PROGRAMMING FOUNDATIONS FOR ANALYTICS

Pocket Reference: Topics 1 & 2

Everything from Python Basics and Python Lists on one sheet. Keep it open beside your Jupyter notebook — look up the syntax, copy the pattern, change the values.

- Topic 1 · Python Basics
- Topic 2 · Python Lists
- Common mistake — read twice

01 Python Basics

Variables · data types · arithmetic · print & input · errors

1.1 Jupyter Lab in 60 seconds

`Shift + Enter`	Run the cell, move to the next one
`Ctrl + Enter`	Run the cell, stay where you are
`Esc then A / B`	New cell above / below
`Esc then M / Y`	Switch to Markdown / Code cell

A **code cell** runs Python and shows output underneath. A **Markdown cell** holds your notes. The **kernel** is the engine that runs your code.

Stuck? Kernel → Restart Kernel and Clear Outputs, then run your cells again from the top.

1.2 The four data types

TYPE	HOLDS	EXAMPLE
`int`	Whole numbers	`42 -7 1_000_000`
`float`	Numbers with a decimal	`3.14 45.90`
`str`	Text in quotes	`"Kopi-O" 'Yes'`
`bool`	Only two values	`True False`

```
type(3.142)      # float
type("Singapore") # str
type(2_123_456_789) # int
```

Underscores in numbers are only for readability — `1_000_000` is still the number one million.

1.3 Variables

SYNTAX

```
variable_name = value
```

```
item_name  = "Wireless Earbuds"
unit_price = 45.90
quantity   = 2
total = unit_price * quantity  # 91.80
```

DO	DON'T
`price`	`2name` (starts with a digit)
`grab_fare`	`kopi price` (has a space)
`cpf_balance`	`for, if, True` (keywords)

= assigns a value. **==** checks if two values are equal. They are not the same thing.

1.4 Comments

Notes for humans. Python ignores them when it runs.

```
# single line — a note on its own line

price = 12.50  # inline — a note beside the code

"""
multi-line — a block description
at the top of a longer program
"""
```

Use comments to say **why** the code does something, not to repeat what it obviously does.

1.5 Arithmetic & BODMAS

OP	DOES	EXAMPLE
`+ -`	Add, subtract	`5 + 3 → 8`
`*`	Multiply	`6 * 7 → 42`
`/`	Divide (always float)	`7 / 2 → 3.5`
`//`	Floor divide (drops decimal)	`7 // 2 → 3`
`%`	Remainder	`7 % 2 → 1`
`**`	Power	`2 ** 8 → 256`

Brackets → **O**rders (`**`) → **D**ivision & **M**ultiplication (left to right) → **A**ddition & **S**ubtraction.

```
12 + 6 / 2 * 3    # 21.0  (6/2=3.0, 3.0*3=9.0, 12+9.0)
(12 + 6) / 2 * 3  # 27.0  (18/2=9.0, 9.0*3)
```

Add brackets even when you don't need them. Clear beats clever.

1.6 Showing output with print()

SYNTAX

```
print(object1, object2, ...)
```

```
print("Hello World")        # Hello World
print(2 + 3)                # 5
course = "Python Basics"
print("Welcome to", course) # Welcome to Python Basics
```

f-strings drop a variable straight into the text. Put `f` before the quotes and the variable in `{ }`.

```
total = 202.5
print(f"Total sales: ${total:.2f}")
# Total sales: $202.50
```

:.2f forces 2 decimal places — always use it for money.

1.7 Getting input with input()

SYNTAX

```
variable = input("prompt message")
```

input() always returns a str — even when the user types 25. Convert it before doing any maths.

```
name  = input("Your name: ")               # str, no conversion needed
bowls = int(input("Bowls sold: "))         # whole number → int()
price = float(input("Price per bowl: $")) # has decimals → float()

print(f"Total: ${price * bowls:.2f}")
```

NEED	USE
Counts: bowls, days, people	`int()`
Money, weights, temperatures	`float()`
Numbers inside text	`str()`

1.8 Reading error messages

An error is not a failure — it is Python telling you exactly where to look. Read the **last line** first: it names the error type.

