ZNOTES.ORG

UPDATED TO 2017 SYLLABUS

CAIE AS LEVEL COMPUTER SCIENCE (9618)

SUMMARIZED NOTES ON THE PRACTICAL SYLLABUS

1. Algorithm Design & Problem-Solving

- **Abstraction:** filtering out and concentrating on the relevant information in a problem; allowing a programmer to deal with complexity
- **Decomposition:** breaking down problems into sub-problems in order to understand a process more clearly; program modules, procedures and functions all help the programmer to break down large problems
- **Algorithm:** a solution to a problem expressed as a sequence of steps

1.2. Identifier Table

- **Identifier:** name given to a variable in order to call it
- An identifier table depicts information about the variable, e.g.

Identifier	Data Type	Description
`NumberDrawn`	`ARRAY[50] : BOOLEAN`	FALSE – indicates the number has not yet been drawn TRUE – indicates the number has been drawn
`Index`	`INTEGER`	used as the subscript/index for the array

- **Rules for naming identifiers:**
  - Must be unique
  - Spaces must not be used
  - Must begin with a letter of the alphabet
  - Consist only of a mixture of letters and digits and the underscore character '_'
  - Must not be a 'reserved' word – e.g. Print, If, etc.

1.3. Basic Program Operations

- **Assignment:** an instruction in a program that places a value into a specified variable
- **Sequence:** programming statements are executed consequently, as they appear in the program
- **Selection:** control structure in which there is a test to decide if certain instructions are executed
  - **IF selection:** testing 2 possible outcomes
  - **CASE selection:** testing more than 2 outcomes
- **Repetition/Iteration:** control structure in which a group of statements is executed repeatedly
  - **FOR loop:** *count-controlled*; executed a set no. of times
  - **WHILE loop:** *pre-conditional*; executed based on condition at start of statements
  - **REPEAT loop:** *post-conditional*; executed based on condition at end of statements

As for selecting what loop to use, it is best to use FOR loops when you know the number of iterations required, and a WHILE or REPEAT loop if you do not know the number of iterations required.

- Iterate over an array: FOR Loop
- Reading a file into a variable: WHILE Loop
- Asking for user input: WHILE/REPEAT Loop
- A loop that should execute n times: FOR Loop

1.4. Stepwise Refinement

- Process of developing a modular design by splitting a problem into smaller sub-tasks, which themselves are repeatedly split into even smaller sub-tasks until each is just one element of the final program.

1.5. Program Modules

- This refers to a modular program design
- **Subroutines:** self-contained section of code, performing a specific task; part of the main program
- **Procedures:** performs a specific task, no value returned to part of code where called
- **Functions:** performs a specific task, returns a value to part of code where called

1.6. Logic Statements

Operator	Meaning
<	Less than
<=	Less than/equal
>	Greater than
>=	Greater/equal
==	Equal to
!=	Not equal to

2. Data Representation

2.1. Data Types

Integer:

- Positive or negative number; no fractional part
- Held in pure binary for processing and storage
- Some languages differentiate short/long integers (more bytes used to store long integers)

Real:

- Number that contains a decimal point
- Referred to as singles and doubles depending upon number of bytes used to store

Character:

- A character is any letter, number, punctuation or space

- Takes up a single unit of storage (usually a byte).

String:

- Combination of alphanumeric characters enclosed in " "
- Each character stored in one byte using ASCII code
- Each character stored in two bytes using Unicode
- Max length of a string limited by available memory.
- Incorrect to store dates or numbers as strings
- Phone no. must be stored as string else initial 0 lost

Boolean:

- Can store one of only two values; "True" or "False"
- Stored in 1 byte: True = 11111111, False = 00000000

Date:

- Dates are stored as a 'serial' number
- Equates to number of seconds elapsed since 1st January 1970 00:00:00 UTC, excluding leap seconds.
- Usually take 8 bytes of storage
- Displayed as dd/mm/yyyy or mm/dd/yyyy

Array:

- Data structure consisting of a collection of elements
- Identified by at least one array index (or key)

File:

- Object that stores data, information, settings or commands
- Can be opened, saved, deleted & moved
- Transferrable across network connections

2.2. ASCII Code

- Uses 1 byte to store a character
- 7 bits available to store data and 8th bit is a check digit
- 2^7 = 128, therefore 128 different values
- ASCII values can take many forms: numbers, letters (capitals and lower case are separate), punctuation, non-printing commands (enter, escape, F1)

2.3. Unicode

- ASCII allows few number of characters; good for English
- Unicode allows others too: Chinese, Greek, Arabic etc.
- Different types of Unicode:
  - UTF-8: compatible with ASCII, variable-width encoding can expand to 16, 24, 32, 40, 48
  - UTF-16: 16-bit, variable-width encoding can expand to 32 bits
  - UTF-32: 32 bit, fixed-width encoding, each character exactly 32 bits

