UNIT I · LESSON 04

Python Programming

data types · operators · control flow · strings · lists · tuples · sets · dictionaries · functions · files · OOP

20
questions in Unit I
C · Python · C++ / OOP

BASICS PYTHON AT A GLANCE

- Created by **Guido van Rossum** (1991); **interpreted**, dynamically typed, high-level
- Source `.py` → **bytecode** `.pyc` → run by **PVM**
- Blocks by **indentation** (usually 4 spaces), no braces
- Comments `#`; docstrings `"""…"""`
- Everything is an **object**; variables are names bound to objects
- `type(x)`, `id(x)`, `isinstance(x, int)`
- Multiple assignment: `a, b = b, a` (swap)

Keywords: `True False None and or not is in def class return lambda yield pass break continue global nonlocal import from as try except finally raise with assert del if elif else for while async await` (35 in 3.x).

BUILT-IN DATA TYPES

Type	Example	Mutable?
int	`10`, `0b101`, `0o17`, `0x1F`	No
float	`3.14`, `2e3`	No
complex	`3+4j`	No
bool	`True`, `False`	No
str	`'hi'`, `"hi"`	**No**
list	`[1, 'a', 2.5]`	**Yes**
tuple	`(1, 2)`, `(5,)`	**No**
set	`{1, 2, 3}`	**Yes**
frozenset	`frozenset({1,2})`	No
dict	`{'a': 1}`	**Yes**
NoneType	`None`	—

- `int` has **unlimited precision**; `bool` is a subclass of int (`True + 1 == 2`)
- Empty set is `set()`, **not** `{}` (that is a dict)

PYTHON-SPECIFIC OPERATORS (CONTD.)

Operator	Example	Result
`**` **power (right assoc.)**	`2 3 2`	512
`is` / `is not`	identity (same object)	—

PYTHON-SPECIFIC OPERATORS

Operator	Example	Result
`/` **true division**	`7 / 2`	3.5
`//` **floor division**	`7 // 2`, `-7 // 2`	3, **-4**
`%` **modulo (sign of divisor)**	`-7 % 3`	**2**

PYTHON-SPECIFIC OPERATORS (CONTD.)

Operator	Example	Result
`in` / `not in`	membership	—
`and` `or` `not`	return an **operand**	`0 or 5` → 5
`:=` **walrus**	assign in expression	3.8+

- Chained comparison: `1 < x < 10`
- No `++` / `--` operators; use `x += 1`
- Bitwise `& | ^ ~ << >>`: `~5 == -6`

IMPORT MODULES & PACKAGES

- `import math`, `from math import sqrt`, `import numpy as np`
- Package = directory with modules (`__init__.py`)
- `if __name__ == "__main__":` runs only when executed directly
- Common: `math, random, os, sys, datetime, re, json, collections, itertools`

IF / LOOPS CONTROL FLOW

```
if m >= 90: g = 'A'
elif m >= 75: g = 'B'
else: g = 'C'
for i in range(1, 10, 2):   # 1 3 5 7 9
    print(i, end=' ')
while n > 0:
    n //= 10
else:
    print('loop ended without break')
```

- `range(stop)`, `range(start, stop, step)` — **stop excluded**
- Loop `else` runs only if the loop **was not broken**
- `pass` = do nothing; `break`, `continue` as in C
- Conditional expression: `x = a if a > b else b`
- `match … case` (3.10) = structural pattern matching

IMMUTABLE STRINGS

- Indexing `s[0]`, negative `s[-1]` (last); slicing `s[start:stop:step]`
- `s[::-1]` reverses; `s[1:4]` → chars 1, 2, 3

IMMUTABLE STRINGS (CONTD.)

- `'ab' * 3` → `'ababab'`; `'a' + 'b'` → `'ab'`
- f-string: `f"{name} scored {m:.2f}"`

IMMUTABLE STRINGS (CONTD.)

Method	Does
upper / lower / title / capitalize	case
strip / lstrip / rstrip	remove whitespace
split(sep) / join(list)	`'a,b'.split(',')`, `'-'.join(['a','b'])`
find / index	position (find → -1, index → error)
replace(a, b)	new string
count(x)	occurrences
startswith / endswith	prefix / suffix test
isdigit / isalpha / isalnum	character tests

`s[0] = 'X'` → **TypeError** (strings are immutable).