ERROR	MEANS	EXAMPLE
`SyntaxError`	Python can't read the code	`print("Hi"`
`NameError`	That name doesn't exist	`print(age)`
`TypeError`	Types don't mix	`"5" + 5`
`ValueError`	Right type, bad value	`int("abc")`

- [ ] **Check the line number** — the problem is on it or just above it.
- [ ] **Count your brackets and quotes** — every one needs a partner.
- [ ] **Check spelling and capitals** — `Name` and `name` are different.
- [ ] **Ask Copilot** — paste the full message and ask it to explain, then verify yourself.

1.9 The five mistakes everyone makes in Topic 1

MISTAKE	WHAT HAPPENS	FIX
Doing maths on `input()` without converting	`"5" + 1` → TypeError, or `"5" * 3` → `"555"`	Wrap it: `int(input(...))`
Using a variable name without quotes as text	`type(Hello)` → NameError	Text needs quotes: `type("Hello")`
Different capitals	`name` defined, `Name` used → NameError	Stay lowercase with underscores
Unclosed quote or bracket	SyntaxError, often pointing at the *next* line	Look at the line above the arrow
Money printed as `91.8`	Looks wrong to a customer	Format it: `f"${total:.2f}"`

02 Python Lists

Create · index · slice · add & remove · copy · analyse

2.1 Creating a list

One name, many values, kept **in the order you typed them**. Square brackets, commas between elements.

SYNTAX

```
my_list = [value1, value2, value3]
```

```
daily_sales = [120, 135, 142, 128, 180, 165, 190]
type(daily_sales)   # <class 'list'>
```

A list can hold anything, even other lists:

```
sizes = [45, 65, 90]                    # all int
flats = ['two_room', 45, 'three_room', 65]  # mixed
table = [['two_room', 45], ['three_room', 65]]  # list of lists
```

2.2 Index — Python counts from 0

Mon	Tue	Wed	Thu	Fri	Sat	Sun
0	1	2	3	4	5	6
120	135	142	128	180	165	190
-7	-6	-5	-4	-3	-2	-1

```
daily_sales[0]    # 120  first (Mon)
daily_sales[2]    # 142  third (Wed)
daily_sales[-1]   # 190  last  (Sun)
daily_sales[-2]   # 165  second last
daily_sales[2+2]  # 180  indexes can be expressions
```

IndexError means you asked for a position that isn't there. A list of 7 items has indexes 0–6, so `daily_sales[7]` fails. Use `-1` for the last item instead of counting.

2.3 Changing an element

Lists are **mutable** — you can overwrite any position.

SYNTAX

```
my_list[index] = new_value
```

```
daily_sales[1] = 145.50   # fix Tuesday
daily_sales
# [120, 145.5, 142, 128, 180, 165, 190]
```

Replace a whole range in one line:

SYNTAX

```
my_list[start:end] = [new1, new2, ...]
```

```
daily_sales[1:4] = [150.30, 143.80, 133.50]
# Tue, Wed, Thu corrected at once
```

2.4 Slicing — take a range

SYNTAX

```
my_list[start : end : step]
```

start is included, end is excluded. `[0:3]` gives you items 0, 1 and 2 — three items, not four.

```
daily_sales[0:3]  # [120, 145.5, 142]  Mon–Wed
daily_sales[:5]   # from the start → Mon–Fri
daily_sales[5:]   # to the end     → Sat–Sun
daily_sales[:]    # the whole list
flats[::2]        # every 2nd item, starting at 0
flats[1::2]       # every 2nd item, starting at 1
```

A slice always gives you back a **new list** — the original is untouched.

2.5 Adding elements

YOU WANT TO ADD…	USE
One value, at the end	`.append(value)`
One value, at a position	`.insert(index, value)`
Many values, at the end	`.extend(other_list)`
Many values, shorter way	`my_list += other_list`

```
daily_sales.append(210)
# [..., 190, 210]

daily_sales.insert(0, 110)      # push to the front
# [110, 120, ...]

daily_sales.extend([162, 187, 186])
daily_sales += [162, 187, 186]  # same thing
```

`.append([1, 2])` adds **one** element that happens to be a list. `.extend([1, 2])` adds **two** elements.

