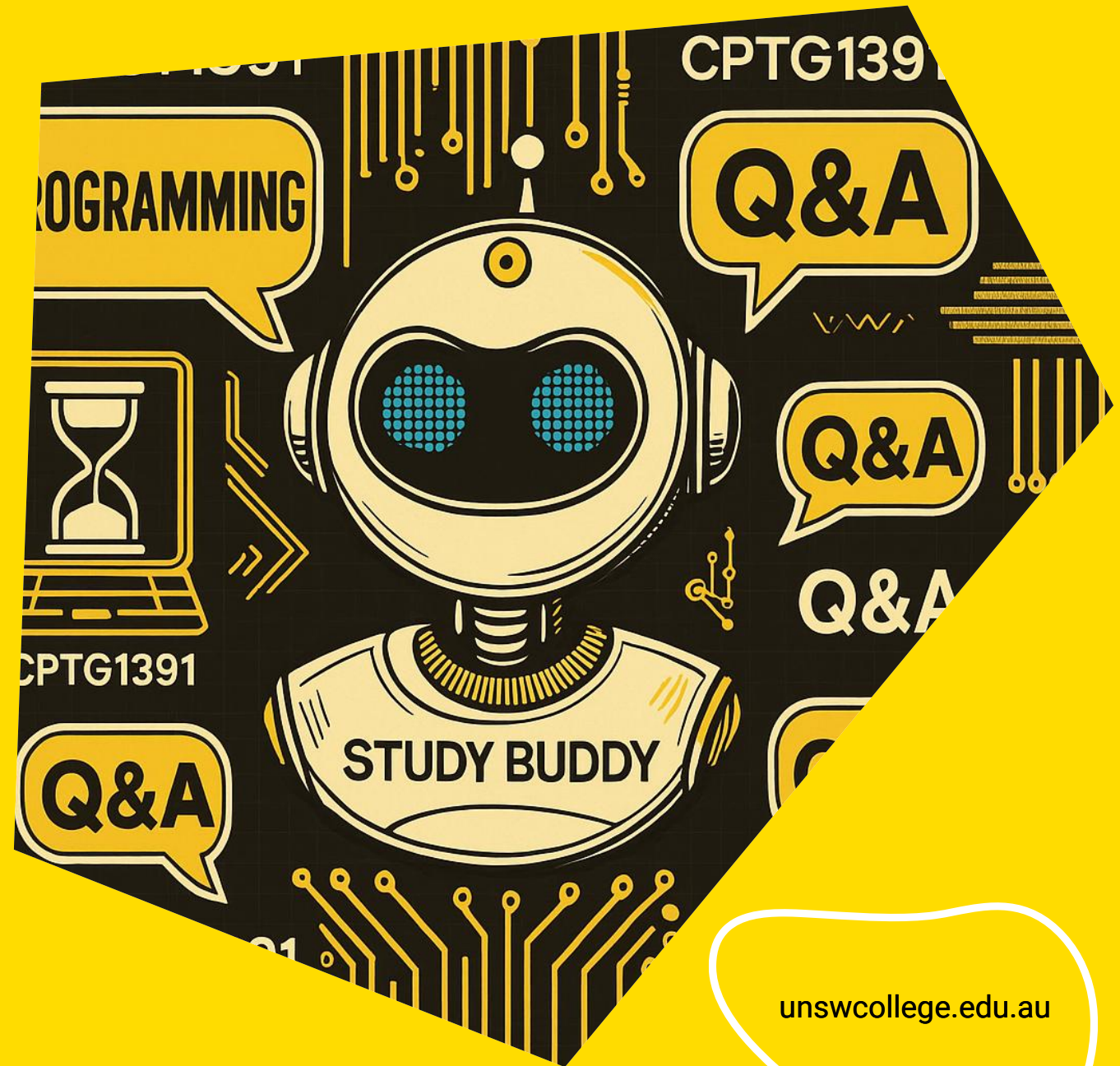


DPST1091 / CPTG1391
Introduction to Programming
Week 3 – Lecture 1

Lecturer and Course Convener:

Dr Pantea Aria

Functions



Agenda

- **Last lecture**

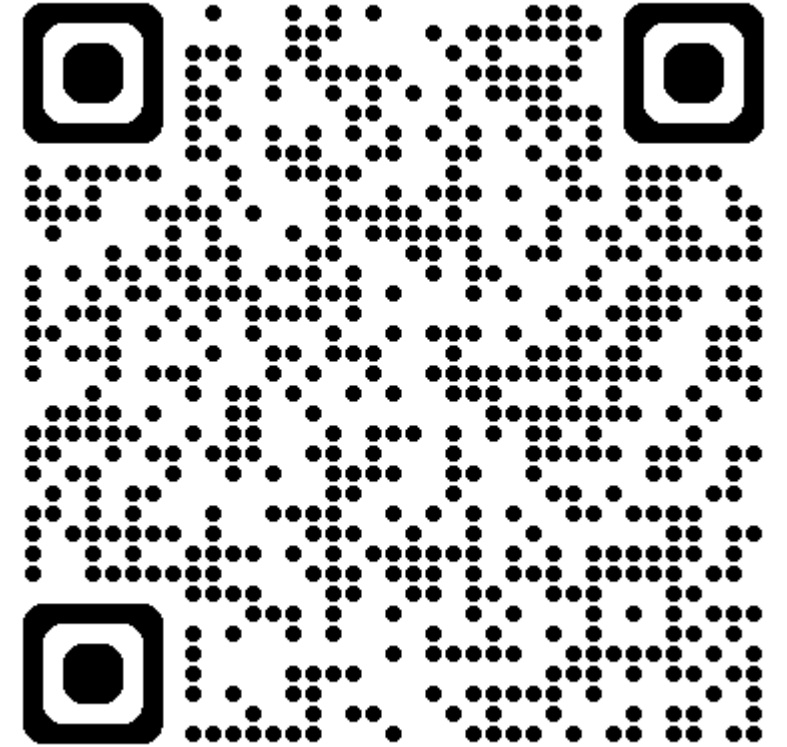
- Nested Loops
- Custom Data Types

- **Today**

- Functions

Link to the Live Lecture Code

<https://cgi.cse.unsw.edu.au/~dp1091/26T2/code/03>



Live lecture code is written for teaching, not perfection.
It may include extra comments and may not always follow
ideal coding style.

Nested Loops recap

→ A **while loop** inside another **while loop** is called a **nested loop**.

→ This structure is very useful for processing **two-dimensional data**, such as **grids** and **matrices**.

	Column →	1	2	3	4	5
Row 1		1	2	3	4	5
Row 2		1	2	3	4	5
Row 3		1	2	3	4	5
Row 4		1	2	3	4	5
Row 5		1	2	3	4	5

Think of the grid like a **table** made of **rows** and **columns**.

→ **Rows** go **across** (left to right).

→ **Columns** go **down** (top to bottom).

Full program:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

```
1#include <stdio.h>
2
3#define ROWS 5
4#define COLUMNS 5
5
6int main() {
7    int i = 0;
8
9    while (i < ROWS) {
10        int j = 1;
11        while (j <= COLUMNS) {
12            printf("%d ", j);
13            j++;
14        }
15        printf("\n");
16        i++;
17    }
18
19    return 0;
20}
```

Demo

→Nested_loops_recap.c

Structs recap

- A **struct** is a user-defined data type that groups different kinds of data together in a single unit.
- Each field in a struct has its **own space in memory** and can be accessed individually.
- Structs are useful when you want to store information as a **well-organised record**, such as details about a person, a student, or a product.

```
struct product {  
    int id;  
    double price;  
};
```

Enums recap

→ **enum** (enumeration) is a user-defined data type in C that consists of **named integer constants**.

→ Gives **meaningful names to numbers**, improving code readability.

```
enum Weekday { MONDAY,  
                TUESDAY,  
                WEDNESDAY,  
                THURSDAY,  
                FRIDAY  
};
```