MUTABLE SEQUENCE LISTS

Method	Effect
append(x)	add one item at end
extend(it)	add all items of iterable
insert(i, x)	insert at index i
remove(x)	delete first x (ValueError if absent)
pop([i])	remove & return (last by default)
sort() / sorted(l)	in place (returns None) / new list
reverse()	in place
index(x) / count(x)	position / frequency
copy() / l[:]	shallow copy

- List comprehension: `[x*x for x in range(5) if x % 2 == 0]` → `[0, 4, 16]`
- `b = a` copies the **reference**; changes to b show in a
- `del a[1:3]`, `len(a)`, `min`, `max`, `sum`
- `[0] * 3` → `[0, 0, 0]`; `[[0]*3]*2` shares inner lists (trap)

COLLECTIONS TUPLES · SETS · DICTS

- **Tuple**: immutable, faster, can be a dict key; packing `t = 1, 2`; unpacking `a, b = t`
- Single-element tuple needs a comma: `(5,)`
- **Set**: unordered, **no duplicates**; `|` union, `&` intersection, `-` difference, `^` symmetric difference
- `add`, `update`, `remove` (error if absent), `discard` (no error)
- **Dict**: key → value; keys must be **hashable** (immutable); insertion-ordered (3.7+)

COLLECTIONS TUPLES · SETS · DICTS (CONTD.)

```
d = {'a': 1, 'b': 2}
d['c'] = 3
d.get('z', 0)        # 0, no KeyError
for k, v in d.items(): print(k, v)
sq = {x: x*x for x in range(4)}
```

- `keys()`, `values()`, `items()`, `pop(k)`, `update(d2)`, `setdefault`

DEF FUNCTIONS

```
def area(l, b=1, *args, **kw):
    """docstring"""
    return l * b
area(4)            # 4  (default b)
area(b=2, l=3)     # 6  (keyword args)
sq = lambda x: x * x
```

- Default args evaluated **once** (mutable default `[]` is shared — trap)
- `*args` → tuple of extra positional args; `**kwargs` → dict of keyword args
- Returns `None` if no return; can return many values (as a tuple)
- Scope rule **LEGB**: Local → Enclosing → Global → Built-in
- `global x`, `nonlocal x` to rebind outer names
- Built-ins: `map(f, it)`, `filter(f, it)`, `zip(a, b)`, `enumerate(it)`, `reduce` (functools)
- **Generator**: function with `yield` (lazy); **decorator**: `@wrapper`

TRY EXCEPTIONS

```
try:
    x = int(input())
    r = 10 / x
except ValueError:
    print('not a number')
except ZeroDivisionError as e:
    print('error:', e)
else:
    print(r)          # runs if no exception
finally:
    print('always')
```

- `raise ValueError("msg")`; custom: `class MyErr(Exception): pass`
- Common: `IndexError, KeyError, TypeError, NameError, FileNotFoundError`

OPEN FILES

Mode	Meaning
'r'	read (default)
'w'	write, truncate/create
'a'	append
'x'	create, error if exists
'b' / 't'	binary / text
'+'	read and write

```
with open('a.txt', 'r') as f:   # auto close
    for line in f:
        print(line.strip())
```

- `read()`, `readline()`, `readlines()`, `write(s)`, `seek`, `tell`

CLASSES OOP IN PYTHON

```
class Account:
    rate = 0.05              # class var
    def __init__(self, bal):  # constructor
        self.__bal = bal     # "private"
    def deposit(self, x):
        self.__bal += x
    def __str__(self):
        return f"Balance {self.__bal}"
class Savings(Account):    # inheritance
    def deposit(self, x):      # override
        super().deposit(x)
```

- `self` = current object; `__init__` constructor; `__del__` destructor
- `_x` protected (convention), `__x` name-mangled "private"
- Multiple inheritance allowed; **MRO** (C3) decides lookup order
- `@staticmethod`, `@classmethod`, `@property`
- Operator overloading by dunder methods: `__add__`, `__len__`, `__eq__`

Q → A RAPID RECALL

Output of `print(type(3/1))`	`<class 'float'>`
`len({1, 1, 2})`	2
`[1,2,3][::-1]`	`[3, 2, 1]`
`'SIPCOT'[1:4]`	`'IPC'`
`list(map(lambda x: x*2, [1,2]))`	`[2, 4]`
`sum(range(5))`	10
`max('apple')`	`'p'`
`{1,2} & {2,3}`	`{2}`
`dict(zip('ab', [1,2]))`	`{'a': 1, 'b': 2}`
`print(*[1,2,3], sep='-')`	`1-2-3`
`2 ** -1`	0.5

Q → A RAPID RECALL (CONTD.)

`int('101', 2)`	5
Immutable built-ins	int, float, str, tuple, frozenset, bool
Python is ... typed	dynamically **and** strongly
Function to get user input	`input()` (always returns str)

TRAP EXAM TRAPS

- `print(0.1 + 0.2 == 0.3)` → **False**
- `a = [1,2]; b = a; b.append(3)` → a is `[1,2,3]`
- `x = (5)` is an int, `x = (5,)` is a tuple
- `'5' + 5` → TypeError; `'5' * 3` → `'555'`
- `round(2.5)` → **2** (banker's rounding)

TRAP EXAM TRAPS (CONTD.)

- `list.sort()` returns **None**
- `bool('False')` → True (non-empty string)
- `3 // -2` → -2; `-3 % 2` → 1