Subprograms and libraries

6.6 · Problem solving with programming · revision notes

Subprograms let you break a program into named, reusable parts. You need to use built-in and library subprograms (such as len() and random.randint()), write your own functions and procedures, with or without parameters, and explain the difference between local and global variables.

Key facts

- Subprogram: a named block of code for one task, run when it is called
- Parameter: variable in the definition. Argument: value passed in the call
- Built-in: len(), print(), input(), int(), float(), str(), round(), range()
- import math; math.sqrt(), math.floor(), math.ceil()
- Global variable: created in the main program, can be read anywhere
- Function: returns a value with return. Procedure: does not return a value
- def name(parameters): defines a subprogram
- import random; random.randint(a, b) includes both a and b
- Local variable: created in a subprogram, only usable there

Notes

Subprograms

- A **subprogram** is a named block of code that carries out one task. It only runs when it is **called**.
- **Pre-existing** subprograms are already written: **built-in** ones such as `len()`, `print()`, `int()` and `round()`, and **library** ones that you import. **User-devised** subprograms are ones you write.
- Benefits: code is written once and reused, so the program is shorter; each subprogram can be tested on its own; a big problem is decomposed into parts that different programmers can write; the main program is easier to read and maintain.

Functions and procedures

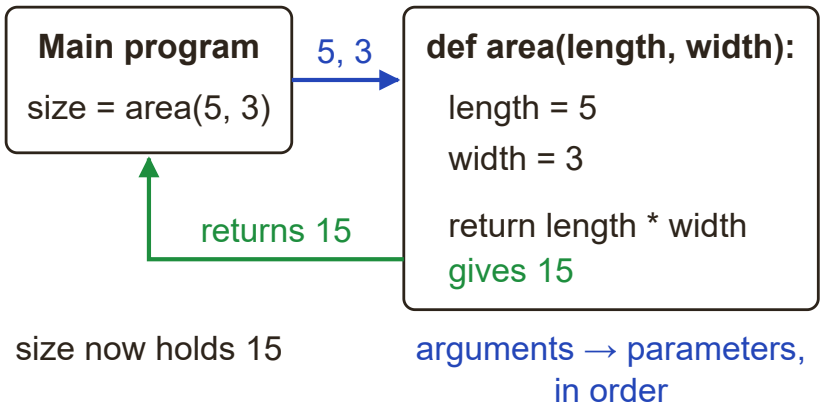
- A **function** returns a value using `return`. A **procedure** carries out a task, such as printing, and does not return a value.
- Both may or may not take **parameters**: the variables in brackets in the definition, which receive values when the subprogram is called. The values passed in are called **arguments**.

```
def area(length, width):
    return length * width

def greet(name):
    print("Hello", name)

size = area(5, 3)
greet("Mo")
```

- `area` is a function: `area(5, 3)` returns 15, which is stored in `size`. `greet` is a procedure: it outputs `Hello Mo` and returns nothing.
- Arguments are matched to parameters in order: `length` receives 5 and `width` receives 3.



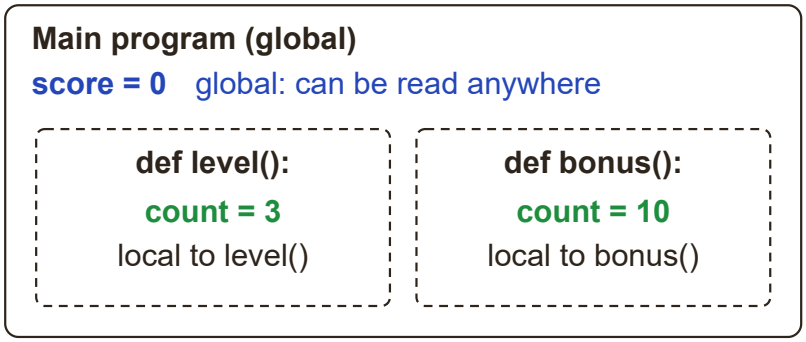
Arguments are passed into the parameters in order; the returned value replaces the call.

Libraries

- Import a library at the start of the program, e.g. `import random` or `import math`.
- `random.randint(1, 6)` gives a random whole number from 1 to 6: both ends are included.
- `math.sqrt(49)` gives 7.0, `math.floor(4.7)` gives 4 and `math.ceil(4.2)` gives 5.
- Library subprograms save time and have already been tested, so they are reliable.

Local and global variables

- A **local variable** is created inside a subprogram. It can only be used inside that subprogram and only exists while the subprogram is running. Parameters are local too.
- A **global variable** is created in the main program, outside any subprogram, and can be read anywhere in the program.



The two local variables called `count` are separate and only exist while their subprogram runs.

- Local variables are better practice: the same name can be used in different subprograms without clashing, the memory is freed when the subprogram ends, and each subprogram is self-contained, so it is easier to test and reuse. Pass data in with parameters and out with `return`. Grade 7+ only
- If a subprogram assigns a value to a variable, Python treats it as a new local variable, even when a global variable has the same name, so the global variable does not change. Grade 8+ only

How to answer each type of question

Functions, procedures and benefits 2 to 4 marks **Grade 4**

1. For the difference, say what each one does: a function returns a value, a procedure does not.
2. For a benefit, give the benefit and a reason, e.g. 'can be reused, so the code only has to be written and tested once'.

