import python

-Brian Hodges

Python

- Interpreted scripting language
- Easy to learn
  - print(“Hello World!”)
- No brackets {}
- Dynamically typed
- “Batteries included”
- Builtin help()
- High demand
  - MIT
  - Youtube, Reddit
- import this

Advanced function declarations

- Optional Parameters
  - Parameters with default values that don’t need to be given when calling
- Parameter Packing
  - Lists
    - * Packs positional arguments into a list
  - Dicts
    - ** Packs keyword arguments into a dictionary
- Docstrings
  - First unassigned string in a class/function becomes a docstring, used for documentation
- Typehints
  - Unenforced type declarations for functions and variables
  - import typing

Generators and List Comprehension

- Foreach loops
  - Enhanced for loops in Java
- Generators are special functions that “yield” values instead of returning
  - Can be iterated over using for loops
  - Can make infinite series
- List comprehensions simplify map and filter on lists
  - [x+1 for x in lst] map +1
  - [x for x in lst if x] filter only true elements
- import contextlib

Context managers

- Make “before” and “after” actions easy
  - ex. Opening and closing a file
- Controlled using the “with” statement
  - Anything in the with block occurs in the context
- Can be made from a generator that yields once
  - Uses the @contextmanager decorator
  - The yielded value is the context value
  - Anything before the yield happens when starting the context
  - Anything after the yield happens when the context ends
  - Errors are raised from the yield statement
- Can be a class with __enter__ and __exit__ methods
  - __enter__ is the beginning of the context, the return value is the context value
  - __exit__ is the end of the context, the arguments may be an error that occured
- Can wrap functions
  - Context managers that support it can be used as a decorator, wrapping all of the code inside the function and saving a level of indentation

Classes

- @property
  - Equivalent to getters and setters
- Magic methods
  - Override standard operators
  - __add__ == ‘+’
  - __sub__ == ‘-’
  - __eq__ == ‘==’
  - __str__ == ‘str()’
- __dict__
  - Attribute containing a dictionary of the class’s attributes

Helpful modules

- Builtins
  - collections
  - itertools
  - functools
  - pathlib
  - re
- Pip
  - pytz
  - sqlalchemy
  - sympy
  - numpy
  - scipy
  - requests
  - BeautifulSoup4

Esoterica

- Imaginary numbers
  - 1+1j
- async
  - Builtin asynchronous functions
  - import asyncio

Code samples: functions.py

```
#!/usr/bin/env python3

from typing import List


def tostr(item):
    '''
    Converts the input to a string
    '''
    return str(item)

print(repr(tostr(1)))
input()


def tostr2(item: str) -> str:
    '''
    Converts the input to a string
    '''
    return str(item)

print(repr(tostr2(2)))
input()


def sum(*args: int) -> int:
    '''
    Returns the sum of the arguments
```

Code samples: fibonacci.py

```
#!/usr/bin/env python3


def fibonacci(start=(0, 1)):
    '''
    Generator for an infinite fibonacci series starting with a and b
    '''
    a, b = start
    while True:
        yield a
        a, b = b, a + b

if __name__ == '__main__':
    import time
    for n in fibonacci((1, 1)):
        print(n)
        time.sleep(1)
```

Code samples: files.py

```
import sys

for filename in sys.argv[1:]:
    with open(filename, 'r') as f:
        for line in f:
            print(line)
```

Code samples: context.py

```
import time
from contextlib import contextmanager

@contextmanager
def sqlalchemy_context(Session, autocommit=False):
    session = Session(autocommit=autocommit)
    try:
        yield session
        session.commit()
    except:
        session.rollback()
        raise
    finally:
        session.close()


class SQLAlchemyContext:
    def __init__(self, Session, autocommit=False):
        self.Session = Session
        self.autocommit = autocommit

    def __enter__(self):
        self.session = self.Session(autocommit=self.autocommit)
        return self.session

    def __exit__(self, exc_type, exc_value, traceback):
        if exc_type:
            self.ession.rollback()
```