2.4. Arrays

- 1-Dimensional (1D) Array: declared using a single index, can be represented as a list

Index	Element
[0]	JONES
[1]	ERICSON
[2]	WILLIAMS
[3]	ADAMS
[4]	JOHAL

- 2-Dimensional (2D) Array: declared using two indices, can be represented as a table

Indexes	[0]	[1]	[2]
[0]	JONES	F	PENN'S
[1]	ERICSON	M	LAMBOURNE
[2]	WILLIAMS	M	PENN'S
[3]	ADAMS	F	CASWALL'S
[4]	JOHAL	M	SWALLOW'S

- Pseudocode:
  - 1-D Array: `array = []`
  - 2-D Array: `array = [[], [], [], …]`
- Python:
  - Declaring an array: `names = []`
  - Adding to an array: `names.append('ZNotes')`
  - Length of array i.e. number of elements: `len(names)`
  - Printing an element in a 1D array: `print(names[element position])`
  - Printing element in a 2D array: `print (a[row][column])`
  - Printing row in a 2D array: `names[row] = [new row]`
  - Printing column: use for loop and keep adding 1 to the row and keep column same

2.5. Bubble Sort

```
BubbleSort( int a[], int n)
Begin
     for i = 1 to n-1
          sorted = true
          for j = 0 to n-1-i
               if a[j] > a[j+1]
                    temp = a[j]
                    a[j] = a[j+1]
                    a[j+1] = temp
                    sorted = false
          end for
          if sorted
               break from i loop
     end for
End |
```

- A FOR loop is set to stop the sort
- Setting a variable 'sorted' to be 'true' at the beginning
- Another FOR loop is set up next in order to search through the array
- An IF is used to see if the first number of the array is greater than the second. If true:
  - First number stored to variable
  - Second number assigned as first number
  - Stored variable assigned to second number
  - Set 'sorted' to 'false' causing loop to start again
- The second FOR loop is count based thus will stop after a specific number of times
- Goes through bigger FOR loop ∴ 'sorted' remains 'true'
- This exits the loop ∴ ending the program

2.6. Linear Search

```
For each item in the list:
     if that item has the desired value:
          stop the search and return the item's location.
Return the item is not in the list
```

- A FOR loop goes through the array
- It compares item in question to those in list using an IF:
  - If item matches with another then search is stopped
  - Also the location where it was found is returned
  - If not found it exits the FOR loop
- Then returns fact that item in question is not in the list

2.7. File Handling

- Files are needed to import contents (from a file) saved in secondary memory into the program, or to save the output of a program (in a file) into secondary memory, so that it is available for future use

Pseudocode:

- Opening a file: `OPENFILE <filename> FOR READ/WRITE/APPEND`
- Reading a file: `READFILE <filename>`
- Writing a line of text to the file: `WRITEFILE <filename>, <string>`
- Closing a file: `CLOSEFILE`
- Testing for end of the file: `EOF()`

Python:

- Opening a file: `variable = open("filename", "mode")`

Where the mode can be:

Mode	Description
r	Opens file for reading only. Pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites file if file exists or creates new file if it doesn't
a	Opens a file for appending. Pointer at end of file if it exists or creates a new file if not

- Reading a file:
  - Read all characters: `variable.read()`
  - Read each line and store as list: `variable.readlines()`
- Writing to a file:
  - Write a fixed a sequence of characters to file: `variable.write("Text")`
  - Write a list of string to file: `variable.write('\'.join('Z', 'Notes'))`

Abstract Data Types (ADT)

An **Abstract Data Type (ADT)** is a collection of data with associated operations. There are three types of ADTs:

- **Stack:** an ordered collection of items where the addition of new items and removal of existing items always takes place at the same end.
- **Queue:** a linear structure which follows the First In First Out (FIFO) mechanism. Items are added at one end (called the rear) and removed from the other end (called the front)
- **Linked List:** a linear collection of data elements whose order is not given by physical placements in memory (non-contiguous). Each element ***points*** to the next.

3. Programming

- Programming is a transferable skill
- **Transferable skill:** skills developed in one situation which can be transferred to another situation.

3.2. Variables

- **Declaring a variable:**

- Pseudocode: "'python DECLARE <identifier> : <data type> "'
- Python: no need to declare however must write above as a comment ("'python #..."')

