Python: variables, operators, basic input/output

Intro. to Computing

`https://mandarm.github.io/computing.html`

Pseudocode to Python

Example: problem 1

Pseudocode

I Initially, set variables X and Y to 0.

II **for** i in $\{1, 2, 3, \ldots, T\}$
$X \longleftarrow X +$ _____.

III **for** j in $\{1, 2, 3, \ldots, D\}$
$Y \longleftarrow Y +$ _____.

IV **If** X _____ Y
then print "*A partially empty reservoir never fills up if all taps and drains are kept open.*"
else print "*An initially empty reservoir fills up in* _____ *minutes if all taps and drains are kept open.*"

V **If** X _____ Y
then print "*A partially full reservoir never becomes empty if all taps and drains are kept open.*"
else print "*An initially full reservoir becomes empty in* _____ *minutes if all taps and drains are kept open.*"

Problem 1: Python equivalent

```
total_water_added_per_min = 0
total_water_removed_per_min = 0

for i in range(num_taps):
    total_water_added_per_min += 1/filling_time

for i in range(num_drains):
    total_water_removed_per_min += 1/draining_time

if total_water_added_per_min == total_water_removed_per_min:
    print("A partially full reservoir never becomes completely full or
     empty when all taps and drains are kept open.")
elif total_water_added_per_min > total_water_removed_per_min:
    print("An initially empty reservoir becomes completely full in",
          1 / (total_water_added_per_min - total_water_removed_per_min),
          "minutes.")
else:
    print("An initially full reservoir becomes completely empty in",
          1 / (total_water_removed_per_min - total_water_added_per_min),
          "minutes.")
```

More meaningful names than X, Y

`range(N)` $\equiv 0, 1, \ldots, N-1$ (more later)

X `+=` $Y \equiv X \leftarrow X + Y$

Later: try replacing comma (`,`) with `+`

Algorithms/pseudocode to programs

What did we need?

- **Variables:** to store values, and to use them in calculations
- **Operators** (and functions): to do addition, etc., and other actions like printing output
- **Statements:** putting variables, operators and functions together to executing actions
  - assignment statements (=, +=)
  - conditional statements (`if`)
  - loops (`for`)

What are we still missing?

- **Input methods:** how do we get the values provided by the user?

Problem 1: missing pieces

Command line arguments

```
import sys

if len(sys.argv) != 3:
    sys.exit('Usage: ' + sys.argv[0] + ' <number of taps> <number of
    drains>')

num_taps = int(sys.argv[1])
num_drains = int(sys.argv[2])
```

Using `input()`

```
    message = 'How long does it take for Tap ' + str(i) + \
        ' to fill an empty reservoir by itself? '
    filling_time = int(input(message))
```

Running Python

Interactively via the (I)Python interpreter

- Run `$ python3`

```
$ python
Python 3.14.6 (main, Jun 15 2026, 11:36:54) [GCC 16.1.1 20260430] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

or

- Run `$ ipython`

```
$ ipython
Python 3.14.6 (main, Jun 15 2026, 11:36:54) [GCC 16.1.1 20260430]
Type 'copyright', 'credits' or 'license' for more information
IPython 9.16.0 - An enhanced Interactive Python.  Type '?'  for help.
Tip:  Use `%timeit` or `%%timeit`, and the `-r`, `-n`, and `-o` options to
easily profile your code.

In [1]:
```

Running “stored” programs

- Write your program in a file and run it

```
$ python <your-file-name> <command-line arguments if any>
```

Python basics

Variables

- Name assigned to memory location or the value stored in it
- Assignment is said to *bind* a name to a value
- Each variable has a *type*
- Variable's type defines operations permitted on that variable
- Same variable (name) can be bound to different values, possibly of different types
  $\Rightarrow$ variables are said to have *dynamic* types

Quick tip

`type()` function returns the type of a variable / expression.

Variables: some terminology

- **binding:** current mapping (if any) for a name to a location
- **l-value:** any expression that corresponds to a location and can thus occur on LHS of assignment
- **r-value:** any expression that can occur on RHS of assignment

All l-values are r-values, but the converse is not true.