Example. (a) State the difference between a function and a procedure. (2)
(b) Give **two** benefits of using subprograms in a program. (2)

Model answer.

(a) A function returns a value to the part of the program that called it (1); a procedure carries out a task but does not return a value (1).

(b) Any two, e.g. the code can be reused without writing it again (1); each subprogram can be tested on its own, so errors are easier to find (1); different programmers can write different subprograms at the same time; the program is easier to read and maintain.

Use a library subprogram

2 to 4 marks | **Grade 5**

1. Import the library at the start of the program.
2. Write the library name, a dot, then the subprogram, e.g. random.randint().
3. Remember that randint includes both of its end values.

Example. (a) Write a program that simulates rolling two six-sided dice and outputs the total of the two dice. (3)
(b) State the smallest and the largest total your program could output. (1)

Model answer.

(a)

```
import random
die1 = random.randint(1, 6)
die2 = random.randint(1, 6)
print("Total:", die1 + die2)
```

Imports the random library (1); generates two random whole numbers from 1 to 6 (1); outputs their total (1).

(b) 2 and 12 (1)

Trace calls to a function

2 to 4 marks | **Grade 5**

1. Match each argument to its parameter, in order.
2. Work through the function body and find the value that is returned.
3. Put the returned value back where the function was called, then carry on.

Example.

```
01 def calc(a, b):
02     result = a * 2 - b
03     return result
04
05 x = calc(6, 4)
06 y = calc(x, 3)
07 print(x, y)
```

(a) State the output of line 07. (2)
(b) Identify **one** local variable in the function `calc`. (1)

Model answer.

(a) `8` (1) `13` (1), output as `8 13`

(b) `result` (1). Also accept `a` or `b`: parameters are local variables.

Write a function

3 to 5 marks **Grade 7**

1. Start with def, the exact name from the question and the parameters in the order given.
2. Calculate the result using the parameters, not input().
3. Use return to send the result back. Do not print it unless the question asks you to.

Example. Write a function `discount(price, percent)` that returns the price after a percentage discount has been taken off. For example, `discount(40, 25)` returns 30.0. (4)

Model answer.

```
def discount(price, percent):
    newPrice = price - price * percent / 100
    return newPrice
```

A `def` line with the name and the two parameters in the correct order (1); works out the amount of the discount, `price * percent / 100` (1); subtracts it from the price (1); returns the result rather than printing it (1).

Explain local and global variables

3 to 4 marks **Grade 8**

1. State the output by tracing the code.
2. Say where each variable is created: inside the subprogram (local) or in the main program (global).
3. Explain that assigning to a name inside a subprogram creates a separate local variable, so the global one is unchanged.

Example.

```
01 total = 100
02
03 def reset():
04     total = 0
05     print("In reset:", total)
06
07 reset()
08 print("Main:", total)
```

State the output of the program and explain why the two values of `total` are different. (4)

Model answer.

Output: `In reset: 0` then `Main: 100` (1).

The `total` in line 04 is a local variable, created inside `reset` (1). It is a different variable from the global `total` created in line 01 (1). Changing the local variable does not change the global one, so line 08 still outputs 100 (1).

Don’t lose marks

- Using print() instead of return in a function, so the program that called it gets no value back.
- Calling a function and not using the result, e.g. writing area(5, 3) on its own line instead of size = area(5, 3).
- Forgetting import random, or writing randint(1, 6) without random. in front: a runtime error.
- Passing arguments in the wrong order, so each parameter receives the wrong value.
- Thinking a subprogram changes a global variable just by assigning to a variable with the same name.
- Defining a subprogram but never calling it, so its code never runs.

More tips

Memory tricks

- A function is like a calculator: it hands back an answer. A procedure is like a printer: it does a job and hands nothing back.
- Parameters are placeholders in the definition; arguments are the actual values in the call.
- randint(1, 6) is a normal dice: both ends are included.
- Local variables live and die inside their subprogram.

Exam technique

- When asked to write a function, include def, the parameters named in the question and a return statement.
- Use the subprogram name and parameters exactly as given in the question, in the same order.
- For benefits of subprograms, give a reason, not just a word: 'reused, so it only has to be written and tested once'.
- When explaining scope, say where the variable is created and where it can be used.

What each grade needs

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

Grade 3 — **Call built-in subprograms such as len()**
Use pre-existing subprograms such as len(), print(), int() and round() with the right arguments.

Grade 4 — **Tell functions and procedures apart**
A function returns a value; a procedure carries out a task and does not return a value.

Grade 5 — **Import and use library subprograms**
E.g. import random, then use random.randint(1, 6) for a dice roll.

Grade 5 — **Give benefits of using subprograms**
Code can be reused, tested on its own and is easier to read and maintain.

Grade 6 — **Write a procedure that takes parameters**
Define it with def, pass values in when you call it, and let it carry out its task.

Grade 7 — **Write a function that returns a value**
Take parameters, calculate a result, return it and use the returned value in the main program.

Grade 8 — **Explain local and global variables**
Say where each variable can be used and why local variables are better practice.

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.