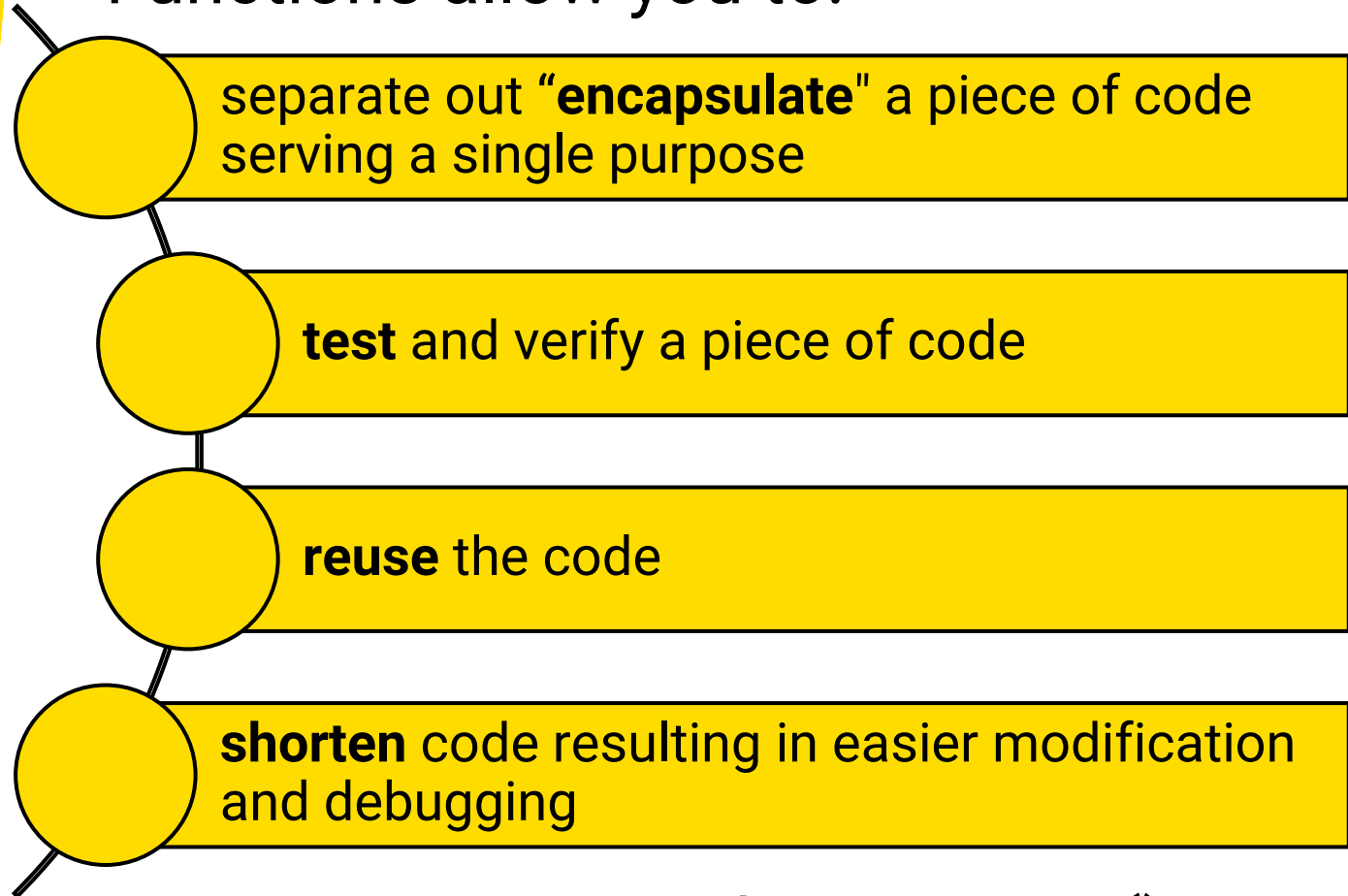


Demo

- enum_coffee_recap.c
- Struct_enum_book_recap.c

