CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

http://www.cit.ie

Several Parameters and value-returning functions

Making Changes to Parameters

- Changes made to a parameter value within the function do not affect the argument
  - Known as *pass by value*
  - Provides a way for unidirectional communication between one function and another function
    - Data flows from the calling function to the called function

Like a photocopy

If you photocopy a page from a book and scribble on the photocopied page, the original remains unchanged.

The relationship between the argument and parameter is the same

1. The parameter is a copy of the argument
2. Changes make to the parameter (our 'photocopy') will not affect the argument

1_unidirectional.py - changing the parameter does not affect the argument


```
What is the child's name? Miriam
Happy Birthday to you
Happy Birthday to you
Happy Birthday dear Miriam
Happy Birthday to you!
Miriam
```

```
def sing(name):
    print("Happy Birthday to you")
    print("Happy Birthday to you")
    print("Happy Birthday dear", name)
    print("Happy Birthday to you!")
    name = "Mickey Mouse"


def main():
    childs_name = input("What is the child's name? ")
    sing(childs_name)
    # lest there be any doubt,
    # childs_name has not become "Mickey Mouse"!
    print(childs_name)

main()
```

What if they have the same name?

2_unidirectional.py

If the arguments and parameters share the same name, it makes no difference – the parameter is still a copy of the argument and changes made to the parameter do not affect the argument.

```
def sing(name):
    print("Happy Birthday to you")
    print("Happy Birthday to you")
    print("Happy Birthday dear", name)
    print("Happy Birthday to you!")
    name = "Mickey Mouse"


def main():
    name = input("What is the child's name?")
    sing(name)
    # lest there be any doubt, name has not become "Mickey Mouse"!
    print(name)

main()
```

```
What is the child's name? Miriam
Happy Birthday to you
Happy Birthday to you
Happy Birthday dear Miriam
Happy Birthday to you!
Miriam
```

More than one parameter

Old MacDonald had a farm…

Old MacDonald had a farm,

E-I-E-I-O.

And on that farm he had a cow,

E-I-E-I-O.

With a moo moo here and a moo moo there

Here a moo, there a moo, everywhere a moo moo

Old MacDonald had a farm,

E-I-E-I-O.

Old MacDonald had a farm…

Old MacDonald had a farm,

E-I-E-I-O.

And on that farm he had a [**animal name**],

E-I-E-I-O,

With a [**animal noise twice**] here and a [**animal noise twice**] there

Here a [**animal noise**], there a [**animal noise**], everywhere a [**animal noise twice**]

Old MacDonald had a farm,

E-I-E-I-O.

```
def verse(animal, sound):
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")
    print("And on the farm he had a", animal)
    print("E-I-E-I-O")
    print("With a", sound, sound,'here')
    print("And a", sound, sound, 'there')
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, sound)
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")


def main():
    verse("Cow", "Moo")


main()
```

How does it work?

```
def verse(animal, sound):
    print("Ol  MacD nald had a farm")
    print("F  E-I  )
    print("  n  farm he had a", animal)
    print("  E-I  )
    print("  a"  ound, sound,'here')
    print("  a",  und, sound, 'there')
    print("  e a  sound)
    print("  re  sound)
    print("  ryw  e a", sound, sound)
    print("  Ma  nald had a farm")
    print("  -E-  ")


def main():
    verse("Cow", "Moo")


main()
```

Copies "Cow" to animal

Copies "Moo" to sound

Calling the function more than once….

```
def verse(animal, sound):
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")
    print("And on the farm he had a", animal)
    print("E-I-E-I-O")
    print("With a", sound, sound, 'here')
    print("And a", sound, sound, 'there')
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, sound)
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")


def main():
    verse("Pig", "Oink")
    verse("Cow", "Moo")
    verse("Duck", "Quack")

main()
```

How does it work?