```
a = 10       # binds a to the value 10
b = a + 5    # binds b to the value 15
```

- `a` $\rightarrow$ l-value in line 1, r-value in line 2
- `10`, `5` $\rightarrow$ r-values, **not** l-values

Variables: syntax

- Names must start with a letter (uppercase/lowercase, i.e., `A-Z`, `a-z`) or *underscore* `_`.
- Names may contain letters, underscores and digits (`0-9`, anywhere other than in the initial position)
- No upper length limit

Keywords / reserved words: cannot be used as ordinary identifiers

```
False    await      else       import     pass
None     break      except     in         raise
True     class      finally    is         return
and      continue   for        lambda     try
as       def        from       nonlocal   while
assert   del        global     not        with
async    elif       if         or         yield
```

Also `match`, `case`

Basic types

- `int` `783` `0` `-192`
- `float` `1.25` `1.2e10` ($\equiv 1.2 \times 10^{10}$) `1.2E-2` ($\equiv 1.2 \times 10^{-2}$)
- `bool:` only two permitted values `True` `False`
  - `0` and `None` $\equiv$ `False`
- `str` sequence of *characters* (Unicode allowed)
  - delimiters: `"..."` also `'...'` `"""..."""`
  - multiline strings allowed with `"""`, e.g.

```
"""String
across
lines"""
```

- `NoneType` only permitted value `None`

Later: binary, octal and hexadecimal integers

Basic operators: arithmetic operators

- Operators: `+` `-` `*` `/` `//` `%` `**`
  - `/` is division, e.g., `1/2` $\rightarrow$ `0.5`
  - `//` is *integer division* or quotient, e.g., `1//2` $\rightarrow$ `0`
  - `**` is exponentiation, e.g., `2**4` $\rightarrow$ `16`
- **Precedence:** TODO: full table ** higher than * higher than / ????

Basic operators: comparisons and logical operators

- **Comparators** or **relational operators**: `<` `>` `<=` `>=` `==` `!=`

  works for `int`, `float`, `string` types

  Later: the `is` operator

- **Logical** or **Boolean operators**: `and` `or` `not`

Exercises: (remember what you observe!)

1. Try the quotient (`//`) and remainder (`%`) operators with negative numbers.
2. *Lexicographic ordering:* try the comparison operators on strings like `"a"`, `"aaa"`, `"ab"`.

Container types

- Tuples `(11, "y", 7.4)`
- Lists `[ 11, "y", 7.4 ]`
- `str`, `bytes`
- Dictionaries `{ 1:11, 2:"y", 3:7.4, 3.14:"pi" }`
  - elements accessed using *keys*

elements accessed using indices

- Sets `{ 11, "y", 7.4 }` — *not "subscriptable" (indexable)*

Sequences / ordered containers (tuples, lists, strings, bytes)

Indexing

negative indexes

	`-n`	`-(n+1)`	`...`	`-2`	`-1`
`l =`	`[ 10,`	`20,`	`...`	`40,`	`50, ]`
	`0`	`1`	`...`	`n-2`	`n-1`

positive indexes

Slicing: `l[start:end:step]`

- `start`: starting index, *default 0*, included in slice
- `end`: ending index, excluded from slice
- `step`: skip `step` elements (negative step $\Rightarrow$ reverse direction)

More useful functions

- **Lists:** (these functions cannot be applied to dictionaries)
  - `l.index(val)`, `l.count(val)`
  - `+` concatenation
  - `append(value)`
  - `extend(sequence)`
  - `insert(index, value)`
  - `sort()`
  - `reverse()`
- **Containers:**
  - Membership: `in`
  - `len()`, `max()`, `min()`, `sum()`, `sorted()`, `reversed()`

Iterators

- `range(n)`
- `range(start, stop, step)`
- `enumerate(l)`

Assignments

```
a = b = c = 0
```

```
y, z, r  =  9.2, -7.6, 0
```

multiple assignments

```
a, b  =  b, a
```

convenient swap

```
+=    -=    *=    /=    %=
```

Control flow

- `if <condition> :`, `elif <condition> :`, `else :`
- `while <condition> :`, `continue`, `break`
- `for <variable> in <container type> :`
  `for <variable> in <iterator> :`
- `def fname(arg1, arg2, arg3) :`
  (remember to include a `return <value>` or `return None`)

Scope is defined by spaces not braces!

Configure your editor appropriately: use spaces instead of tabs.

Input / output

- Read using `s = input("Prompt string:  ")`
- `s` is always a string
- Convert `s` to appropriate type using `int()`, `float()`, `bool()`, etc.

Misc. issues

Mutable vs. immutable types

- Immutables: `str`, `bytes`, tuples, frozensets
- Mutables: lists, dictionaries, sets

Cheatsheets

- https://perso.limsi.fr/pointal/python:memento
- https://perso.limsi.fr/pointal/python:abrege
- http://ehmatthes.github.io/pcc/cheatsheets/README.html