favourite team**
- **Open soccer_dashboard.py and review the code**
  - Note the use of the `st.cache()` decorated to improve performance
- **Make changes to the code (e.g. the title of the dashboard) and rerun the app**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review

- **Streamlit makes it very easy to publish your work**
  - Especially via Streamlit Cloud
- **Can be extended with custom UI components**
  - Uses React
- **Dash by Plotly is a popular alternative**
  - More flexible
  - Larger ecosystem (for now)
  - More complex

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Review and Next Steps

LEARNING TREE INTERNATIONAL

Review

- **"Advanced" is a subjective term**
- **In this course we've focused on the following major themes**
  - tooling
  - language features
  - testing
  - performance
- **Time-constraints only permitted introductions to most topics**
  - Many of the concepts and libraries covered warrant courses on their own
- **Pick the areas that are most relevant to your work and start drilling down into them**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Next steps

- **The Python ecosystem is *huge***
- **There are many Python communities**
  - Data science
  - Machine learning
  - DevOps
  - Automation
  - Security research
  - Web app development
  - API development
- **Few people span the entire Python ecosystem**
- **Each of the areas has advanced techniques, tools and packages of their own**
- **Explore what the community you are interested in has to offer**

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.

Questions?

© Learning Tree International, Inc. All rights reserved. Not to be reproduced without prior written consent.