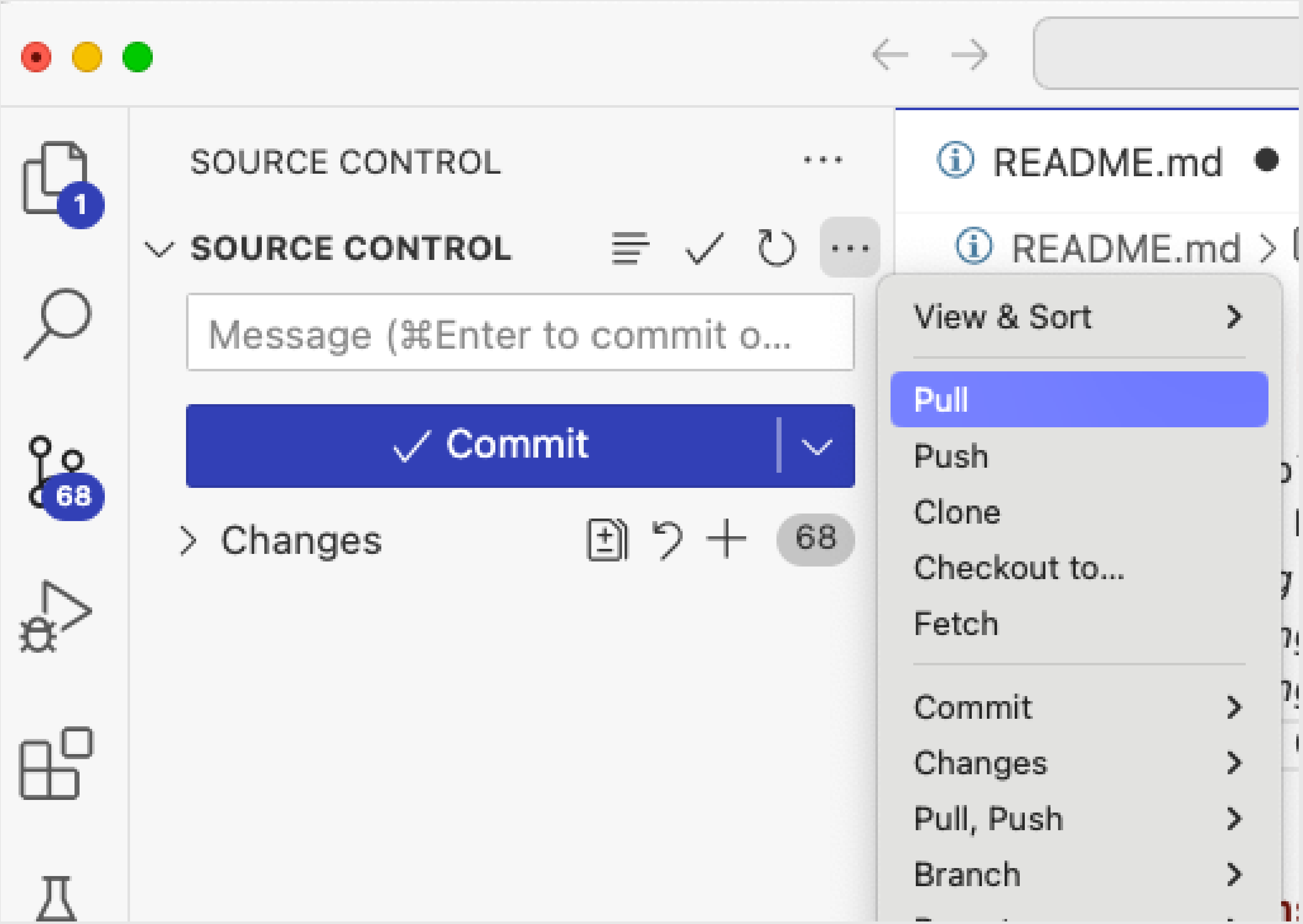
Applied Programming I

Programming computers in a nutshell.

Introduction to programming

Part 2



Reminder: did you do 'git pull'?

