Data types and data structures

6.3 · Problem solving with programming · revision notes

How a program stores data: the data types integer, real, Boolean, character and string, variables and constants, and the data structures you use in Python (lists for one- and two-dimensional arrays, and records). You also need to manipulate strings: their length, the position of characters, substrings, and upper and lower case.

Key facts

- Integer: whole number (int). Real: number with a decimal part (float)
- Variable: value can change. Constant: value fixed while the program runs
- List indexes start at 0; the last index is len(list) − 1
- Record: related fields of different data types, e.g. ["Kai", 15, True]
- s[a:b] starts at index a and stops before index b
- Boolean: True or False (bool). Character: one symbol. String: text (str)
- Casting: int(), float(), str()
- Two-dimensional list: name[row][column]
- String tools: len(s), s.find(x), s[a:b], s.upper(), s.lower()

Notes

Data types

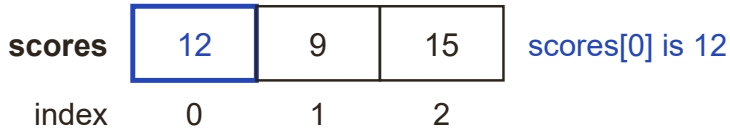
- **Integer** (`int`): a whole number, e.g. `-4` or `250`.
- **Real** (`float`): a number with a decimal part, e.g. `3.75`.
- **Boolean** (`bool`): only `True` or `False`.
- **Character**: a single letter, digit or symbol, e.g. `"Y"`. Python stores it as a string of length 1.
- **String** (`str`): a sequence of characters, e.g. `"CB4 1PQ"`. Phone numbers are strings: they can start with 0 and you never do maths with them.
- **Casting** converts between types: `int("42")` gives 42, `float("2.5")` gives 2.5 and `str(7)` gives `"7"`.

Variables and constants

- A **variable** is a named location in memory whose value can change while the program runs.
- A **constant** holds a value that does not change while the program runs, e.g. `VAT_RATE = 0.2`. Python does not stop you changing it, so constant names are written in capitals to show they should not be changed.
- Constants make code easier to read and maintain: to change the value you edit one line, not every place it is used.

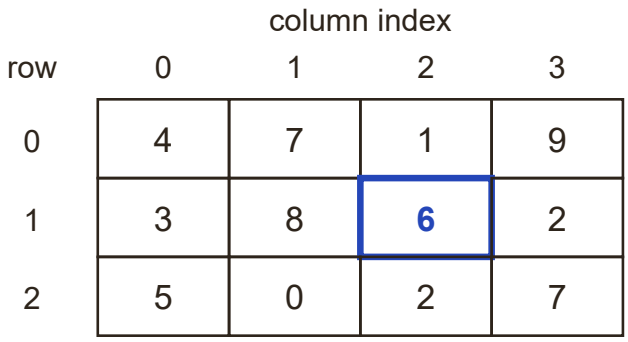
Arrays and records

- An **array** holds many items under one identifier, and each item has an **index** starting at 0. Python uses lists: in `scores = [12, 9, 15]`, `scores[0]` is 12.



Each item in a list has an index, counted from 0.

- `len(scores)` gives the number of items; `scores.append(20)` adds an item to the end; `scores[1] = 10` changes an item.
- A **two-dimensional array** is a list of lists, like a table: `grid[1][2]` is row 1, column 2, both counted from 0.

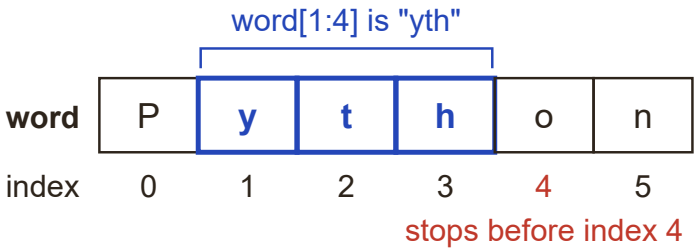


`grid = [[4, 7, 1, 9], [3, 8, 6, 2], [5, 0, 2, 7]]`: the row index comes first, then the column index.

- A **record** groups related fields that can have different data types, e.g. `["Kai", 15, True]` for a name, an age and whether the person is a member. A list of records is two-dimensional: `club[2][0]` is the name in the third record.

Manipulating strings

- **Length**: `len("Python")` is 6.
- **Position**: characters are indexed from 0, so `word[0]` is the first character. `word.find("t")` gives the index of the first `t`, or −1 if there is none.
- **Substring**: `word[1:4]` gives the characters at indexes 1, 2 and 3 (it stops before 4). For `"Python"` this is `"yth"`.



A slice starts at the first index and stops **before** the second.

- **Case conversion**: `word.upper()` gives `"PYTHON"` and `word.lower()` gives `"python"`. The variable only changes if you assign the result: `word = word.upper()`.
- **Concatenation**: `+` joins strings, so convert numbers first: `"Age: " + str(15)`.

