houses0.py

```python
# Filters out duplicate houses using loop

students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
    {"name": "Padma", "house": "Ravenclaw"},
]

houses = []
for student in students:
    if student["house"] not in houses:
        houses.append(student["house"])

for house in sorted(houses):
    print(house)
```

houses1.py

```python
# Filters out duplicate houses using set

students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
    {"name": "Padma", "house": "Ravenclaw"},
]

houses = set()
for student in students:
    houses.add(student["house"])

for house in sorted(houses):
    print(house)
```

bank0.py

```
# Implements a bank account

balance = 0


def main():
    print("Balance:", balance)


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

bank1.py

```python
# UnboundLocalError

balance = 0


def main():
    print("Balance:", balance)
    deposit(100)
    withdraw(50)
    print("Balance:", balance)


def deposit(n):
    balance += n


def withdraw(n):
    balance -= n


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

bank1.py

bank2.py

```
# Uses global

balance = 0


def main():
    print("Balance:", balance)
    deposit(100)
    withdraw(50)
    print("Balance:", balance)


def deposit(n):
    global balance
    balance += n


def withdraw(n):
    global balance
    balance -= n


```

bank2.py

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

bank3.py

```
# Uses class


class Account:
    def __init__(self):
        self._balance = 0

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

    def deposit(self, n):
        self._balance += n

    def withdraw(self, n):
        self._balance -= n


def main():
    account = Account()
    print("Balance:", account.balance)
    account.deposit(100)
```

bank3.py

```
23      account.withdraw(50)
24      print("Balance:", account.balance)
25
26
27  if __name__ == "__main__":
28      main()
```

meows0.py

```
# Demonstrates a constant

MEOWS = 3

for _ in range(MEOWS):
    print("meow")
```

meows1.py

```
# Demonstrates a class constant


class Cat:
    MEOWS = 3

    def meow(self):
        for _ in range(Cat.MEOWS):
            print("meow")


cat = Cat()
cat.meow()
```

meows2.py

```
# Demonstrates TypeError


def meow(n):
    for _ in range(n):
        print("meow")


number = input("Number: ")
meow(number)
```

meows3.py

```python
# Argument ... has incompatible type


def meow(n: int):
    for _ in range(n):
        print("meow")


number = input("Number: ")
meow(number)
```

meows4.py

```python
# Incompatible types in assignment


def meow(n: int):
    for _ in range(n):
        print("meow")


number: int = input("Number: ")
meow(number)
```

meows5.py

```python
# Success


def meow(n: int):
    for _ in range(n):
        print("meow")


number: int = int(input("Number: "))
meow(number)
```

meows6.py

```python
# Prints None because mistakes meow as having a return value


def meow(n: int):
    for _ in range(n):
        print("meow")


number: int = int(input("Number: "))
meows: str = meow(number)
print(meows)
```

meows7.py

```python
# Annotates return value, ... does not return a value


def meow(n: int) -> None:
    for _ in range(n):
        print("meow")


number: int = int(input("Number: "))
meows: str = meow(number)
print(meows)
```

meows8.py

```python
# Success


def meow(n: int) -> str:
    return "meow\n" * n


number: int = int(input("Number: "))
meows: str = meow(number)
print(meows, end="")
```

meows9.py

```
# Adds docstring to function.


def meow(n):
    """Meow n times."""
    return "meow\n" * n


number = int(input("Number: "))
meows = meow(number)
print(meows, end="")
```

meows10.py

```
# Uses Sphinx docstring format


def meow(n):
    """
    Meow n times.

    :param n: Number of times to meow
    :type n: int
    :raise TypeError: If n is not an int
    :return: A string of n meows, one per line
    :rtype: str
    """
    return "meow\n" * n


number = int(input("Number: "))
meows = meow(number)
print(meows, end="")
```

meows11.py

```python
# Uses command-line argument

import sys

if len(sys.argv) == 1:
    print("meow")
elif len(sys.argv) == 3 and sys.argv[1] == "-n":
    n = int(sys.argv[2])
    for _ in range(n):
        print("meow")
else:
    print("usage: meows11.py [-n NUMBER]")
```

meows12.py

```python
# Uses command-line argument

import argparse

parser = argparse.ArgumentParser()
parser.add_argument("-n")
args = parser.parse_args()

for _ in range(int(args.n)):
    print("meow")
```

meows13.py

```python
# Adds description, help

import argparse

parser = argparse.ArgumentParser(description="Meow like a cat")
parser.add_argument("-n", help="number of times to meow")
args = parser.parse_args()

for _ in range(int(args.n)):
    print("meow")
```

meows14.py

```python
# Adds default, type; removes int()

import argparse

parser = argparse.ArgumentParser(description="Meow like a cat")
parser.add_argument("-n", default=1, help="number of times to meow", type=int)
args = parser.parse_args()

for _ in range(args.n):
    print("meow")
```

unpack0.py

```
# Unpacks a list

first, _ = input("What's your name? ").split(" ")
print(f"hello, {first}")
```

unpack1.py

```python
# Passes positional arguments as usual
# https://harrypotter.fandom.com/wiki/Wizarding_currency


def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


print(total(100, 50, 25), "Knuts")
```

unpack2.py

```
# Indexes into list


def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = [100, 50, 25]

print(total(coins[0], coins[1], coins[2]), "Knuts")
```