Today's objectives

- Understand some principles of programming languages:
    - Variables continued
    - Control flow continued
        - Conditions
        - Loops
    - Functions

Comments in code

- Programming languages allow comments. These are not considered as program code but for better understanding the programmer’s thoughts.
- Codes are recognized by the interpreter/compiler by preceding characters, e.g., ‘//’, ‘#’, ’/* … */.
- Comments in code should be useful, thus, explaining something important that is not obvious.

Comments in code: examples

- Comment that give you information

```
/* purpose: compute the second binomial
for given input parameters of integer a and b,
it returns integer value of a*a + 2*a*b + b*b */
int second_binomial(int a, int b) { …
```

- Comment that does not give you information that is no obvious

```
if x > 0: # if x > 0
```

Variables

Part 2

Variables:

- Types of variables:
  - Primitive data types
    - Boolean: true or false
    - Numeric:
      - Integer (sign or unsigned, different range/sizes)
      - Floating-point types (different accuracy/size)
      - Characters
  - Composite data types
      - Arrays (fixed and dynamic)
      - String (array of characters)
      - Structs (own build data type)
      - Enumerations
      - Pointers and References
      - Dictionaries and Sets

Naming of variables

- With developing software, there are conventions used that describe how the source code should ‘look’ like. These guidelines are usually given by the organization developing the software.
- The naming of variables is part of the conventions.
- Programming languages also have preferred layout of variables.

Typical variable naming conventions.

- camelCase: thisIsMyVariable
- PascalCase: ThisIsMyVariable
- snake_case: this_is_my_variable
- UPPPER_SNAKE_CASE: THIS_IS_MY_VARIABLE

Common naming usage

	C	C#	Python	JavaScript
camelCase		variables		variables, functions
PascalCase		class names, functions	class names	
snake_case	variables, functions		variables, functions	
UPPER_SNAKE_CASE	const values	const values	const values	const values

Primitive data types

- Boolean: true or false
- Numeric:
  - Integer (sign or unsigned, different range/sizes)
  - Floating-point types (different accuracy/size)
  - Characters

Boolean data types

- Python : b = True
- JavaScript : let b = true;
- C : bool b = true;
- C# : bool b = true;

Boolean data types can be either true or false. Their main use is for conditions.

Integer data types

- The size of an integer data type is given in number of bytes. The size determines the range the integer data type can cover.
- The size of data types can be system dependent.
- An unsigned integer has only zero and positive values, a signed integer negative and positive values.
- Negative values are in the two's complement representation in most programming languages.

Integer data types (positive numbers)

- Integer numbers are represented as binary numbers. As example, consider an unsigned number of the size of one byte (=8 bits).

Bit number	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
represents	128	64	32	16	8	4	2	1
Example	1	1	0	0	1	1	0	1
Bit	128	64	0	0	8	4	0	1
Result	128+	64+			8+	4+		1 = **205**

?

How are negative numbers represented?

Any idea?

Integer data types (negative numbers)

- Signed integers use one bit as sign. The numbers are in the two-complement.

Bit number	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
represents	+/-	64	32	16	8	4	2	1
Example	1	1	0	0	1	1	0	1
If b7=1, flip	-	0	1	1	0	0	1	0
and minus 1	-	0	1	1	0	0	0	1
Result	-		32+	16+			2+	1 = **-51**

Two's complement

0b is prefix for binary,
0x is prefix for hexadecimal

- Example: 37 – 37:
- Positive number is 37 = 32+4+1 = 0b 0010 0101.
- Negative number is given by flipped positive binary number + 1:
  flip(0b0010 0101) = 0b1101 1010,
  0b1101 1010 + 0b1 = 0b1101 1011
- Binary addition of both (highest bit (called carry) is discarded:
  0b0010 0101 +
  0b1101 1011 =
  0b0000 0000

Integer data types and ranges (LP64)

Data type	Size in bytes	Signed range	Unsigned range
byte or char	1	$-2^{7}...2^{7} - 1$	$0...2^{8} - 1$
short or word	2	$-2^{15}...2^{15} - 1$	$0...2^{16} -1$
int	4	$-2^{31}...2^{31} - 1$	$0...2^{32} -1$
long int	8	$-2^{63}...2^{63} - 1$	$0...2^{64} -1$

- The data types can be language and operating system dependent. The LP64 data model is very common (e.g. GCC and Unix-based systems).