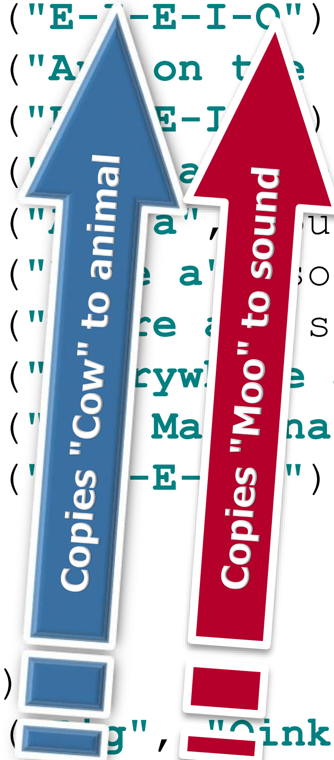
```
def verse(animal, sound):
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")
    print("A[illegible]on t[illegible]e farm he had a", animal)
    print("[illegible]E-I[illegible])
    print([illegible]und, sound,'here')
    print("[illegible]a", [illegible]und, sound, 'there')
    print("[illegible]e a[illegible]ound)
    print("[illegible]ce a[illegible] sound)
    print("[illegible]cyw[illegible]e a", sound, sound)
    print("[illegible] Ma[illegible]nald had a farm")
    print("[illegible]-E-[illegible]")


def main():
    verse("Pig", "Oink")
    verse("Cow", "Moo")
    verse("Duck", "Quack")


main()
```

Copies "Pig" to animal

Copies "Oink" to sound

How does it work?

```
def verse(animal, sound):
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")
    print("A[illegible] on t[illegible]e farm he had a", animal)
    print("[illegible]E-I[illegible])
    print([illegible]a[illegible]und, sound,'here')
    print("[illegible]a",[illegible]und, sound, 'there')
    print("[illegible]e a[illegible]sound)
    print("[illegible]re a[illegible] sound)
    print("[illegible]ryw[illegible]e a", sound, sound)
    print("[illegible] Ma[illegible]nald had a farm")
    print("[illegible]-E-[illegible]")



def main()[illegible]
    verse([illegible]g", [illegible]ink")
    verse("Cow", "Moo")
    verse("Duck", "Quack")


main()
```



How does it work?

```
def verse(animal, sound):
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")
    print("And on the farm he had a", animal)
    print("E-I-E-I-O")
    print("With a", sound, sound,'here')
    print("And a", sound, sound, 'there')
    print("Here a", sound)
    print("There a", sound)
    print("Everywhere a", sound, sound)
    print("Old MacDonald had a farm")
    print("E-I-E-I-O")


def main():
    verse("Cow", "Moo")
    verse("Sheep", "Baa")
    verse("Duck", "Quack")


main()
```

Copies "Duck" to animal

Copies "Quack" to sound

Order matters!

```
verse("Oink", "Pig")
```

Both parameters are non-null so this function call works but makes no sense

- `animal` gets the value "`Oink`"
- `sound` gets the value "`Pig`"

```
Old MacDonald had a farm
E-I-E-I-O
And on the farm he had a Oink
E-I-E-I-O
With a Pig Pig here
And a Pig Pig there
Here a Pig
There a Pig
Everywhere a Pig Pig
Old MacDonald had a farm
E-I-E-I-O
```

Arguments may be literal values


```
Verse("Cow", "Moo");
```

- `"Cow"` is passed to the function and stored in the function's local parameter `animal`.
- `"Moo"` is passed to the function and stored in the function's local parameter `sound`.

Arguments may be variables

```
def main():
    animal_name = input("Animal >>> ")
    animal_sound = input("Sound >>> ")
    verse(animal_name, animal_sound)
```

- **animal_name** is a variable, the value of which is passed to the function and stored in the function's local parameter **animal**.
- **animal_sound** is a variable, the value of which is passed to the function and stored in the function's local parameter **sound**.

```
Animal >>> Pig
Sound >>> Oink
Old MacDonald had a farm
E-I-E-I-O
And on the farm he had a Pig
E-I-E-I-O
With a Oink Oink here
And a Oink Oink there
Here a Oink
There a Oink
Everywhere a Oink Oink
Old MacDonald had a farm
E-I-E-I-O
```

Arguments may be expressions

```
verse("C" + "ow", "Mo" + "o")
```

`"C" + "ow"` → `"Cow"` is an argument, a value passed to the subroutine to be copied and used.

