Foundations of Computing

The Basics of Programming

Ekaterina Vylomova
Summer Term 2023

THE UNIVERSITY OF MELBOURNE

January, 18th: On this day...(random facts)

- **1486** – King Henry VII of England marries Elizabeth of York, daughter of Edward IV, uniting the House of Lancaster and the House of York
- **1778** – James Cook is the first known European to discover the Hawaiian Islands, which he names the "Sandwich Islands".
- **1896** – An X-ray generating machine is exhibited for the first time by H. L. Smith.
- **1911** – Eugene B. Ely lands on the deck of the USS Pennsylvania anchored in San Francisco Bay, the first time an aircraft landed on a ship.
- **2003** – A bushfire kills four people and destroys more than 500 homes in Canberra, Australia.

Lecture Agenda

- Last lecture:
  - Computers speak binary, but we don't
  - High level programming languages make life easier
  - Programming languages differ from natural languages
  - We will use Python inside Grok
- This lecture:
  - Programming
  - Blocking
  - Operators and their priority
  - Basic Data Types

Lecture Outline

Programming

- Computer programs are simply sets of steps to complete some task
- Determining what the steps should be requires learning how computers "think"... and how a particular programming language expresses the way a computer thinks
- At its most basic level, a program is made up of a sequence of **statements** that are executed sequentially one after the other

A Simple Program for Visiting a Friend's House

- Head south on the Peninsula Freeway
- Take the Red Hill exit
- Keep taking the left turn at each intersection until you cross the bridge
- If you reach the beach, turn around and go back (you've gone too far)
- Park at the Red Hill Bakery

Basic Programming Building Blocks

- The basic building blocks of programming are:
  - **statements** (= single "commands" to the computer)
  - **sequence** (= linear sequence of statements)
  - **control** (= perform sequence of statements IF condition holds)
  - **loops** (= repeat sequences of statements)
  - **functions** (= blocks of code that can be run with different inputs)
  - **recursion** (= blocks of code that call themselves with different inputs)

On Algorithms and Computability (Advanced)

Turtle Programming

- As an illustration of this, without getting bogged down in the details too much, let's play around with Turtle graphics, using the "blockly" programming language
- Basic commands:
  - advance forward/backward `N` units
  - turn left/right `N` degrees

Class Exercise

- Using just `move` and `turn` statements, build blockly code to draw an equilateral triangle with side length 100

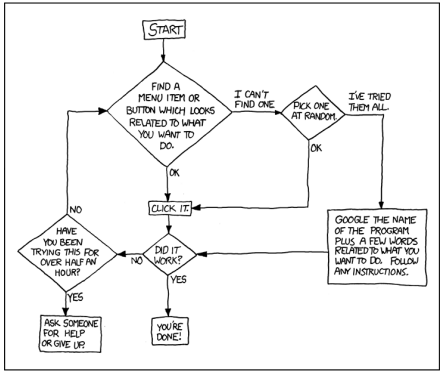
Program Design

- Many modern computing languages express similar concepts
- They allow "conditioning" on particular values, "looping" over sub-sets of steps, and "nesting" of loops
- Common ways to abstractly represent programs are:
  - flowcharts
  - "pseudo-code" (i.e. a computer program in an abstract language, without the "bookkeeping" that individual languages require)
    `http://www.bestrecipes.com.au/recipe/choc-chip-cookies-L4351.html`

Example Flowchart

DEAR VARIOUS PARENTS, GRANDPARENTS, CO-WORKERS, AND OTHER "NOT COMPUTER PEOPLE."

WE DON'T MAGICALLY KNOW HOW TO DO EVERYTHING IN EVERY PROGRAM. WHEN WE HELP YOU, WE'RE USUALLY JUST DOING THIS:



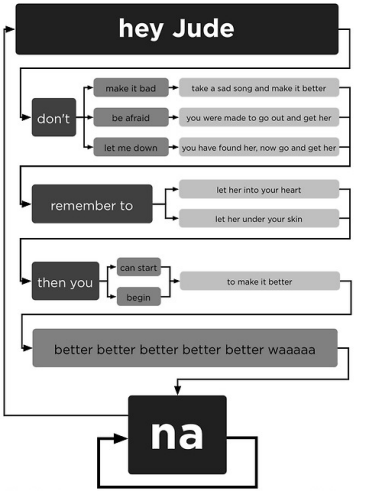
PLEASE PRINT THIS FLOWCHART OUT AND TAPE IT NEAR YOUR SCREEN.
CONGRATULATIONS; YOU'RE NOW THE LOCAL COMPUTER EXPERT!

