WEEK 1 WORKSHOP

# Build Your First Mission Control Panel

A hands-on introduction to Python through a space mission

Dr Lei Wang (Griffith University)

**YOUR MISSION**

You are a software engineer working for a space agency. Build the first version of a Mission Control Panel. It cannot ask questions or make decisions yet, but it can calculate mission information and display it clearly. Each week, the same application will grow as you learn new Python concepts.

This workshop follows the Week 1 learning sequence: print(), numeric values, expressions, arithmetic operators, precedence, type(), and incremental development. Later Mission Control features are reserved for the appropriate later weeks.

## What you will build

The Week 1 dashboard demonstrates the core programming ideas introduced in the lecture. The supplied activity explicitly positions it as the first version of an application that will evolve throughout the semester.

| Dashboard item | Python idea |
|---|---|
| Mission ID | Text displayed with print() |
| Rocket speed | Integer value |
| Mission duration | Integer value |
| Distance travelled | Multiplication |
| Communication delay | Division |
| Fuel tanks required | Floor division // |
| Remaining fuel | Remainder % |
| Solar power units | Exponentiation ** |

| | |
|---|---|
| Operator precedence | Order of calculation |
| Python types | type() |

**THE BIG PICTURE**

The dashboard is deliberately simple in Week 1. Later versions will add user input and variables, decisions, loops and functions. Your goal today is to understand every line, not simply obtain a working screen.

# Learning objectives

- Create and run a Python script.
- Explain REPL versus script.
- Use print() to display information.
- Recognise integer and floating-point values.
- Recognise literals, operators and expressions.
- Use +, -, *, /, //, %, and **.
- Predict results using operator precedence.
- Use type() to inspect a value.
- Read a program from top to bottom and explain it.
- Connect calculations to a real-world task.
- Build a program incrementally.

# Set up your workspace

Create a file named:

```
mission_control.py
```

Run it with Python 3:

```
python3 mission_control.py
```

For quick experiments, use the REPL:

```
python3
>>> 7 + 3 * 2
16
>>> type(7)
<class 'int'>
```

**REPL VS SCRIPT**

REPL = quick experiments. Script = saved program that you can rerun and improve.

**THINK**

Why is it useful to test a tiny expression in the REPL before putting it into a larger program?

# Build the Mission Control Panel - step by step

## Step 1 - Create the dashboard heading

```
print("========================================")
print("       SPACE MISSION CONTROL")
print("========================================")
```

Run it. You have your first dashboard screen.

**THINK**

What changes if you edit the text inside the quotation marks?

**KEY INSIGHT**

print() displays information. Python executes these statements from top to bottom.

## Step 2 - Add mission information

```
print("\nMission ID: 2026-001")
print("\nRocket Speed (km/h):")
print(28000)
print("\nMission Duration (hours):")
print(36)
```

| Code | Meaning |
|---|---|
| print("Rocket Speed...") | Text displayed by Python |
| print(28000) | An integer value |
| print(36) | Another integer value |

**THINK**
Which values above are integer literals?
Hint: Look for numbers written directly in the program.

## Step 3 - Calculate distance travelled

Distance = speed x time. Python uses "*" for multiplication.

```
print("\nDistance Travelled (km):")
print(28000 * 36)
```

**THINK**
Predict 28000 * 36 before running the program.
Hint: Do the arithmetic yourself first.

**PROGRAMMING INSIGHT**
"28000 * 36" is an expression. Python evaluates it to produce a value, which print() then displays.

## Step 4 - Calculate communication delay

Using the simplified mission model: 384,400 km divided by 300,000 km/s.

```
print("\nCommunication Delay (seconds):")
print(384400 / 300000)
```

The "/" operator performs ordinary division. The result is a floating-point value.

**THINK**
What type do you expect from 384400 / 300000?
Hint: Check with type() rather than relying only on intuition.

## Step 5 - Calculate fuel tanks required

Suppose the mission needs 125,000 litres and each tank holds 7,000 litres. We want the number of complete tank-capacity units.

```
print("\nFuel Tanks Required:")
print(125000 // 7000)
```

The “//” operator performs floor division.