Integer example

- Python : b = 5
- JavaScript : let b = 5;
- C# : int b = 5;
- C : int b = 5;

How are floating point numbers represented?

Any idea? ... This is somehow more complicated.

Floating-point numbers

- Floating-point data types use a binary format to encode numbers. A common format is described in the standard IEEE 754.
- The basic idea is to use one bit for the sign, some bits for an exponent and some bits for the mantissa. E.g., a 32-bit format would use this layout:

1 bit	8 bits	23 bits
Sign	Exponent	Mantissa

Floating-point numbers: example

Encoding the number 21.375 manually in IEEE 754 single precision:

1. Write integer and fractional part in binary form:
   21 = 0b10101 and .375 = .25 + .125 = .0b011
   (note the bits for the fractions are $2^{(-1)}$, $2^{(-2)}$, $2^{(-3)}$,...) and shift for representation with a 1 before the point: 0b1.0101011 x 2^4
2. Encode mantissa: the leading 1 does not need to be encoded: 0b0101001
3. Encode exponent with bias 127: 127+4 = 131 = 0b1000 0011
4. Encode sign, exponent and mantissa (complete with 0s):
   0b0100 0001 1010 1011 0000 0000 0000 0000 = 0x41AB 0000

Floating-point numbers examples

- Python : a = 5.1 # 64-bit
- JavaScript : let a = 5.1; // 64-bit
- C# : float a = 5.1F; // 32-bit
- C : float a = 5.1f; // 32-bit

C# and C have ‘double’ as 64-bit floating-point data type.

Characters

- Characters are encoded as numbers. The type of the encoding determines the size of the character.
- ASCII encoding uses 255 possible characters: one byte is enough.
- UTF-16 encoding uses more characters: two bytes are used.
- Example ASCII-codes: digits 0 to 9 are decimal numbers 48 to 57, uppercase characters A to Z are 65 to 90, lowercase characters a to z are 97 to 122.

Characters example

- Python : c = ‘a’
- JavaScript: let c = ‘a’;
- C# : char c = ‘a’;
- C : char c = ’a’;

Your task:
Encode “hello” in ASCII

In groups of two, encode the characters of ”hello”. Time: 10 minutes.

Then, we discuss together.

ASCII codes for ‘hello’

h	e	l	l	o
104	101	108	108	111

Type conversion / casting

- A type conversion is required if a variable of one datatype is assigned to a variable of another datatype.
- As datatypes have different purposes (e.g., ranges in case of integers), the variable's value need to fit.
- Implicit type conversion often works from a lower sized data type into a higher sized data type, e.g., short to int to long.
- Explicit type conversion / casting (should be used for higher to lower data type) requires using features of the programming language.

Type conversion examples – C

```
short x = 5;
int y = x;          /* implicit conversion */
char c = 'h';       /* c has the ASCII value of 'h', 104 */
unsigned int z = c; /* implicit conversion */

float f = 2.6f;
int a = f;          /* implicit conversion! */

int b = 256;
unsigned char u = b; /* implicit conversion, attention! What is the result? */

unsigned int ui = (unsigned int) b; /* explicit conversion using cast operator() */
ui = b;                             /* implicit conversion */
```

Type conversion examples – C#

```
short x = 5;
int y = x;    // implicit conversion
char c = 'h'; // c has the ASCII value of 'h'
uint z = c;   // implicit conversion

float f = 2.6f;
int a = (int) f;   // explicit conversion / cast

int b = 256;
byte u = (byte) b;  // explicit conversion / cast

uint ui = (uint) b; // explicit conversion / cast
// ui = b; /* error in c# */
```

Type conversion examples - JavaScript

- JavaScript has only one type for numbers: Number
- Conversion is possible for other data types, e.g., Number and Strings (later)

Type conversion examples - Python

- Python converts implicitly to the higher data type, but has explicit casting operators:

```
c = 5.4       # c is float, value = 5.4
d = int(c)    # expl. conv., d is int, value = 5
e = float(d) # expl. conv. d is float, value = 5.0
f = c + d     # impl. conv. f is float, value = 10.4
```

Untyped and typed languages: notes

1. In untyped languages, it is sometimes not intuitive to know which kind of data type a variable has.
2. Conversion of types can be done (happen) implicit or can be (must be) done explicit. However, type conversion should be done carefully (as it can result in errors).

