

Functions and variables

Overview

- **Programming environment**
 - Tools for coding, the command line, the Python interpreter
- **Core programming concepts**
 - Functions, arguments and parameters, side effects and return values, variables, bugs
- **Working with strings**
 - User input, string manipulation, formatting output
- **Working with numbers**
 - Data types, arithmetic, type conversion, rounding and formatting
- **Software engineering best practices**
 - Comments and pseudocode, interactive mode, style and readability
- **Defining custom functions**
 - scope, default values, organizing code, `def`

The programming environment

- Tools for coding
- The command line interface
- The Python interpreter
- An editor

Tools for coding

- **Programs can be written in any text editor**
 - a programming editor makes life easier
- **Visual Studio Code (VS Code)**
 - popular
 - syntax highlighting – colours code by meaning, making mistakes easier to spot
 - integrated terminal – run your code without leaving the editor
- **Other environments are available**
 - Thonny (beginners)
 - Simple programming **Text Editor** & **Terminal Application**
 - Windows:** **Notepad++** & **PowerShell**
 - mac OS:** **TextMate** & **Terminal**

The command line interface

- **Not every interaction is mouse-based**
 - Terminals: speak (type) directly to your machine
- **Why use it?**
 - faster once you know the commands
 - it's how you'll run your Python programs in this course
- **Aren't command lines old fashioned?**
 - Yes, as old as the 1970's!
 - Old fashioned?
 - Many modern AI/LLM tools are command line based!
 - The past is sometimes also the future!

The Python interpreter

- **Python is two things at once**
 - a language – the rules for writing code
 - a program (the interpreter) – reads and runs that code
- **How it reads your code**
 - top-to-bottom, left-to-right, like a typical book
 - one instruction at a time
- **What it actually does**
 - translates your code into binary (0s and 1s)
 - Not easy for us to digest
 -but the computer can (and very quickly)

Core programming concepts

- Functions
- Arguments and parameters
- Return values
- Variables
- Bugs and syntax errors

Functions

- **Functions are verbs of a program**
 - actions that let your program do things
- **The simplest example: `print`**
 - displays text on the screen
 - `print("hello, world")`

Arguments and parameters

- **Arguments**

- values you pass into a function when you call it
- e.g. the string `"hello, world"` in `print("hello, world")`

- **Parameters**

- placeholders in a function's definition: indicate what it can accept
- Defined in functions `def`. Example, Python's `print()` function definition

```
def print(*objects, sep='', end='\n', .....):
```

- **Know that these are different things**

- you will refer to arguments as parameters and vice versa
- ...as long as you *know* the difference
- ...it is what things do that matters over what they are called

Side effects and return values

- **Side effect**

- something a function does, e.g. printing text to the screen

- **Return value**

- data a function hands back, so the rest of the program can use it
- example:

```
user_input = input("Please enter some input >")
```

- **Why the difference matters**

- a printed value is gone once it's on screen
- a returned value can be stored and reused

Variables store values

- **Variable are containers for values**
 - stored in the computer's memory, under a name you choose
- **What they're for**
 - storing data so it can be reused later in the program
 - e.g. `name = "Guido van Rossum"`
 - storing in memory at some *address*



<https://www.youtube.com/watch?v=-DVyjdW4t9I>

Bugs and syntax errors

- **A bug is simply a mistake in your program**
- **A syntax error**
 - occurs when your code breaks the language's grammar rules
 - e.g. a missing closing parenthesis (or a missing colon :
- **Good news**
 - Python usually tells you exactly which line the error is on
 - Not as verbose as a large language model (good or bad?)

Working with strings

- User input
- String manipulation
- Formatting output

User input

- `input()` asks the user for text
 - pauses the program until the user types something and presses enter
- **Always a string**
 - whatever the user types, `input()` returns it as a string – even if it looks like a number

String manipulation

- **Python gives you built-in methods to clean up text**
 - `.strip()` – removes whitespace from the left and right
 - `.capitalize()` – makes the first letter uppercase
 - `.title()` – makes the first letter of every word uppercase
 - `.split()` – breaks a string into parts, based on a separator

Concatenation and parameters

- **Concatenation**

- joining strings together with `+`

- **Named parameters**

- override a function's default behaviour
 - e.g. `print`'s `sep` (what goes between values) and `end` (what comes after)

F-strings and escape characters

- **F-strings**

- a modern, elegant way to build strings
- drop variables straight into curly braces: `f"hello, {name}"`

- **Escape characters**

- a backslash lets you type characters that are otherwise awkward
- `\"` for a quote mark, `\n` for a new line

Working with numbers

- Data types
- Arithmetic operators
- Type conversion
- Rounding and formatting

Data types: int and float

- **int** – whole numbers
 - no upper bound in Python (your computer memory is the limit)
- **float** – numbers with a decimal point
 - finite precision – very long decimals can be rounded internally