unpack3.py

```python
# Unpacks a list


def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = [100, 50, 25]

print(total(*coins), "Knuts")
```

unpack4.py

```
# Passes named arguments as usual


def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


print(total(galleons=100, sickles=50, knuts=25), "Knuts")
```

unpack5.py

```
# Indexes into a dict


def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = {"galleons": 100, "sickles": 50, "knuts": 25}

print(total(coins["galleons"], coins["sickles"], coins["knuts"]), "Knuts")
```

unpack6.py

```python
# Unpacks a dict


def total(galleons, sickles, knuts):
    return (galleons * 17 + sickles) * 29 + knuts


coins = {"galleons": 100, "sickles": 50, "knuts": 25}

print(total(**coins), "Knuts")
```

unpack7.py

```python
# Prints positional arguments


def f(*args, **kwargs):
    print("Positional:", args)


f(100, 50, 25)
```

unpack8.py

```python
# Prints named arguments


def f(*args, **kwargs):
    print("Named:", kwargs)


f(galleons=100, sickles=50, knuts=25)
```

yell0.py

```
# Prints a word in uppercase


def main():
    yell("This is CS50")


def yell(word):
    print(word.upper())


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

yell1.py

```
# Passes a list


def main():
    yell(["This", "is", "CS50"])


def yell(words):
    uppercased = []
    for word in words:
        uppercased.append(word.upper())
    print(*uppercased)


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

yell2.py

```
# Prints arbitrarily many args in uppercase


def main():
    yell("This", "is", "CS50")


def yell(*words):
    uppercased = []
    for word in words:
        uppercased.append(word.upper())
    print(*uppercased)


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

yell3.py

```
# Uses map


def main():
    yell("This", "is", "CS50")


def yell(*words):
    uppercased = map(str.upper, words)
    print(*uppercased)


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

yell4.py

```
# Uses list comprehension


def main():
    yell("This", "is", "CS50")


def yell(*words):
    uppercased = [arg.upper() for arg in words]
    print(*uppercased)


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

gryffindors0.py

```python
# Filters by house using loop

students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]

gryffindors = []
for student in students:
    if student["house"] == "Gryffindor":
        gryffindors.append(student["name"])

for gryffindor in sorted(gryffindors):
    print(gryffindor)
```

gryffindors1.py

```python
# Filters by house using list comprehension

students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]

gryffindors = [
    student["name"] for student in students if student["house"] == "Gryffindor"
]

for gryffindor in sorted(gryffindors):
    print(gryffindor)
```

gryffindors2.py

```python
# Uses filter and key with lambda

students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]


def is_gryffindor(s):
    return s["house"] == "Gryffindor"


gryffindors = filter(is_gryffindor, students)

for gryffindor in sorted(gryffindors, key=lambda s: s["name"]):
    print(gryffindor["name"])
```

gryffindors3.py

```python
# Uses filter with lambda

students = [
    {"name": "Hermione", "house": "Gryffindor"},
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Ron", "house": "Gryffindor"},
    {"name": "Draco", "house": "Slytherin"},
]


gryffindors = filter(lambda s: s["house"] == "Gryffindor", students)

for gryffindor in sorted(gryffindors, key=lambda s: s["name"]):
    print(gryffindor["name"])
```

gryffindors4.py

```python
# Creates list of dicts using loop

students = ["Hermione", "Harry", "Ron"]

gryffindors = []

for student in students:
    gryffindors.append({"name": student, "house": "Gryffindor"})

print(gryffindors)
```

gryffindors5.py

```
# Uses dictionary comprehension instead

students = ["Hermione", "Harry", "Ron"]

gryffindors = [{"name": student, "house": "Gryffindor"} for student in students]

print(gryffindors)
```

gryffindors6.py

```
# Uses dictionary comprehension instead

students = ["Hermione", "Harry", "Ron"]

gryffindors = {student: "Gryffindor" for student in students}

print(gryffindors)
```

gryffindors7.py

```python
# Iterates over a list by index

students = ["Hermione", "Harry", "Ron"]

for i in range(len(students)):
    print(i + 1, students[i])
```

gryffindors8.py

```
# Uses enumerate instead

students = ["Hermione", "Harry", "Ron"]

for i, student in enumerate(students):
    print(i + 1, student)
```

sleep0.py

```python
# Prints n sheep

n = int(input("What's n? "))
for i in range(n):
    print("🐑" * i)
```

sleep1.py

```
# Adds main


def main():
    n = int(input("What's n? "))
    for i in range(n):
        print("\U0001f441" * i)


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

sleep2.py

```python
# Returns n sheep from helper function


def main():
    n = int(input("What's n? "))
    for i in range(n):
        print(sheep(i))


def sheep(n):
    return "🐑" * n


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

sleep3.py

```python
# Returns a list of sheep


def main():
    n = int(input("What's n? "))
    for s in sheep(n):
        print(s)


def sheep(n):
    flock = []
    for i in range(n):
        flock.append("🐑" * i)
    return flock


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

sleep4.py

```python
# Uses yield


def main():
    n = int(input("What's n? "))
    for s in sheep(n):
        print(s)


def sheep(n):
    for i in range(n):
        yield "🐑" * i


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

say.py

```
1  import cowsay
2  import pyttsx3
3
4  engine = pyttsx3.init()
5  this = input("What's this? ")
6  cowsay.cow(this)
7  engine.say(this)
8  engine.runAndWait()
```