Arguments may be expressions


```
def main():
    animal_name = input("What is the animal? ")
    animal_sound = input("What is the sound that a {} makes? ".format(animal_name))
    verse(animal_name.capitalize(), animal_sound.capitalize())
```

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

http://www.cit.ie

Value-returning functions

- void function: group of statements within a program for performing a specific task
  - Call function when you need to perform the task
    - sing_verse(7)
    - sing_last_verse()

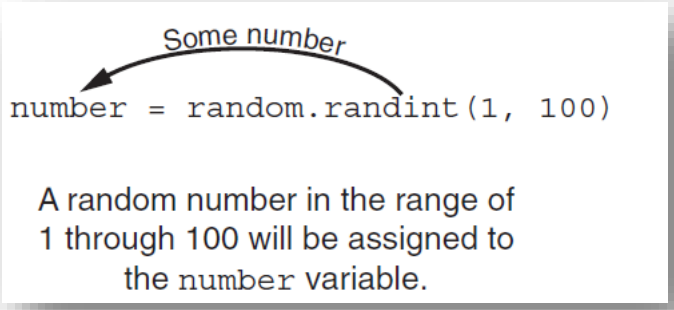
- Value-returning function: similar to void function, but **returns a value**
  - Value returned to part of program that called the function when function finishes executing

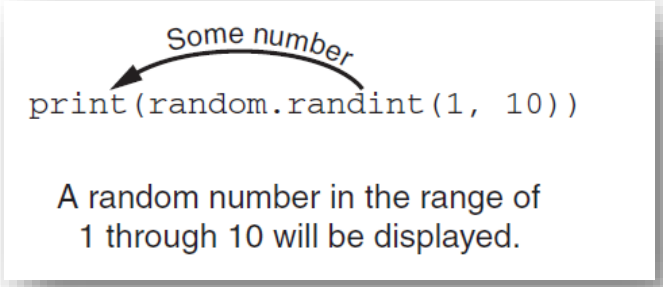
Introduction to using value-returning functions

- Before we write our own value returning functions it is worth noting that you have been ***using*** them in semester 1 from **Python's Standard Library**
- Full details → https://docs.python.org/3/library/index.html
- We will take a look at
  - The random number function
  - Some math functions

Generating Random Numbers

- Random number are useful in a lot of programming tasks
  - Python gives us functions to generate random numbers in the random module
    - `import random`
- **`randint`** function: generates a random number in the range provided by the arguments
  - Needs 2 arguments, a minimum and limit
  - Returns a random number to the calling function
  - Accessed using dot notation `random.randint()`





The `math` Module `import math`

`math` Module Function	Description
`acos(x)`	Returns the arc cosine of `x`, in radians.
`asin(x)`	Returns the arc sine of `x`, in radians.
`atan(x)`	Returns the arc tangent of `x`, in radians.
`ceil(x)`	Returns the smallest integer that is greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x` in radians.
`degrees(x)`	Assuming `x` is an angle in radians, the function returns the angle converted to degrees.
`exp(x)`	Returns e^x
`floor(x)`	Returns the largest integer that is less than or equal to `x`.
`hypot(x, y)`	Returns the length of a hypotenuse that extends from (0, 0) to (`x`, `y`).
`log(x)`	Returns the natural logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`radians(x)`	Assuming `x` is an angle in degrees, the function returns the angle converted to radians.
`sin(x)`	Returns the sine of `x` in radians.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x` in radians.