- **Assigning variables:**

  "'python <identifier> ← <value>"' or "'python <expression>"'

"'python identifier = value"' or "'python expression"' or "'python "string""'

3.3. Selections

- "IF" Statement
  - Pseudocode: `IF...THEN...ELSE...ENDIF`
  - Python: `if (expression): (statements) else: (statements)`
- "CASE" Statement
  - Pseudocode: `CASE OF variable: ... ... ... OTHERWISE: ... ENDCASE`
  - Python: `if (expression): (statement) elif (expression): statement) ... else: (statement)`

3.4. Iterations

Count-controlled Loop	
FOR <identifier> ← <val1> TO <val2> STEP <val3> <statement(s)> ENDFOR	
for x in range(value1, value2): statement(s)	
Post condition Loop	
REPEAT <statement(s)> UNTIL <condition>	Not possible in Python Use "'python WHILE"' and "'python IF"'
Pre-condition Loop	
WHILE <condition> <statement(s)> ENDWHILE	while expression: statement(s)

3.5. Built-in Functions

String/character manipulation:

- Uppercase or lowercase all characters: `("string").upper()` `("string").lower()`
- Finding length of a string: `len("string")`
- Converting:
  - String to Integer - `int("string")`
  - Integer to String - `str(integer)`

Random number generator: `random.randint(a, b)`
Where `a` and `b` defines the range

3.6. Benefits of Procedures and Functions:

- Lines of code can be re-used; don't have to be repeated
- Can be tested/improved independently of program
- Easy to share procedures/functions with other programs
- Create routines that can be called like built-in command

3.7. Procedure

Procedure: subroutine that performs a specific task without returning a value

- **Procedure without parameters:**

PROCEDURE <statement(s)>ENDPROCEDURE	def identifier():statement(s)

- When a procedure has a parameter, the function can either pass it by either reference or value
- **Pass by value:** data copied into procedure so variable not changed outside procedure

```
PROCEDURE <identifier> (BYVALUE <param>: <datatype>)
<statement(s)>
ENDPROCEDURE
def identifier(param):
statement(s)
```

- **Pass by reference:** link to variable provided so variable changed after going through procedure *(not in Python)*

```
PROCEDURE <identifier> (BYREF <param>: <datatype>)
<statement(s)>
ENDPROCEDURE
```

- **Calling a procedure:**

CALL ()	Identifier()

3.8. Function

Function: subroutine that performs a specific task and returns a value
Functions are best used to avoid having repeating blocks of code in a program, as well as increasing the reusability of code in a large program.

```
FUNCTION <identifier> (<parameter>: <data type>) RETURNS <datatype>
<statement(s)>
ENDFUNCTION
def identifier(param):
statement(s)
return expression
```

4. Software Development

4.1. Program Development Cycle

- **Analyze problem:** define problem, record program specifications and recognize inputs, process, output & UI
- **Design program:** develop logic plan, write algorithm in e.g. pseudocode or flowchart and test solution
- **Code program:** translate algorithm into high level language with comments/remarks and produce user interface with executable processes
- **Test and debug program:** test program using test data, find and correct any errors and ensure results are correct
- **Formalize solution:** review program code, revise internal documentation and create end-user documentation
- **Maintain program:** provide education and support to end-user, correct any bugs and modify if user requests

There are three different development life cycles:

- **Waterfall model:** a classical model, used to create a system with a linear approach, from one stage to another
- **Iterative model:** a initial representation starts with a small subset, which becomes more complex over time until the system is complete
- **Rapid Application Development (RAD) model:** a prototyping model, with no (or less) specific planning put into it. More emphasis on development and producing a product-prototype.

4.2. Integrated Development Environment

- A software application that allows the creation of a program e.g. Python
- Consists of a source code editor, build automation tools, a debugger

Coding:

- Reserved words are used by it as command prompts
- Listed in the end-user documentation of IDE
- A series of files consisting of preprogrammed-subroutines may also be provided by the IDE

Initial Error Detection:

- The IDE executes the code & initial error detection carried out by compiler/interpreter doing the following:
  - Syntax/Logic Error: before program is run, an error message warns the user about this
  - Runtime Error: run of the program ends in an error

Debugging:

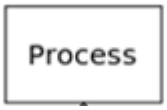
- Single stepping: traces through each line of code and steps into procedures. Allows you to view the effect of each statement on variables
- Breakpoints: set within code; program stops temporarily to check that it is operating correctly up to that point
- Variable dumps (report window): at specific parts of program, variable values shown for comparison

4.3. Structure Charts

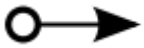
- **Purpose:** used in structured programming to arrange program modules, each module represented by a box
- Tree structure visualizes relationships between modules, showing data transfer between modules using arrows.
- Example of a top-down design where a problem (program) is broken into its components.