Untyped languages and input

- Input functions may use an input as string (text) and not as number in untyped languages. This means, that an explicit conversion is needed. For example:
- Python: `a = int(input("a:"))`
- JavaScript: `const a = parseInt(prompt("a: "));`

Break

10 minutes

Your task:
Using cast operators, test if a given float value has decimals.

In groups of two, write a program in C, C# or Python that checks if a value (given as a float) does not have a fractional part (is integer). Time: 15 minutes.

Then, we discuss together.

One possible approach to the task (Python)

```
a = 5.1 # float
b = int(a) # integer

if (a == b):
  print('no frac part')
else:
  print('frac part')
```

Control flow

Part 2

Control flow

- The execution path of a program is determined by decisions. The decisions are based on conditions that are checked during the program's runtime.
- Great benefits of software is the flexible control flow allowing
  - choosing different paths depending on a state of the program (**selection**), and
  - repeating tasks many times (**iteration**).
- Programming languages contain statements for selection and iteration.

Selection

Most programming languages contain the following kind of selection statements:

- **if**...else if... if : check a condition and branch.
- **switch** : choose one out of many options.

‘If’ example in C, JavaScript and C#

```
if (x > 0) {
    y = 1;
}
else if (x < 0) {    /* you can have none or many ‘else if’
    y = -1;
}
else {
    y = 0;
}
```

‘If’ example in Python

```
if x > 0:
    y = 1
elif x < 0:
    y = -1
else:
    y = 0
```

‘switch’ example in C, JavaScript and C#

```
switch (x) {
    case 0:
        y = 0;
        break;
    case 1:
        y = 1;
        break;
    default:
        y = 2;
        break;
}
```

Attention: without the ’break’, the program would continue with the code following (the next case).

‘match’ example in Python (version > 3.10)

```
match x:
    case 0:
        y = 0
    case 1:
        y = 1
    case _: #default
        y = 2
```

Iterations (or loops)

Iteration / loop statements specify how often a block of code is executed. A loop can be executed a number of times, and this number is given. Or a loop is executed as long as a condition is true. The most common iterations statements are:

- **for:** repeat a given block a certain number of times
- **while**: repeat a given block as long as a certain condition is true

'for'/'while' example in C, JavaScript and C#

```
/* i, y, z all initialised = 0 before */
for (i = 0; i < 10; i++) { /* i++ is the same as i = i + 1 */
  y = y + 1;              /* y = y + 1 is the same as y++ */
}
while (y > 0) {
  y = y - 1;              /* same as y-- */
  z = z + 2;              /* same as z+= 2 */
}                                          /* z is 20 hereafter */
```

‘for’/’while’ example in Python

```python
y = 0
z = 0
for i in range(10): # for any in sequence, range gives a sequence
    y = y + 1

while y > 0:
    y = y - 1
    z = z + 2
```

Your task:
Write a program using iteration

In groups of two, write a program in at least two languages of C, C#, JavaScript or Python that computes the sum of all integer numbers from 1 to n using a loop (do not use the arithmetic sequence formular) . The variable n shall be defined in the code. Time: 25 minutes.

One possible approach to the task (C#)

```
int n = 10;
int sum = 0;
for (int i = 1; i <= n; i++){
  sum = sum + i;
  Console.WriteLine("" + sum);
}
```

Boolean algebra

Logic and combinations

Boolean algebra

- Boolean algebra is a branch of algebra.
- Its variables are binary, representing logical values: true=1 and false=0.
- And its logical operators are NOT (negation, first precedence), AND (conjunction, second precedence) and OR (disjunction, third precedence).
- Common symbols are NOT (~, ¬), AND (·, ∧), OR(+, ∨).
- It is used for **combining conditions**.