6_1_Random Math.py


```
import random
import math
number = random.randint(1,100)
print(number, math.sqrt(number))
```

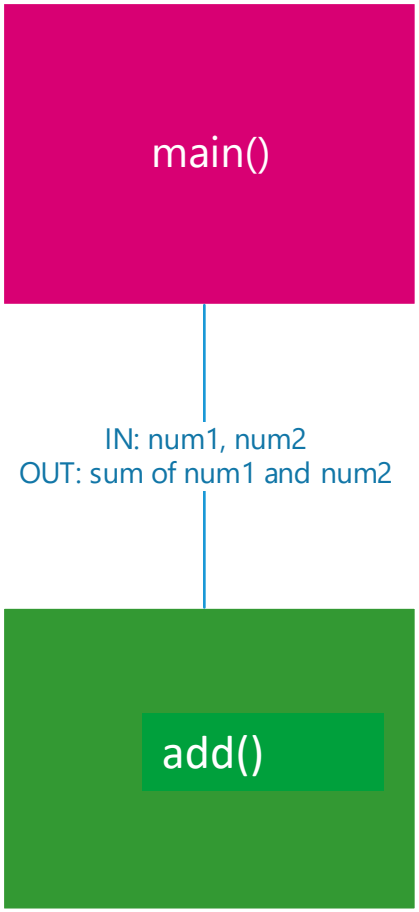
6_2_Random Math.py
import only what you need….


```
from random import randint
from math import sqrt
number = randint(1,100)
print(number, sqrt(number))
```

Writing Your Own Value-Returning Functions

- Write a function and add a `return` statement
  - Format: `return` *`expression`*
    - The value for *`expression`* will be returned to the part of the program that called the function
  - The expression in the `return` statement can be a complex expression, such as a sum of two variables or the result of another value- returning function

Showing data flow



Write A Value-Returning Functions

7_Adding.py

```
9
14.2
Hello!
```

```
def add(num1, num2):
    result = num1 + num2
    return result


def main():
    print(add(4, 5))

    answer = add(9.7, 4.5)
    print(answer)

    print(add("Hello", "!"))


main()
```

The name of the function is add().

The parameters are num1 and num2.

The value of result is returned to the calling function.

4 is copied to num1.

5 is copied to num2.

9 is returned from the function and printed to the screen.

9.7 is copied to num1.

4.5 is copied to num2.

14.2 is returned from the function to a variable called answer.

answer is printed to the screen.

"Hello" is copied to num1.

"!" is copied to num2.

"Hello!" is returned from the function and printed to the screen.

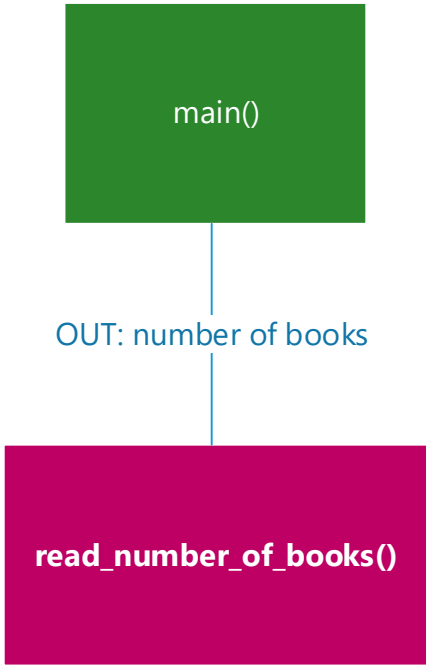
How to *Use* Value-Returning Functions

- Value-returning function can be useful in specific situations
  - Example: have function prompt a user for input and return the user's input
  - Simplify mathematical expressions
  - Complex calculations that need to be repeated throughout the program
- Use the returned value
  - Assign it to a variable or use as an argument in another function e.g. print

Prompting the user for data with validation

6-reading-functions.py

```
def read_number_of_books():
    while True:
        try:
            number = int(input("Number of books >>> "))
            if number > 0:
                break
            else:
                print("Non-negative numbers please...")
        except ValueError:
            print("Must be numeric...")
    return number


def main():
    number_of_books = read_number_of_books()
    print("You have asked for", number_of_books, "books.")


main()
```