Rules:

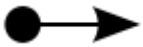
- **Process:** Represents a programming module e.g. a calculation



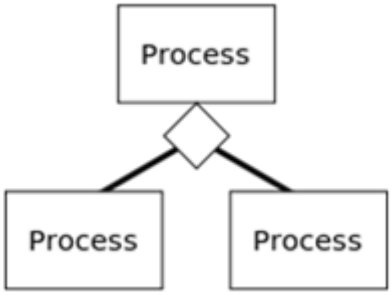
- **Data couple:** Data being passed from module to module that needs to be processed



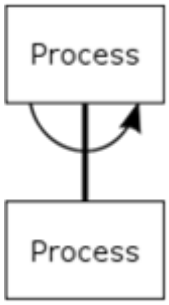
- **Flag:** Check data sent to start or stop a process. E.g. check if data sent in the correct format



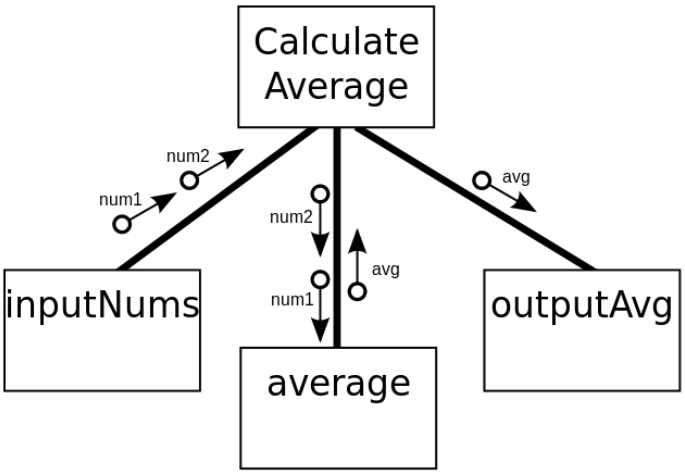
- **Selection:** Condition will be checked and depending on the result, different modules will be executed



- **Iteration:** Implies that module is executed multiple times



Example:



4.4. Types of Errors

Syntax errors:

- When source code does not obey rules of the language
- Compiler generates error messages
- Examples:
  - Misspell identifier when calling it
  - Missing punctuation – colon after if
  - Incorrectly using a built-in function
  - Argument being made does not match data type

Run-time errors:

- Source code compiles to machine code but fails upon execution *(red lines show up in Python)*
- When the program keeps running and you have to kill it manually
- Examples:
  - Division by 0
  - Infinite loop – will not produce error message, program will just not stop until forced to

Logic errors:

- Program works but gives incorrect output
- Examples:
  - Out By One – when '>' is used instead of '>='
  - Misuse of logic operators

4.5. Corrective Maintenance

- Corrective Maintenance is correcting identified errors
- White-Box testing: making sample data and running it through a trace table
- Trace table: technique used to test algorithms; make sure that no logical errors occur e.g.

Line	largest	x	value[x]> largest	output
1	38	-		
2		2		
3			false	

4.6. Adaptive Maintenance

- Making amendments to:
  - Parameters: due to changes in specification
  - Logic: to enhance functionality or more faster or both
  - Design: to make it more user friendly

4.7. Testing Strategies

Black box testing:

- Use test data for which results already calculated & compare result from program with expected results
- Testing only considers input and output and the code is viewed as being in a 'black box'

White box testing:

- Examine each line of code for correct logic and accuracy.
- May record value of variables after each line of code
- Every possible condition must be tested

Stub testing:

- Stubs are computer programs that act as temporary replacement for a called module and give the same output as the actual product or software.
- Important when code is not completed however must be tested so modules are replaced by stubs

Dry run testing:

- A process where code is manually traced, without any software used
- The value of a variable is manually followed to check whether it is used and updated as expected
- Used to identify logic errors, but not execution errors

Walkthrough testing:

- A test where the code is reviewed carefully by the developer's peers, managers, team members, etc.
- It is used to gather useful feedback to further develop the code.

Integration testing:

- Taking modules that have been tested on individually and testing on them combined together

- This method allows all the code snippets to integrate with each other, making the program work.

Alpha testing:

- This is the testing done on software 'in-house', meaning it is done by the developers
- Basically another term for 'first round of testing'

Beta testing:

- This is the testing done on the software by beta users, who use the program and report any problems back to the developer.
- Basically another term for 'second round of testing'

Acceptance testing:

- A test carried out by the intended users of the system: the people who requested the software.
- The purpose is to check that the software performs exactly as required.
- The acceptance criteria should completely be satisfied for the program to be released.

CAIE AS LEVEL
Computer Science (9618)