Source(s): `https://xkcd.com/627/`

Equivalent Pseudocode

```
1: repeat
2:     find a related menu item OR pick one at random you haven't tried
3:     if found one then
4:         click it
5:         if it worked then
6:             done!
7:         else if been going for > 30 mins then
8:             give up!
9:         end if
10:    else
11:        Google a solution
12:        go to 5
13:    end if
14: until done! OR give up!
```

More Interesting Flowchart



Source(s): `http://laughingsquid.com/hey-jude-flow-chart/`

Lecture Outline

Python as a Glorified Calculator I

To start out, let's use Python as a glorified calculator:

- basic arithmetic: `+` (addition) `-` (subtraction) `/` (division) `*` (multiplication)
- also: `**` (exponent), `%` (modulo), `//` (integer/"floor" division)
- 

Beware, in Python2, $10/3 = 3$

Python as a Glorified Calculator II

- Python uses "BODMAS" to associate **"operands"** (the targets of **operations/operators**) by default, which you can override with "parentheses":
  - `1 + 2 * 3` vs. `(1 + 2) * 3`
- special character _ stores the value of the last calculation

BODMAS: B-Brackets, O-Orders (powers/indices/roots), D-Division, M-Multiplication, A-Addition, S-Subtraction

Operation Priority

- exponentiation: $2^{100} \rightarrow$ 2 ** 100 ($2^{4^5} \rightarrow$ 2 ** 4 ** 5 (right-to-left))
- (Unary) minus: -4; plus: +4
- (Binary) multiplication: 2 * 4; division: 5 / 2; floor division (div): 5 // 2; modulus (mod): 5 % 2 (left-to-right: a / b * c)
- (Binary) addition: 2 + 4; subtraction: 4 - 2

Operation Priority

- Bitwise AND
- Bitwise XOR
- ...
- Boolean OR

Class Exercise

- -2**3*2 = ? Answer on SpeakUp: Room 45980

Class Exercise

- -2**(3*2) = ? Answer on SpeakUp: Room 45980

Class Exercise

- Armed just with these operators, let's explore the limitations of what we can do; is it possible to:
  - calculate $n!$ $(= n \times (n-1) \times ...2 \times 1)$ for an arbitrary n?

Vote on SpeakUp: Room 45980 (A: Yes, B: No)

The print Function

- The `print()` function can be used to print the value of the operand (of any type)

```
>>> print(1)
1
```

- In the terminal, there is no noticeable difference between printing and executing a variable:

```
>>> print(1)
1
>>> 1
1
```

but when you "run" code from a file, you will only see the output of `print()` functions

The print Statement

- In Python 2, you can use either the `print` statement (`print ...`) or the `print` function (`print(...)`), but Python 3 only allows the `print` function

```
>>> print(1)
1
>>> print 1    # Python 2
1
```

so if you use Python2 code from the web, remember to convert `print` statements to `print` functions

Jargon Alert

- Syntax: "the arrangement of words and phrases to create well-formed sentences in a language"

```
print hello''()      incorrect syntax
print('hello')       correct syntax
```

- Semantics: "the meaning of a word, phrase, or text"

```
print(1 + 2)         "+" = add two numbers
```

Lecture Outline

Basic Data Types (1)

- In Python, every object has a "type", which defines: (a) what operators, "functions", and "methods" can be applied to it, and (b) its semantics
- The two number types we will see most of are:
  - `int` (integer)
  - `float` (real number)

  also `complex` for complex numbers
- So how does Python work out the type for a given (real) number? If it contains a decimal place (.), it's a `float`, otherwise it's an `int`

Basic Data Types (1): int

Unlike C, Python does not make distinction between ‘long int’, ‘short int’, ’unsigned int’

