Computing

Lesson 5: Structured Programming

Programming Part 4: Subroutines

Ben Garside

[1] *Materials from the Teach Computing Curriculum created by the National Centre for Computing Education*

Task 1 - Improve the code - worked example

Code before

```
1 to_guess = 3
2 not_guessed = True
3 while not_guessed:
4     print("Guess a number")
5     number = int(input())
6     if number == to_guess:
7         break
```

Code after

```
1 to_guess = 3
2 not_guessed = True
3 while not_guessed:
4     print("Guess a number")
5     number = int(input())
6     if number == to_guess:
7         not_guessed = False
```

Task 1 - Improve the code - instructions

Improve the code that you will find over the next three slides.

- Copy and paste the ‘before’ code into your development environment.
- Edit the code so that the block only has 1 entry point and 1 exit point.
- Paste the improved code into the ‘after’ box

Note: The program should perform in exactly the same when when executed

Tip: Use the worked example on the previous slide as a guide

Task 1 - Improve the code - 1

Code before

```
def and_function(a, b):
    if a == True and b == True:
        return True
    else:
        return False


one = 4 == 4
two = 2 == 2

print(and_function(one, two))
```

Code after (your solution)

```

```


Task 1 - Improve the code - 2

Code before

```
def multiple_five(number):
    if number % 5 == 0:
        return "Multiple of 5"
    else:
        return "Not a multiple of 5"

print(multiple_five(12))
```

Code after (your solution)

```

```

Task 1 - Improve the code - 3

Code before

```
def password_check():
    password = "12345"
    entered_pass = ""
    pass_not_valid = password != entered_pass
    attempts = 0
    while pass_not_valid and attempts < 3:
        print("Enter a password:")
        entered_pass = input()
        attempts = attempts + 1
        if password == entered_pass:
            break
    if password != entered_pass:
        return "Access Denied"
    else:
        return "Access Granted"


print(password_check())
```

Place your code on the next slide

Task 1 - Improve the code - 3

Code solution

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```


Task 2 - Structure chart - instructions

The next slide contains an incomplete structure chart for the dog walking weekly invoice program. Decide on the interface requirements for each subroutine.

Program description

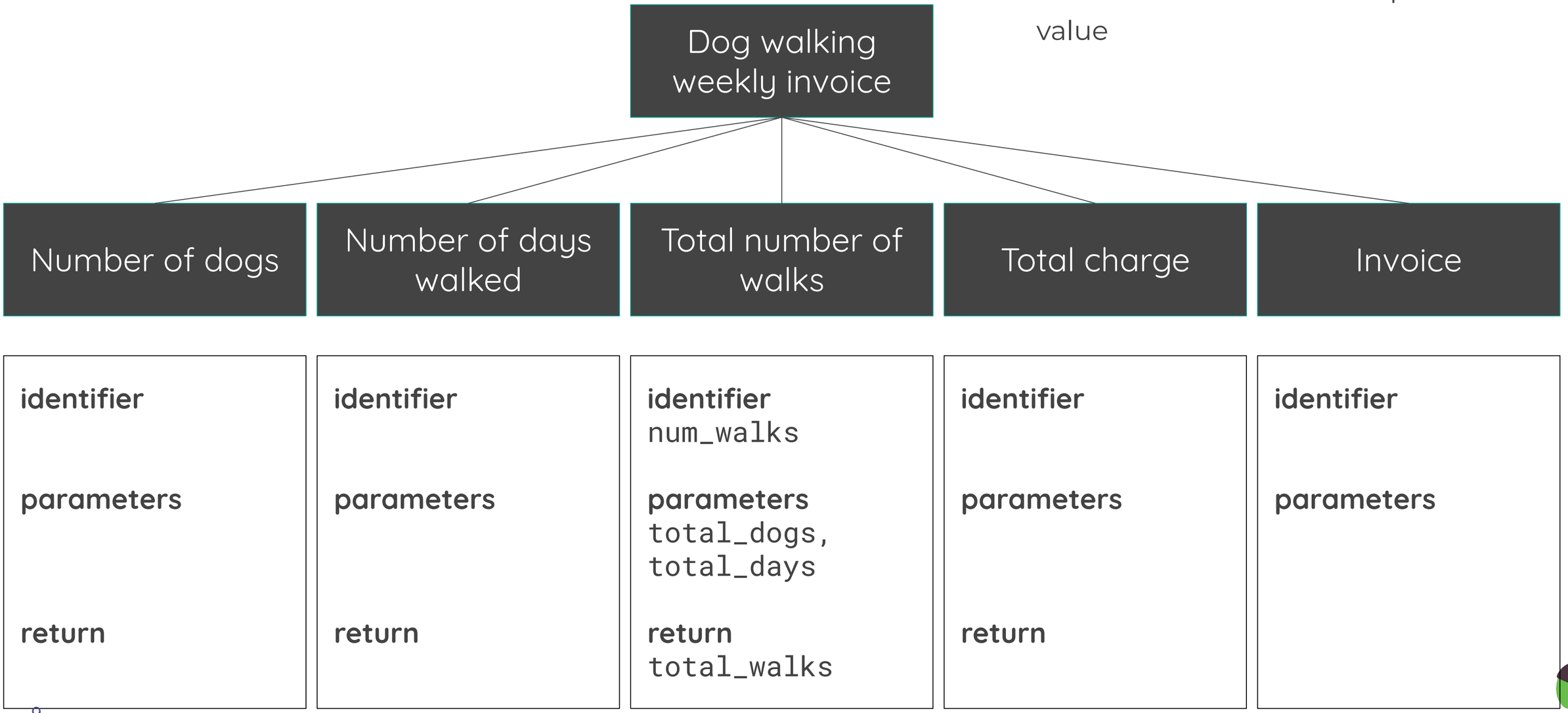
A dog walker would like a program that provides a weekly invoice for their clients based on the number of dogs, the number of walks and the cost per walk. The program should allow the user to:

- Enter the number of dogs the client has
- Enter the number of days they have walked the dogs
- Calculate the number of walks based on **number of dogs x number of days**
- Calculate the total cost based on **number of walks x 4.00**
- Display the relevant invoice information: Number of dogs, number of days, total number of walks, total cost

Structure chart: solution

Important points

- Not all subroutines will require parameters
- Not all subroutines will require a return value



Task 3 - Complete the program

Open the partially completed code here (oaknat.uk/comp-ks4-dogstart) or copy and paste the code below into your development environment.

```
def num_dogs():

  return total_dogs

def num_days():

  return total_days

def num_walks(total_dogs, total_days):

  return total_walks

def total_charge(total_walks):

  return total_cost

def invoice(total_dogs, total_days, total_walks, total_cost):

total_dogs = num_dogs()
total_days = num_days()
total_walks = num_walks(total_dogs,total_days)
total_cost = total_charge(total_walks)
invoice(total_dogs, total_days, total_walks, total_cost)
```


Task 3 - testing the program

If your program works correctly then the following input and output should perform as expected.

Example: (✔ if it was successful) Note: Use this example to check your program. This is the output your program should produce for the given input.		✔
A message is displayed to prompt the user	`Number of dogs for this client:`	
The user enters a value	`3`	
A message is displayed to prompt the user	`How many days has the dog been walked?`	
The user enters a value	`4`	

The program calculates the total cost and displays the invoice to the user	`Number of dogs: 3`
	`Number of days walked: 4`
	`Total number of walks: 12`
	`Total cost: 48.0`