Lecture 3 – The basics of types

Basics

+ - / *

- With division, there will be decimal place
- ****** (exponent, to the power of)
- **%** (modulo, remainder after division)
- **//** ("floor" division, whole number rounded down for division)
- BODMAS → can override with parentheses ()
- **Underscore (_)** returns the value of the last calculation
  - Even if variables are used to calculate, can use it calculate something new
- If float used in equation, will return a float
- Numerically "break" python
  - Be careful when dealing with decimal numbers (finite number of decimal places) (i.e. pi)
  - 0/0 → Zerodivisionerror

Sequence of items

- Sequence (iterable) – the decomposition of an object into a well define ordering of items
  - Text, sounds and images as sequences
- Manipulation of objects: "iteration" over iterables

Programming and design

- "conditioning" on particular values, "looping" over subsets of steps and "nesting" of loops
- Ways to represent programs : Flowcharts, Pseudo-code (use normal English to write out code)

The print function: print()

- In the console (terminal) - no noticeable difference between printing and executing a variable
- But when you "run" code from a file, you will only see the output of print () functions
- Print("2"+"3") → 23

Syntax and semantics

- Syntax – arrangement of words/phrases to create sentences
  - Print hello ''(incorrect syntax) → print ('hello') → correct syntax
- Semantic – the meaning of a word, phrase/text
  - Print (1+2) "+"= addition

Types

- Every object has a type which defines: semantics + operators, functions and methods can be applied to it
- 2 number types
  - Int (integer)
  - Float (real number) – contains a decimal place, otherwise it is an integer (even if whole number
  - Complex (complex numbers) complex(1) → 1 + 0j
- **Type function: type ()**
  - **print (type(2))→ class 'int'** **print type((2.0))→ class 'float'**
  - The semantics of operators and functions is determined by types of the operands
    - Print (type (1+1))→ type 'int'
    - Print (type (1/2)) → type 'float'

Literals and variables

- Literals –value is fixed and has invariant semantics
- Variable – used to store values of arbitrary name via assignment (=) which is an operator
  - The RHS is first 'evaluated', then the value is assigned to the LHS (use original value to calc)
  - The assignment can only be to a single object (LHS)s
  - Variables changed only through reassignment
- Stacking - assign same evaluated result to multiple variables A = g = d = g = 3

Lecture 4 – conditionals

Variable naming conventions

- Variables need to start with alphabet or underscore
  - Can consist of alphanumeric characters and underscores
  - Casing is significant (Apple is different to apple)
- "reversed words" (operators, literals and built-in functions) cant be used for variable name (in, noy)
    - Valid: abd123, _123abc Invalid: 1, a-z, 13CABS, in
- Type conversion
  - Determines type of literal and variables based on syntax (literals) + type of assigned value(variable)
  - To cast literal/variable to a different type
    - **Int(), float(), str(), complex()**

Strings (str)

- "chunk" of text, standardly enclosed within either single or double quotes
- To include quotation marks and slashes → 'escape; then with a \(backslash) in front
  - **\" \' **
- And special character for formatting → **\t (tab) \n(newline)**
- OR can use triple quote to avoid escaping/special characters
  - """"Ow," he said/yelled."""
- String operators
  - Main binary operators which can be applied to strings are:
    - **+ (concatenation)** Print("a" + "b") → ab
    - *** (repeat string N times)** Print('Z' *2) → ZZ
    - **In(subset)**
      - Print('z' in "zizzer zazzer zuzz") → true
      - Ask if z is a substring (left) of the superstring(right)
- Overloading → + and * means different things in **int** and **float** than to strings \

The boo1 type

- Boolean(bool)type has 2 values: True or False
- **bool function bool()**
  - Every type has unique for which bool() evaluates to False
    - String: empty ("") Int: zero (bool(0.0) or bool(0)) Bool(None)
  - OR you can type in..
    - 4>3 → false "bob" in "bobbity bob" ➔ true

Comparison

- == equality; not the same as =
- > , >= , <, <= != (no equal to) In (an element of)

Logic operatators (and, or, not)

And	**True**	**False**
True	True	False
False	False	False

or	**True**	**False**
True	True	True
False	True	False

not	**True**	**False**
	False	True

- Precedence not > and > or [nao]
  - Example: not true and false or true and false
    - ((not true) and false) or (true and false)
    - False or false
- Example
  - Tall and ears == "rabbit" and 3 → bool(tall) and bool(ears=="rabbit") and bool(3)
  - Equality takes precedence