2.6 Removing elements

YOU KNOW…	USE	GIVES BACK
The value	`.remove(value)`	Nothing
The position, need the value	`.pop(index)`	The removed value
The position, don't need it	`del my_list[index]`	Nothing

```
daily_sales.remove(165)    # deletes the value 165
del daily_sales[1]         # deletes whatever is at index 1

monday = daily_sales.pop(0)  # removes AND keeps it
last   = daily_sales.pop()   # no index → the last item
```

`.remove()` only deletes the **first** match. If 165 appears twice, one is left behind.

2.7 Analysing a list

FUNCTION	ANSWERS
`len(my_list)`	How many items?
`sum(my_list)`	What's the total?
`min(my_list)`	What's the lowest?
`max(my_list)`	What's the highest?
`sorted(my_list)`	In order, smallest first

```
print("Days recorded:", len(daily_sales))
print("Weekly total : $", sum(daily_sales))
print("Best day     : $", max(daily_sales))

# average = total ÷ how many
avg = sum(daily_sales) / len(daily_sales)
print(f"Daily average: ${avg:.2f}")

sorted(daily_sales, reverse=True)  # highest first
```

`sum()`, `min()` and `max()` need **numbers only**. A mixed list like `['two_room', 45]` raises a TypeError — slice out the numbers first with `[1::2]`.

2.8 Copy vs reference — the classic trap

`=` does **not** make a copy. It gives the same list a second name, so a change through one name shows up in the other.

```
x = [1, 2, 3]
y = x            # same list, two names
y[1] = 7
print(x)         # [1, 7, 3]  ← changed too!
```

```
x = [1, 2, 3]
y = x.copy()     # a separate list
y[1] = 7
print(x)         # [1, 2, 3]  ← safe
print(y)         # [1, 7, 3]
```

WANT	USE	ORIGINALS
Two names, one list	`b = a`	Linked
An independent copy	`b = a.copy()`	Unchanged
A new combined list	`c = a + b`	Unchanged

03 When to use what

Pick the right tool in one glance

I need to store…

One value that has one meaning	`price = 4.50`
Many values of the same kind	`sales = [4.5, 6.0]`
Rows of related data	`[['Fan', 70], ['Aircon', 1200]]`

Rule of thumb: the moment you find yourself naming `day1`, `day2`, `day3` — stop and use a list.

I need to show a number…

Quick check while coding	`print(total)`
Text plus a value	`print("Total:", total)`
Money or a tidy report	`f"${total:.2f}"`
One decimal place	`f"{avg:.1f}"`

My list is wrong — what now?

One value is wrong	`my_list[i] = new`
A run of values is wrong	`my_list[1:4] = [...]`
Something is missing at the end	`.append()`
Something is missing in the middle	`.insert()`
Something shouldn't be there	`.remove() / del`

Before you ask for help — run this checklist

- [ ] **Did I run the cell?** Editing a cell changes nothing until you press Shift + Enter.
- [ ] **Did I run the cells above it?** A variable defined in a cell you skipped does not exist yet.
- [ ] **Did I convert my input?** Anything from `input()` is text until you say otherwise.
- [ ] **Did I check off-by-one?** First item is index 0; the end of a slice is excluded.
- [ ] **Did I read the last line of the error?** It names the error type and the line number.
- [ ] **Did I ask Copilot to explain, not to solve?** "Explain this TypeError" teaches you more than "fix my code".

Check yourself

1 · What is printed?

```
print(7 // 2, 7 % 2)
```

3 1 — floor division drops the decimal, `%` gives the remainder.

2 · What type is `age`?

```
age = input("Age: ")
```

str — always. `age + 1` would raise a TypeError.

3 · What is the result?

```
12 + 6 / 2 * 3
```

21.0 — division and multiplication happen first, left to right.

4 · What does this give?

```
nums = [10, 20, 30, 40]
nums[1:3]
```

[20, 30] — index 3 is excluded.

5 · What is `x` at the end?

```
x = [1, 2, 3]
y = x
y.append(4)
```

[1, 2, 3, 4] — `y` is the same list, not a copy.

6 · Which line fails?

```
f = ['two_room', 45]
len(f)
sum(f)
```

sum(f) — TypeError, you cannot add text to a number.