**THINK**
Why is // more useful than / for counting complete tank-capacity units?
Hint: Would 17.857 complete tanks make practical sense?

**IMPORTANT**
“/” and “//” answer different questions. Do not treat them as interchangeable.

## Step 6 - Calculate remaining fuel

```
print("\nRemaining Fuel (litres):")
print(125000 % 7000)
```

The “%” operator returns the remainder after division.

**THINK**
What does “%” tell us that “//” does not?
Hint: Think: complete groups versus leftover amount.

**MEMORY TRICK**
“//” -> how many complete groups fit?
“%” -> what is left over?

## Step 7 - Demonstrate exponentiation

```
print("\nSolar Power Units:")
print(2 ** 10)
```

“**” means exponentiation: 2 raised to the power 10.

**THINK**
Predict 2 ** 10 before running the program.
Hint: Think of repeated multiplication.

## Step 8 - Investigate operator precedence

```
print("\nOperator Precedence:")
print(10 + 3 * 2)
print((10 + 3) * 2)
```

Multiplication is evaluated before addition. Parentheses explicitly change the grouping.

| Expression | How to evaluate |
|---|---|
| 10 + 3 * 2 | 3 * 2 first, then + 10 |
| (10 + 3) * 2 | 10 + 3 first, then x 2 |

**THINK**

Predict both results before running them.

Hint: Apply precedence rather than reading strictly left-to-right.

**PROFESSIONAL HABIT**

If an expression could be misunderstood, parentheses can make your intended calculation obvious.

## Step 9 - Inspect Python types

```
print("\nPython Types:")
print(type(28000))
print(type(384400 / 300000))
print(type(-1 * 1.2))
```

| Expression | Expected type |
|---|---|
| 28000 | int |
| 384400 / 300000 | float |
| -1 * 1.2 | float |

**THINK**

Why does -1 * 1.2 produce a float?

Hint: Look at the type of 1.2.

# Your completed Week 1 dashboard

The supplied Week 1 reference program uses the same core calculations and type checks shown below.

```
print("========================================")
print("       SPACE MISSION CONTROL")
print("========================================")

print("\nMission ID: 2026-001")

print("\nRocket Speed (km/h):")
print(28000)

print("\nMission Duration (hours):")
print(36)

print("\nDistance Travelled (km):")
print(28000 * 36)

print("\nCommunication Delay (seconds):")
print(384400 / 300000)

print("\nFuel Tanks Required:")
print(125000 // 7000)

print("\nRemaining Fuel (litres):")
print(125000 % 7000)

print("\nSolar Power Units:")
print(2 ** 10)

print("\nOperator Precedence:")
print(10 + 3 * 2)
print((10 + 3) * 2)

print("\nPython Types:")
print(type(28000))
print(type(384400 / 300000))
print(type(-1 * 1.2))
```

**DO NOT JUST COPY**

After your program runs, change at least three values. Predict how the output should change, then test it. Understanding the code is the real goal.

# Mission challenges

These challenges stay within Week 1 concepts. They are designed to make you think rather than introduce next week's features.

## Challenge 1 - Change the mission

Create your own mission scenario. Change the mission ID, rocket speed, mission duration, total fuel and tank capacity.

- Predict the new distance.
- Predict the new number of complete tanks.
- Predict the new remaining fuel.
- Run the program and compare.

**THINK**
Which outputs change automatically because they depend on the numbers you changed?

## Challenge 2 - Add a new calculation

Add one additional mission calculation using Week 1 operators. For example: another distance calculation, a power calculation, a division, or a remainder calculation.

**THINK**
Which operator best matches your real-world question, and why?

## Challenge 3 - Spot the operator mistake

```
print(125000 / 7000)
```

A teammate says: "This tells us how many complete fuel tanks we need."

**THINK**
Do you agree? If not, which operator should be used and why?

## Challenge 4 - Predict the output

```
print(20 // 6)
print(20 % 6)
print(20 / 6)
print(2 ** 5)
print(10 + 4 * 2)
print((10 + 4) * 2)
```

Write down your predictions first. Then run the code and investigate any differences.