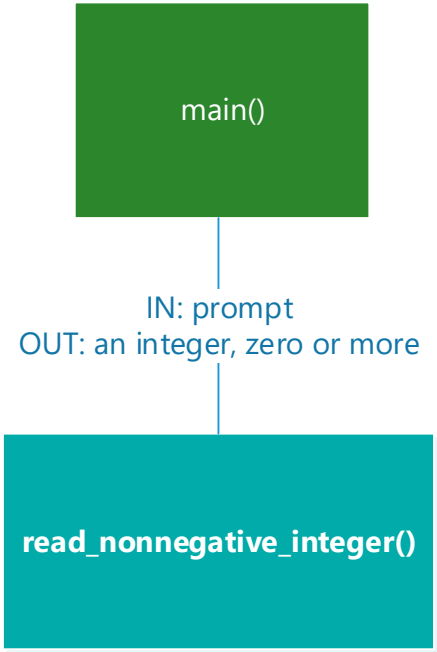
Making the function better….

- We should be able to re-use functions but this function only reads the number of books.
- To make it more generally useful we should change the prompt from "Number of books >>> " to whatever we need as we need it
- How?
  - Make the prompt a parameter

Edited function to include prompt as a parameter and renamed to better indicate its purpose

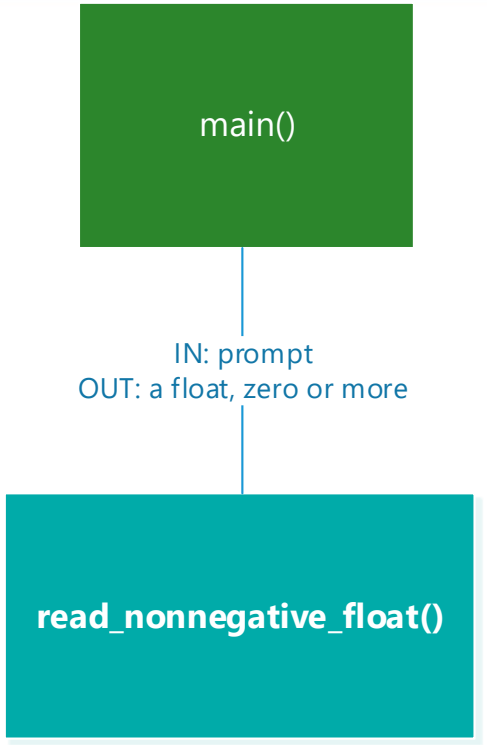
7-reading-functions.py

```
def read_nonnegative_integer(prompt):
    while True:
        try:
            number = int(input(prompt))
            if number > 0:
                break
            else:
                print("Non-negative numbers please...")
        except ValueError:
            print("Must be numeric...")
    return number


def main():
    number_of_books = read_nonnegative_integer("Number of books >>> ")
    print("You have asked for", number_of_books, "books.")

    number_of_children = read_nonnegative_integer("How many children >>> ")
    print("You have specified ", number_of_children, "children")

main()
```



You could write similar code for float...

```
def read_nonnegative_float(prompt):
    while True:
        try:
            number = float(input(prompt))
            if number >= 0:
                break
            else:
                print("Non-negative numbers please...")
        except ValueError:
            print("Must be numeric...")
    return number
```

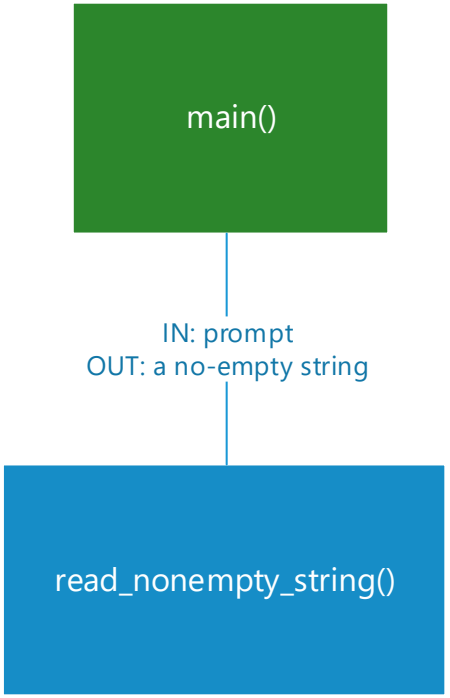



```
weight = read_nonnegative_float("Weight of flour (g) >>> ")
print("You have specified for", weight, "grammes.")
```