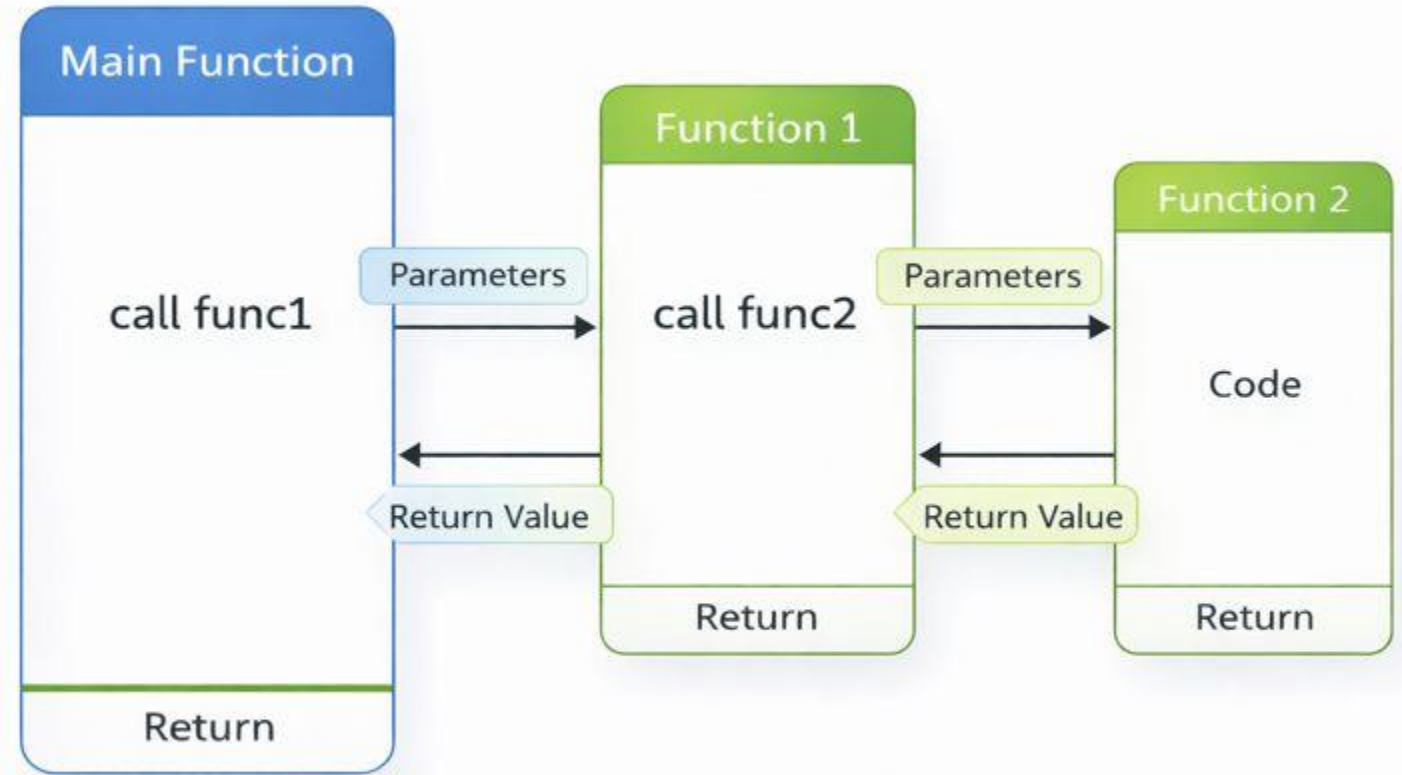
Functions

Functions allow you to:



Functions we already use: *main()*, *printf()*, *scanf()*

How it works in a simple diagram

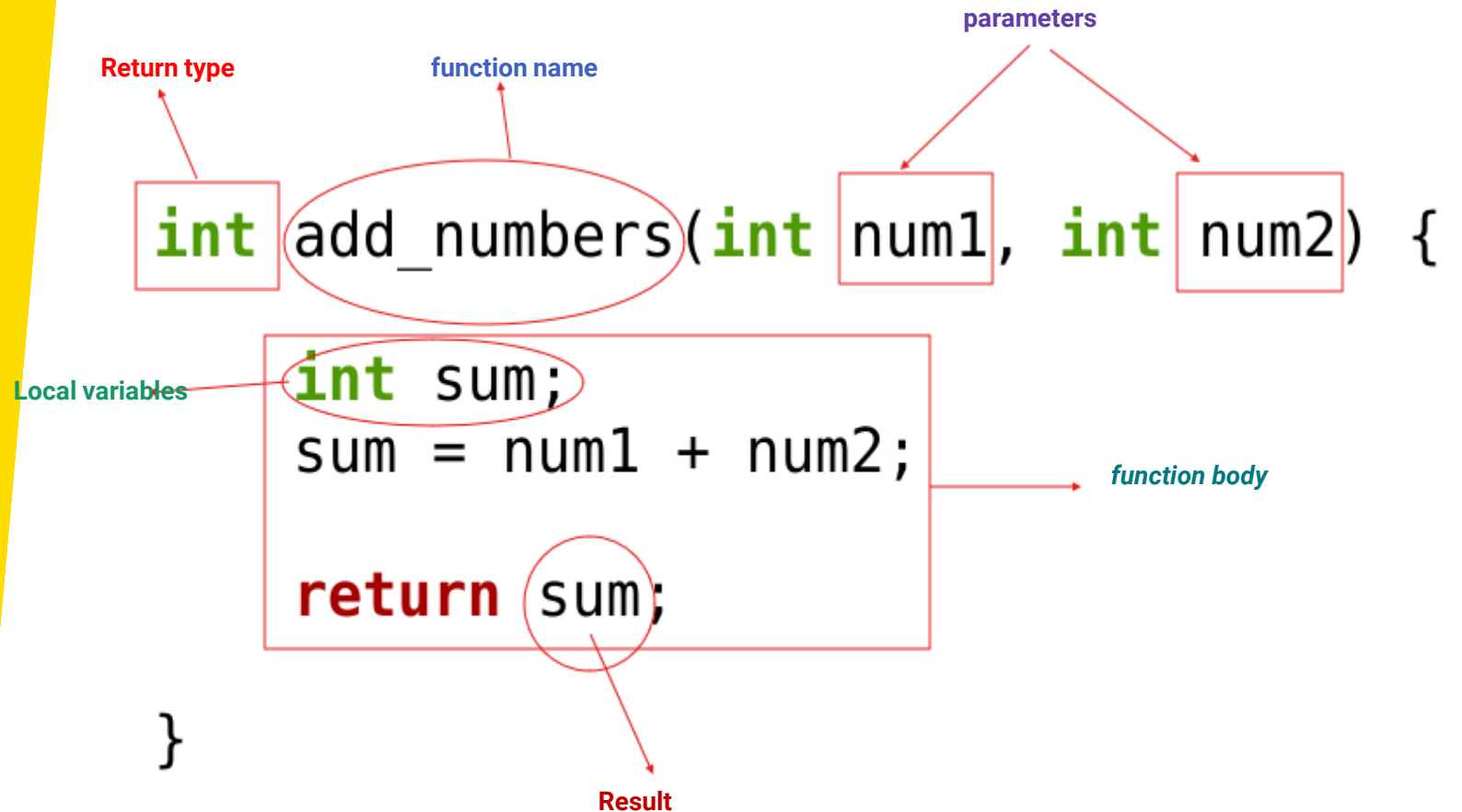


Demo

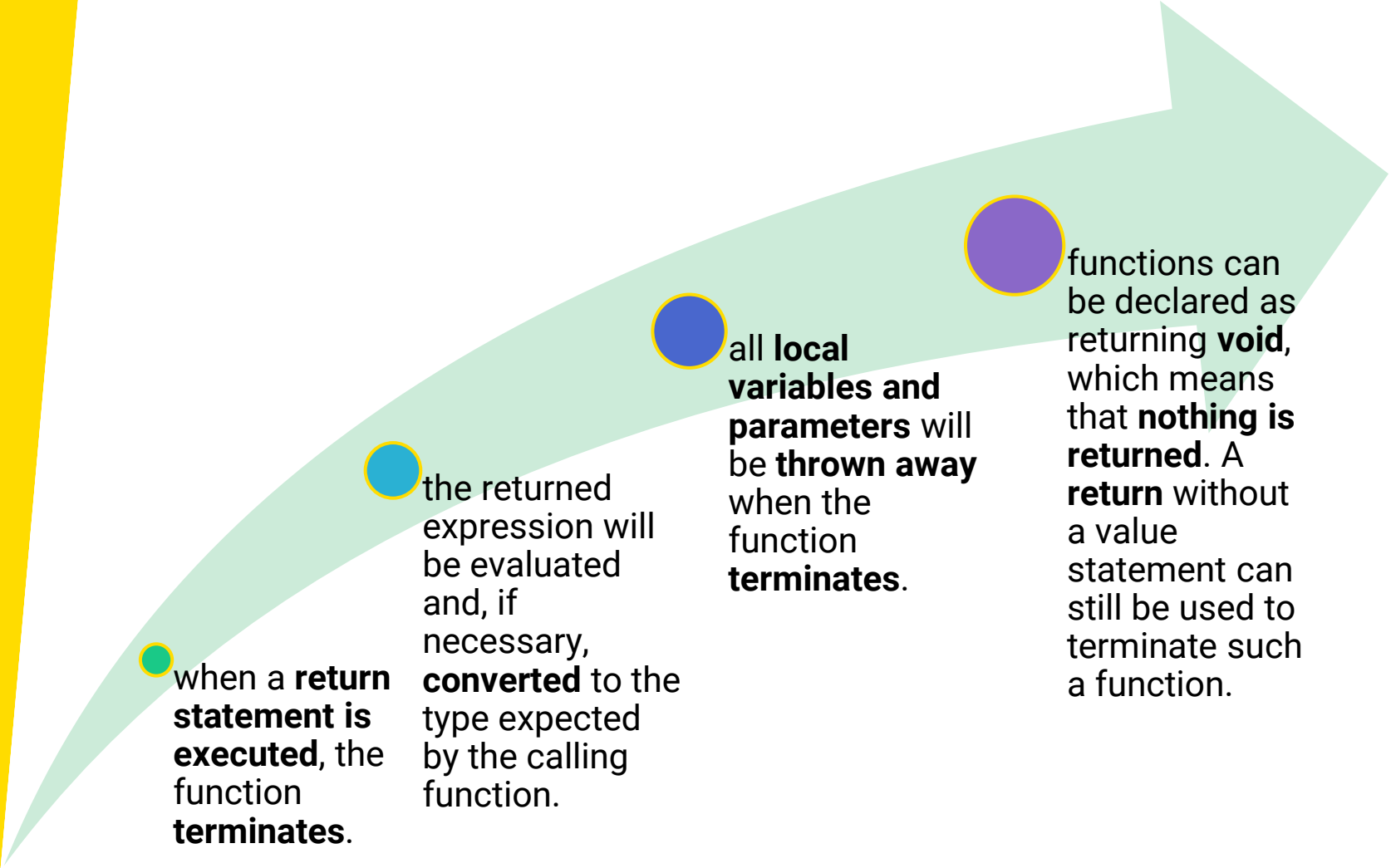
→function_hello.c

Structure of a Function

1. Return type
2. Function name
3. Parameters (inside brackets, comma separated) – input
4. Variables (only accessible within the body of function)
5. Return statement – output/results



The return Statement



when a **return statement is executed**, the function **terminates**.

the returned expression will be evaluated and, if necessary, **converted** to the type expected by the calling function.

all **local variables and parameters** will be **thrown away** when the function **terminates**.

functions can be declared as returning **void**, which means that **nothing is returned**. A **return** without a value statement can still be used to terminate such a function.

Function Properties

function have a **type** - the type of the value they **return**

type **void** for functions that return **no value**

function can not return **arrays** (we will see later)

function have their own **variables** created when function called and **destroyed** when function returns

return statement **stops** execution of a function

return statement specifies value to return, unless function is of type **void**

run-time error if end of **non-void function** reached **without return**

A full Program:

```
1 #include <stdio.h>
2
3 // Function prototype
4 double cube(double x);
5
6 int main(void) {
7     double a, b;
8
9     printf("42 cubed is %lf\n", cube(42.03));
10
11     a = 2;
12     b = cube(a);
13
14     printf("2 cubed is %lf\n", b);
15
16     return 0;
17 }
18
19 // calculate x to the power of 3
20 double cube(double x) {
21     double result;
22     result = x * x * x;
23     return result;
24 }
```

Calling function cube

Functions with No Return Value (called procedures too)

Some functions do not to compute a value.
They are useful for "**side-effects**" such as output.

```
void print_sign(int b) {  
    if (b < 0) {  
        printf("negative");  
    } else if (b == 0) {  
        printf("zero");  
    } else {  
        printf("positive");  
    }  
}
```