How to answer each type of question

State the most suitable data type 1 mark each **Grade 4**

1. Ask what values the data can take: whole numbers, decimals, two options, or text.
2. Anything you would never calculate with, such as a phone number, is a string.
3. Use the names integer, real, Boolean, character and string.

Example. A gym stores data about each member.
State the most suitable data type for each item.
(a) The member's surname (1)
(b) The monthly fee, e.g. 24.99 (1)
(c) Whether the member has paid this month (1)
(d) The number of visits this month (1)

Model answer.

(a) String (1)
(b) Real (float) (1)
(c) Boolean (1)
(d) Integer (1)

Explain why a constant is used

2 marks | Grade 5

1. Say what a constant is: a value that does not change while the program runs.
2. Give a benefit and link it to the program in the question.

Example. A game program uses the constant `MAX_PLAYERS = 4` in several places.
Give **two** reasons why a constant is used instead of typing 4 each time. (2)

Model answer.

If the maximum number of players changes, the value only has to be changed in one place (1).

The name `MAX_PLAYERS` makes the code easier to understand than the number 4 on its own (1).

State the output of string operations

1 mark each | Grade 6

1. Write the string out and number each character from 0.
2. For a slice [a:b], take the characters from index a up to, but not including, index b.
3. Write the output exactly, with the correct upper or lower case.

Example.

```
city = "Cardiff"
```

State the output of each statement.
(a) `print(len(city))` (1)
(b) `print(city[2:5])` (1)
(c) `print(city.upper())` (1)
(d) `print(city.find("f"))` (1)

Model answer.

(a) `7` (1)

(b) `rdi` (1)

(c) `CARDIFF` (1)

(d) `5`: the first `f` is at index 5 (1)

Use a two-dimensional list

3 to 5 marks **Grade 7**

1. Remember the order: row index first, then column index, both from 0.
2. To visit every item, use a loop over the rows with a loop over each row's items inside it.
3. Make the code work for any size: loop over the list itself or use len(), not fixed numbers.

Example. Each row of this list holds the sales of one shop over three days.

```
sales = [[12, 8, 15], [9, 14, 11]]
```

(a) State the value of `sales[1][0]`. (1)
(b) Write code that outputs the total of all the values in `sales`. Your code must work for any number of rows and columns. (3)

Model answer.

(a) 9 (1)

(b)

```
total = 0
for row in sales:
    for value in row:
        total = total + value
print(total)
```

Sets `total` to 0 before the loops (1); nested loops that visit every value in every row (1); adds each value to `total` and outputs it after the loops (1). For this list the output is 69.

Don’t lose marks

- Storing a phone number as an integer, which loses the leading 0.
- Counting indexes from 1. The first item is at index 0.
- Forgetting that input() and data from a file are strings: "5" + "3" is "53", not 8.
- Thinking upper() or lower() changes the variable by itself. You must assign the result.
- Mixing up the row and column indexes in a two-dimensional list.
- Giving 'number' or 'decimal' as a data type. Say integer or real.

More tips

Memory tricks

- If you would never do maths with it (phone number, postcode, ID with leading zeros), store it as a string.
- Slicing [a:b]: a is in, b is out, and the slice has b − a characters.
- Row then column, like reading a table: go down to the row, then along to the column.
- find() gives −1 when the character is not there, a quick way to check whether a string contains it.
- Constant names in CAPITALS, so you spot them straight away.

Exam technique

- When you justify a data type, link it to the data: 'real, because the price has a decimal part'.
- Before working out a slice or an index, write the index of each character above the string.
- Code that processes a list should work for any length: loop over the list or use len(), never a fixed number.
- Keep the order of fields in a record exactly as the question gives it, e.g. name at index 0, age at index 1.

What each grade needs

What you need to be able to do, from the first marks up to the top grade.

Grade 3 — **Name the primitive data types**
Integer, real, Boolean and character, which Python stores as int, float, bool and str.

Grade 4 — **Choose a suitable data type for data**
E.g. real for a price, Boolean for yes or no, string for a phone number.

Grade 4 — **Explain the difference between variables and constants**
A variable's value can change while the program runs; a constant's value cannot.

Grade 5 — **Convert between data types by casting**
Use int(), float() and str() before calculating or joining text.

Grade 5 — **Access and change items in a list**
Use an index that starts at 0, len() to count items and append() to add one.

Grade 6 — **Find length, position, substrings and case**
Use len(), find(), slicing such as word[1:4], upper() and lower().

Grade 7 — **Use a two-dimensional list**
Access an item with name[row][column] and use nested loops to process every item.

Grade 8 — **Store and process records in a list**
Hold each record as an inner list of fields and loop through the records to search or total them.

Free notes, Full packs, practice papers and mock exams at chhetriacademy.com. Written for this site; not taken from past papers. Stuck? 1-to-1 tutoring with a free 30-minute trial lesson.