`char`	Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned. It contains CHAR_BIT bits.[3]	8	%c
`signed char`	Of the same size as *char*, but guaranteed to be signed. Capable of containing at least the [−127, +127] range.[3][note 1]	8	%c (or %hhi for numerical output)
`unsigned char`	Of the same size as *char*, but guaranteed to be unsigned. Contains at least the [0, 255] range.[3]	8	%c (or %hhu for numerical output)

`short`
`short int`
`signed short`
`signed short int`
	Short signed integer type. Capable of containing at least the [−32,767, +32,767] range.[3][note 1]	16	%hi or %hd

`unsigned short` `unsigned short int`	*Short* unsigned integer type. Contains at least the [0, 65,535] range.[3]	16	%hu

`int`
`signed`
`signed int`
	Basic signed integer type. Capable of containing at least the [−32,767, +32,767] range.[3][note 1]	16	%i or %d

`unsigned` `unsigned int`	Basic unsigned integer type. Contains at least the [0, 65,535] range.[3]	16	%u

`long`
`long int`
`signed long`
`signed long int`
	Long signed integer type. Capable of containing at least the [−2,147,483,647, +2,147,483,647] range.[3][note 1]	32	%li or %ld

`unsigned long` `unsigned long int`	*Long* unsigned integer type. Capable of containing at least the [0, 4,294,967,295] range.[3]	32	%lu

`long long`
`long long int`
`signed long long`
`signed long long int`
	Long long signed integer type. Capable of containing at least the [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] range.[3][note 1] Specified since the C99 version of the standard.	64	%lli or %lld

`unsigned long long` `unsigned long long int`	*Long long* unsigned integer type. Contains at least the [0, +18,446,744,073,709,551,615] range.[3] Specified since the C99 version of the standard.	64	%llu

Source(s): `https://en.wikipedia.org/wiki/C_data_types`

Basic Data Types (2): str

A single data type for all types of strings, from a single character to a novel-long text (defined by a single or double quotes):

```
>>> print(type('a'))
<class 'str'>
>>> print(type("hello world"))
<class 'str'>
```

More on the topic during the next lecture!

Basic Data Types (3): bool

Boolean values (either `True` or `False`), any non-empty string and non-zero value is evaluated as `True`

```
>>> print(bool("False"))
True
>>> print(type(False))
<class 'bool'>
>>> print(type(1==0))
<class 'bool'>
```

More on the topic during the next two lectures!

Basic Data Types (4)

- Use the function `type` to determine the type of an object:

```
>>> print(type(1))
<class 'int'>
>>> print(type(1.0))
<class 'float'>
```

- The semantics of operators and functions is determined by the types of the operands:

```
>>> print(type(1 + 2))
<class 'int'>
>>> print(type(1.0 + 2))
<class 'float'>
```

Types and Operators (overloading)

- Some operators are defined differently for specific classes/types (polymorphism), e.g. "+" that is addition for numerical types and concatenation for strings;

```
>>> 2 + 4
6
>>> "2" + "4"
'24'
```

or "*":

```
>>> 2 * 4
8
>>> "2" * 4
'2222'
```

Basic Data Types (5)

- Python implicitly determines the type of each literal and variable, based on its syntax (literals) or the type of the assigned value (variables)
  No need to assign type explicitly (like in C): int a;

```
>>> a = 5
>>> print(type(a))
<class 'int'>
>>> a = "Hello"
>>> print(type(a))
<class 'str'>
```

Type Conversion

- To "convert" a literal/variable to a different type, we use functions of the same name as the type: `int()`, `float()`, `complex()`

```
>>> print(float(1))
1.0
>>> print(int(1.5))
1
>>> int('a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'a'
```

Literals, Variables, Assignment

Mutable vs Immutable Data Types

Lecture Summary

- Programming Building Blocks
- Priority of operations: BODMAS
- Types: what are they, what basic types have we learned, and how do you determine the type of an object?
- What are the basic numeric types, what operators are associated with them, and what interactions are there between the types and operators?
- Type conversion: how do we test the type of an object, and how do you convert the type of an object?