Returning Strings

```
def read_nonempty_string(prompt):
    while True:
        s = input(prompt)
        if len(s) > 0:
            something_is_wrong = False
        else:
            print("Please type something...")
    return s


def main():
    firstname = read_nonempty_string("First name >>> ")
    surname = read_nonempty_string("Surname >>> ")
    print(firstname, surname)


main()
```

You could add something else here e.g. s.isalpha() would ensure that the characters are all letters.



Returning Boolean Values

- Boolean function: returns either `True` or `False`
  - Use to test a condition such as for decision and repetition structures
    - Common calculations, such as whether a number is even, can be easily repeated by calling a function

```
def read_integer(prompt):
    while True:
        try:
            number = int(input(prompt))
            break
        except ValueError:
            print("Must be a whole number")
    return number


def is_even(x):
    return x % 2 == 0


def main():
    print(is_even(5))
    print(is_even(8))

    number = read_integer("What is the number? ")

    if is_even(number):
        print("{} is even".format(number))
    else:
        print("{} is odd".format(number))


main()
```

Menu Driven Programs

- Menu-driven program: displays a list of operations on the screen, allowing user to select the desired operation
  - List of operations displayed on the screen is called a *menu*
- Program uses a decision structure to determine the selected menu option and required operation
  - Typically repeats until the user quits

Plan

- We will need a function to
  1. Read and return the user's choice of beverage
     - OUT: user's choice of beverage
  2. Process the user's choice to return the cost of the beverage
     - IN: user's choice
     - OUT: cost of beverage
  3. Display the cost of the user's choice of beverage
     - IN: cost of beverage

12_menu.py

```
# global constants
COFFEE = 2.20
TEA = 1.70
MILK = 1.65

# function to get the user's drink choice
def get_choice():
    menu = "Would you like " + "\n\t1: Coffee" \
                               "\n\t2: Tea" \
                               "\n\t3: Milk" \
                               "\n==> "
    while True:
        try:
            number = int(input(menu))
            if 1 <= number <= 3:
                break
            else:
                print("Values 1, 2 or 3 please...")
        except ValueError:
            print("Values 1, 2 or 3 please...")
    return number

def process_choice(choice):
    if choice == 1:
        cost = COFFEE
    elif choice == 2:
        cost = TEA
    elif choice == 3:
        cost = MILK
    return cost

def print_choice(c):
    print("That will be €{:.2f}".format(c))

def main():
    users_choice = get_choice()
    cost_of_beverage = process_choice(users_choice)
    print_choice(cost_of_beverage)


main()
```

13_menu.py

```
COFFEE = 2.20
TEA = 1.70
MILK = 1.65

def get_choice():
    menu = "Would you like " + "\n\t1: Coffee" \
                               "\n\t2: Tea" \
                               "\n\t3: Milk" \
                               "\n\t4: Quit" \
                               "\n==> "
    while True:
        try:
            number = int(input(menu))
            if 1 <= number <= 4:
                break
            else:
                print("Values 1, 2, 3 or 4 please...")
        except ValueError:
            print("Values 1, 2, 3 or 4 please...")
    return number

def process_choice(choice):
    if choice == 1:
        cost = COFFEE
    elif choice == 2:
        cost = TEA
    elif choice == 3:
        cost = MILK
    return cost

def print_choice(c):
    print("That will be €{:.2f}".format(c))

def main():
    while True:
        users_choice = get_choice()
        if users_choice == 4:
            break
        cost_of_beverage = process_choice(users_choice)
        print_choice(cost_of_beverage)


main()
```

Today….

You should now be able to

- Explain what "pass by value" means
- Write code that receives multiple values
- Describe the significance of "pass by value" parameters
- Write code that returns a value from a function