Demo

→function_void.c

Function Parameters

- functions take **0 or more** parameters
- parameters are variables **created each time function** called and destroyed when function returns
- parameters initialized with the value **supplied by the caller**
- if parameters **variables changed** in the function **has no effect outside the function**

```
1 #include <stdio.h>
2
3 // Function prototype
4 void change_values(int x, int y);
5
6 int main(void) {
7     int a = 5;
8     int b = 10;
9
10    printf("Before function call: a = %d, b = %d\n", a, b);
11
12    // Call-by-value: copies of a and b are passed
13    change_values(a, b);
14
15    // Original variables are unchanged
16    printf("After function call: a = %d, b = %d\n", a, b);
17
18    return 0;
19 }
20
21 // Function definition (after main)
22 void change_values(int x, int y) {
23     // Parameters are created when the function is called
24
25     x = x + 10;
26     y = y + 20;
27
28     printf("Inside function: x = %d, y = %d\n", x, y);
29
30     // Parameters are destroyed when the function returns
31 }
```

Function Prototypes

Specifies key information about the function:

- ▶ function return type
- ▶ function name
- ▶ number and type of function parameters

Allows top-down order of functions in file More readable
Allows us to have function definition in separate file.

Example prototypes:

```
double power(double x, int n);  
void print_sign(int b);
```

Demo

→function_add.c

Library functions

- Over 700 function are defined in the C standard library.
- You'll need to use less than 20 of these in this course.
- The C compiler needs to see a prototype for these functions before you use them.
- You do this indirectly with **#include** line
- For example, **stdio.h** contains prototypes for **printf** and **scanf**

```
#include <stdio.h>
```

```
int main(void) {
```

```
    printf("I love Programming!\n");
```

```
}
```

Example:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(void) {
5     double number = 9.0;
6     double result1, result2;
7
8     // Using sqrt() from math.h
9     result1 = sqrt(number);
10
11    // Using pow() from math.h
12    result2 = pow(number, 2);
13
14    printf("Square root of %.2f is %.2f\n", number, result1);
15    printf("%.2f squared is %.2f\n", number, result2);
16
17    return 0;
18 }
```

structs and functions

A structure can be passed as a parameter to a function:

```
void print_student(student s) {  
    printf("%s z%d\n", s.name, s.zid);  
}
```

A function can **return a structure**:

```
student read_student(int zid, double marks) {  
    student s;  
    s.zid = zid;  
    s.marks = marks;  
    return s;  
}
```

Demo

→function_struct.c

What did we learn today?

Topic	Summary	Example
What is a Function?	A function is a reusable block of code that performs a specific task. It helps organise programs into smaller, manageable parts.	<code>printf(), scanf(), main()</code>
Why Use Functions?	• Reuse code • Reduce repetition • Make programs easier to debug • Improve readability • Test parts independently	<code>add_numbers()</code>
Function Structure	A function consists of: 1. Return type 2. Function name 3. Parameters 4. Local variables 5. Function body 6. Return statement	<code>int add(int a, int b) { int sum = a + b; return sum; }</code>
Return Type	Specifies the type of value returned by the function. Use <code>void</code> if nothing is returned.	<code>int, double, void</code>
Parameters	Input values passed to a function. A function can have zero or more parameters.	<code>int add(int x, int y)</code>
Return Statement	Ends the function and sends a value back to the caller.	<code>return sum;</code>
Void Functions	Functions that do not return a value. Used for tasks such as printing output.	<code>void print_sign(int n)</code>
Function Call	Invokes a function and optionally stores its returned value.	<code>result = cube(3);</code>
Local Variables	Variables declared inside a function. They exist only while the function is running.	<code>int sum;</code>
Call by Value	passes copies of arguments to functions. Changes to parameters do not affect original variables.(for now)	<code>change_values(a, b);</code>
Function Prototype	Declares a function before it is used, telling the compiler its return type, name, and parameters.	<code>double cube(double x);</code>
Library Functions	Built-in functions provided by header files.	<code>sqrt(), pow(), printf()</code>
Functions and Structs	Structs can be passed to functions and returned from functions.	<code>student read_student(...);</code>

Voice of the Student

Anonymous ongoing feedback
Anything you wanted to share with me



UNSW
College

Dr Pantea Aria

Functions

Voice of the Student



[Voice of the Student – Fill out form](#)

See you soon ...