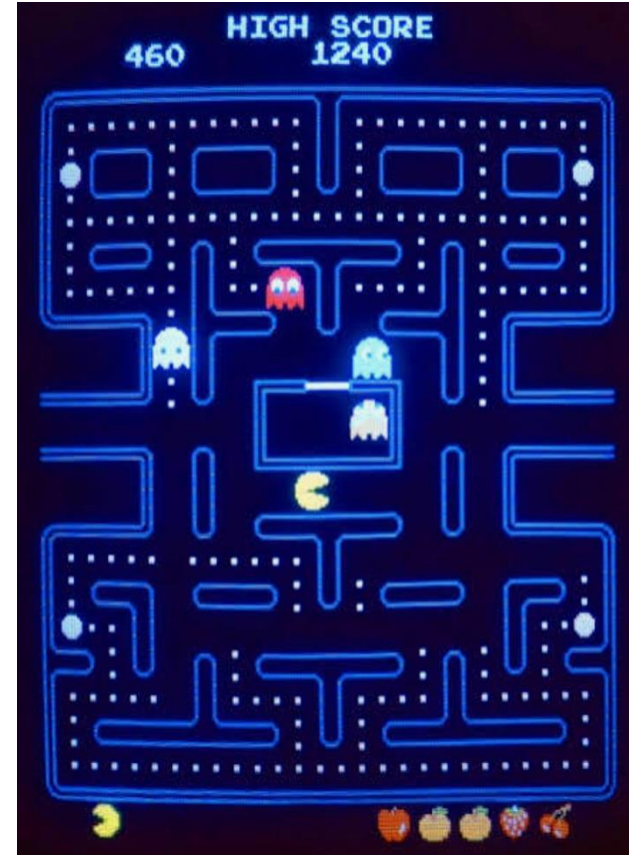
Arithmetic operators

- **The basics**

- + addition, - subtraction
- * multiplication, / division

- **Modulo: %**

- finds the remainder of a division
- e.g. $7 \% 2$ is 1
- $pos = (pos + vel) \% width$
world wrap-around



Type conversion

- `input()` always returns a string
 - even if the user types a number, Python sees text
- **Converting text to numbers**
 - `int()` for whole numbers, `float()` for decimals
 - needed before you can do maths with user input, for example

Rounding and formatting numbers

- `round()`
 - rounds to the nearest whole number, or to a chosen number of digits
- **Formatting inside f-strings**
 - add commas for thousands, or limit decimal places
 - e.g. `f"{value:.2f}"` shows exactly two decimal places

Good practices

- Comments and pseudocode
- Interactive mode 'REPL'
- Style and readability in scripts

Comments and pseudocode

- **Inline comments**

- notes for humans, starting with #
- the computer ignores them completely

- **Pseudocode**

- an outline of your program in plain English
- map out the logic before you write a single line of Python

Interactive mode

- **Python can run one line at a time**
 - type directly into the terminal and see results immediately
 - REPL: Read, Evaluate, Print, Loop
- **Why use it**
 - great for quick tests, without writing a whole file
- **Generally, you will put code into Scripts:**
 - Persistnet place on disk, called by by python
 - Much better than REPL

Style and readability

- **Rarely one way to solve a problem**
- **Prioritise:**
 - simple, readable code over 'clever' code
 - code other people – and future you – can follow easily

Custom functions

- The `def` keyword
- Scope
- Default values
- Organizing code

The `def` keyword

- `def` lets you invent your own functions
 - give it a name, some parameters, and a block of code to run
 - e.g. `def greet(name) :`

Scope

- **Variables have a home**
 - a variable defined inside a function only exists inside that function
- **Why it matters**
 - keeps functions self-contained, and prevents accidental name clashes

Functions (tips)

- **clear descriptive names**
 - `calculate_area()`
 - `euclidean_distance()`
- **...rather than lazy naming**
 - `doStuff, f1`
- **clear single-purpose**
 - `calculate_area()`
- **...rather than multi-purpose**
 - `calculate_area_and_print()`

args / params **in**



functions have behaviour

product/result returned **out**

Functions (tips)

- **use parameters, not hard coding**
 - will encourage flexible functions and re-use
- **use return values**
 - don't just print out what you should return for further use within another area of the code

```
# Bad: only works for radius = 5  
def area_circle():  
    return 3.14 * 5 * 5  
  
# Good: works for any radius  
def area_circle(radius):  
    return 3.14 * radius * radius
```

```
# Less flexible  
def add(a, b):  
    print(a + b)  
  
# Better  
def add(a, b):  
    return a + b
```

Default values

- **Parameters can have a default value**
 - e.g. `def greet(name="friend") :`
- **Why it's useful?**
 - lets a function be called with fewer arguments – or none at all.

Organizing code: `main()`

- **Convention: put program's high-level logic in `main()`**
 - keeps the top-level flow of the program easy to follow
- **Call it at the very bottom of the file**
 - ensures every other function is already defined before it's used

Summary

- **The programming environment**
 - VS Code, the command line, the Python interpreter
- **Core programming concepts**
 - functions, arguments and parameters, side effects and return values, variables, bugs
- **Working with strings**
 - input, string methods, formatting output
- **Working with numbers**
 - data types, arithmetic, type conversion, rounding and formatting
- **Good practices**
 - comments and pseudocode, interactive mode to quick-test, style and readability
- **Defining custom functions**
 - `def`, scope, default values, organizing code

Thanks!