# Debugging checklist

| Check | Ask yourself |
|---|---|
| Syntax | Are brackets and quotation marks correct? |

| | |
|---|---|
| Operator | Did I choose /, //, %, or ** correctly? |
| Value | Did I enter the intended number? |
| Order | Did Python evaluate the expression as I expected? |
| Output | Does the displayed result make sense? |
| Prediction | What did I expect before running it? |
| Evidence | What exactly did Python produce? |

**DEBUGGING MINDSET**
An error is information. Compare what you expected with what actually happened, then find the smallest part of the program that explains the difference.

# Questions to discuss

- Why does a computer need precise instructions?
- Why is print() useful?
- What is the difference between a value and an expression?
- Which dashboard values are integers and which are floats?
- When is // more useful than /?
- When is % useful in a real program?
- Why do the two precedence examples differ?
- What does type() tell us?
- Why build the dashboard incrementally?
- What information would you want the dashboard to ask for next week?

# Week 1 Mission Control checkpoint

Before finishing, make sure you can do each of these without relying on another student's code.

- [ ] Create and run a .py file.
- [ ] Explain REPL versus script.
- [ ] Use print().
- [ ] Identify int and float values.
- [ ] Identify literals, operators and expressions.
- [ ] Use /, //, %, and ** correctly.
- [ ] Predict simple expressions using precedence.
- [ ] Use parentheses to control grouping.
- [ ] Use type().
- [ ] Explain every important line in your dashboard.
- [ ] Change values and test whether results make sense.

# The dashboard will grow

The supplied workshop explicitly plans the dashboard as an evolving application: Week 2 adds input and variables; Week 3 adds if statements; Week 4 adds loops; Week 5 turns calculations into functions.

| Week | New capability | Mission Control evolution |
|---|---|---|
| 1 | Values, operators, expressions, print() | Static mission calculations |
| 2 | Input + variables | User enters mission information |
| 3 | Boolean logic + if | Launch safety decisions |
| 4 | while / for / break / continue | Repeated telemetry processing |
| 5 | Functions | Reusable mission calculations |
| Later | Data structures, files, OOP, modules | More capable application |

**DESIGN PRINCIPLE**

Each new concept should solve a limitation of the previous version.

Ask: "What can my program not do yet?" Then ask: "What new Python idea could make it possible?"

# Final reflection

**Which calculation was most interesting? Why?**

Your answer: _____

**Which operator was easiest to confuse?**

Your answer: _____

**What prediction was wrong, and what did you learn?**

Your answer: _____

**What should the dashboard ask the user next week?**

Your answer: _____

**What mission decision would you like a future version to make?**

Your answer: _____

# Key takeaways

| Remember | Why it matters |
|---|---|

| | |
|---|---|
| Programs follow defined rules. | Computers do not guess your intention. |
| print() displays information. | It lets you see program output. |
| Expressions produce values. | Calculations are building blocks of programs. |
| int and float differ. | Operations can produce different types. |
| / and // differ. | Choose the operator that matches the question. |
| % returns a remainder. | Useful for grouping and divisibility. |
| ** is exponentiation. | It performs powers. |
| Parentheses control grouping. | They also make intent clearer. |
| type() investigates a value. | Use it when unsure about a type. |
| Build incrementally. | Small working steps make larger programs manageable. |
| Predict, run, compare. | This develops real programming skill. |

**MISSION CONTROL PRINCIPLE**

Today you built a small but real program.

Keep your mission_control.py file: it is the foundation for the next version.

# Week 1 technical reference

| Feature | Example |
|---|---|
| print() | print("SPACE MISSION CONTROL") |
| Integer | 28000 |
| Float | 384400 / 300000 |
| Multiplication | 28000 * 36 |
| Division | 384400 / 300000 |
| Floor division | 125000 // 7000 |
| Remainder | 125000 % 7000 |
| Exponentiation | 2 ** 10 |
| Precedence | 10 + 3 * 2 |
| Parentheses | (10 + 3) * 2 |
| Type inspection | type(28000) |