Boolean algebra: truth table

A	B	NOT A	A AND B	A OR B
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

- The truth table shows the results of the operations, e.g., the result if A = 0 and B = 0 then A OR B = 0.
- Other operations can be built. Popular operations are XOR (exclusive OR) = (A+B).(~A + ~B), NAND (NOT AND) = ~(A.B), XNOR (exclusive NOT OR) = (A+~B).(~A+B).

Boolean algebra: laws

Law	OR form	AND form
Identity	$A+0 = A$	$A.1 = A$
Idempotence	$A+A = A$	$A.A = A$
Annihlation	$A+1 = 1$	$A.0 = 0$
Commutativity	$A+B = B+A$	$A.B = B.A$
Associativity	$A+(B+C) = (A+B)+C$	$A.(B.C) = (A.B).C$
Distributivity	$A+B.C = (A+B).(A+C)$	$A.(B+C) = A.B + A.C$
De Morgan	$\sim(A+B) = \sim A \,.\, \sim B$	$\sim(A.B) = \sim A + \sim B$
Complementation	$A+ \sim A = 1$	$A. \sim A = 0$
Involution	$\sim(\sim A) = A$	
Involution	$\sim A + \sim B = \sim(A+B)$	$\sim A \,.\, \sim B = A.B$

Your task:
Truth table for XOR, NAND, XNOR

In groups of two, develop the truth table for the operations. Time: 15 minutes.

Then, we discuss together.

Truth table: XOR, NAND, XNOR

A	B	A XOR B	A NAND B	A XNOR B
0	0	0	1	1
0	1	1	1	0
1	0	1	1	0
1	1	0	0	1

Combining conditions in C, C# and JavaScript

- These languages use ‘!’ for logical not, ‘&&’ for logical and and ‘||’ for logical or of conditions, e.g.,

```
if ((x > 0) || !(y < 0)) {
/* ... */
}
```

- Though the priority of this logical operators is lower than of the arithmetic ones, it is always a good idea to use brackets .

Combining conditions in Python

- Python uses the keywords *not*, *and*, *or:*

```
if (x < 0) and not (y > 0):
    print ("something")
```

Functions

Encapsulate blocks

Functions

- Functions encapsulate code blocks so that these blocks can be reused. The make a program modular and manageable.
- A **function** has a **name**. It can have **arguments** as input and a **return** value.
- A functions has its an own scope of variables. This means, that a variable introduced in the function is not visible to its outside.

Example: functions in C and C#

```
/* return value is int, name is second_binomial, arguments are a of type integer and b of type integer */
int second_binomial(int a, int b) {
  int twice_a_b = 2*a*b;      /* temporary variable/not necessary*/
  return a*a+ twice_a_b +b*b; /* return keyword for return value */
} /* a, b, twice_a_b only known in the function */

…
int r = second_binomial(4, 3); /* call the function */
```

Example: functions in JavaScript

```
function secondBinomial(a, b) {
  const twiceAB = 2*a*b;
  return a*a + twiceAB + b*b;
}

const r = secondBinomial(4, 3);
```

Example: functions in Python

```
def second_binomial(a, b):
    twice_a_b = 2*a*b
    return a*a+ twice_a_b + b*b

r = second_binomial(4, 3)
```

Recursive functions

- A function that calls itself to solve a problem is called **recursive**. It needs to have a base case (to avoid stack overflow, comes next week) and a recursive case that builds on the base case.

Example of recursive function in Python

```
def factorial(n):
    # Base case: if n is 0 or 1, return 1
    if n == 0 or n == 1:
        return 1
    else:
    # Recursive case: n * factorial of (n-1)
        return n * factorial(n - 1)

# Example usage
print(factorial(5)) # Output: 120
```

Your task:
Write a program using recursion

In groups of two, write a program in in at least two languages of C, C#, JavaScript or Python that computes the sum of all numbers from 0 to n using recursion. The variable n shall be defined in the code. Time: 25 minutes.

One possible approach in Python

```
def sum_nb(nb):
 if nb==1:
  return 1
 else:
  return nb+sum_nb(nb-1)

print(sum_nb(10))
```

Today’s summary

- Variables continued: primitive data types
- Control flow continued: selection and loops
- Boolean algebra
- Functions and recursion

Questions ?

Generated with AI