Information Technologies II

BSTI2202 (Biannual)
BTTI1002 (Quarterly)
Teaching notes

UNIVERSIDAD TECMILENIO

Tools

In order for you to make the most of your educational experience in this type of certificates, we recommend that you review these **tutorials**.

Module 1

Introduction

Algorithms are usually solutions to problems or procedures to achieve a goal, they are independent of the entity that will put these procedures or solutions into action.

In computer science, digital computers end up developing these actions and executing these programs, therefore, during the design of an algorithm it is important to understand the way in which digital computers understand, process and execute the algorithm instructions.

A digital computer is not as versatile as a human being, although there are areas of knowledge that seek to provide computers with these reasoning capabilities, the most common is to take advantage of speed and precision when performing basic operations to compensate for the lack of reasoning and deduction.

Teaching notes for the teacher that gives the topic:

1. Iterations

An iteration is a repetition within a cycle that must be executed a defined number of times.

A digital computer is able to reason, deduce, infer or generate new knowledge as a human being would, however, digital computers are able to perform logical operations much more quickly and accurately than human beings, therefore, computational algorithms seek to take advantage of this by breaking down complex problems into much simpler subproblems, even when these subproblems are tedious and repetitive.

For example, is a human is asked to calculate the answer of $4 \times 6 \times 2$, he, depending on his experience, can use his great memory and his deducing and reasoning capability to solve it the following way:

By his memory, he knows that $4 \times 6 = 24$ and the double is 48.

Meanwhile, for a computer it is not posible neither to store all the multiplication tables in memory nor is it efficient to look for the result in the same memory.

If trying to mimic human behavior, the computer would need to look up the result of 4×6 in its memory and then look up the result of 24×2. This strategy would force

the computer to have stored all possible multiplication of numbers, which is impossible.

So, the programmer chooses to take advantage of the digital computer by breaking down the action of multiplying into simpler operations:

$$4 \times 6 = 4 + 4 + 4 + 4 + 4 + 4 = 24$$

$$24 \times 2 = 24 + 24 = 48$$

The areas of knowledge, such as computational intelligence, automatical learning and deep learning, among others, seek to generate procedures or methodologies so that a digital computer becomes able to imitate human reasoning or any other entity in nature.

Although this procedure requires six sums and for a human being this can be tedious and inefficient, it would take a digital computer a fraction of a second to perform them. Digital computers are capable of performing logical operations in billionths of a second.

In fact, the frequency of processors, often considered wrong as the speed of the processor, indicates the time it takes for the computer to perform a basic operation. For example, for a 2 GHz processor it takes $\frac{1}{2\times10^9} = 5 \times 10^{-10}$ seconds to performa a basic operation.

It can be mentioned that the basic operations of a digital computer are much simpler than an addition, in fact, the operation of adding two numbers breaks down into several even simpler operations, which will be discussed in point 7.

So, it is clear that repetitive and iterative procedures are a basic part of any algorithm implemented in a digital computer. The ability to understand the general problem and break it down into several subactions is acquired with experience and practice, so it is important to insist, with the learners, on always looking for ways to acquire new knowledge that will allow them to simplify their codes, for example, in activity "Level of Practice: Be Attentive, *Tengshe*" it is possible to identify a pattern in the actions to be carried out.

This screen was directly obtained from the software that is being explained in the computer for educational ends.

However, to express an iterative procedure that executes the same actions as the code shown, different combinations are required in the conditions that trigger each action, which implies the use of compound conditionals (topic 6), so the code shown will produce the most efficient solution that can be generated so far, but after concluding topic 6 they will have a tool for making more effcient codes and this will happen continuously during their professional lives, that is, the ability to use iterations is in constant development.

Teaching notes for the teacher that gives the topic:

2. Loops and iterations with conditionals *IF*

The loop is the base structure to create an iterative algorithm. The loop refers to the set of instructions necessary to delimit the procedures to be repeated and establish the conditions to continue or stop the repetition or iteration.

Iteration conditions can be as simple as completing a perfectly defined number of iterations or as complex as combining consequences within the same procedure.

For example, back to multiplication, the condition that tells us how many times to repeat the addition process is a count from 1 to the value of one of the multiplicands.

Pseudocode to solve the multiplication of the integer numbers a and b

$$a \times b \; a, b \in \mathbb{Z}$$

Define the variable “counter” and assign it an initial value of 1:
counter = 1
Define the variable “result” and assign it an initial value of 0:
result = 0
Assign the label “start” at the beginning of the program:
start:
Perform the sum of the value stored in “result” and the value of a:
result = result + a
Increase in one the value of the “counter”:
counter = counter + 1
Verify if the value of the counter is equal to b, if it is, show the result:
if counter = b
print (result)
If not, go back to start:
else goto start

The algorithm,

```
counter = 1
result = 0
start:
result = result + a
counter = counter + 1
if counter = b:
        print (result)
else:
        goto start
```

represents a loop that will repeat the procedure (result + a) b times; b is the number of iterations of the loop and the “if, else” conditionals correspond to the iteration conditions.

This procedure is the simplest and most basic to achieve a loop that is usually known as **iterations with if conditionals**, since it is possible to generate a loop from only if,else conditionals.

These structures are so common in digital computer programs that instructions or procedures that automate or simplify the implementation of loops have been generated.

2.1 Loop of cycle *for* as a conditional if

The first procedure implemented in Python and many other languages to generate loops is the cycle for. Depending on the programming language, the syntax, that is, the way of writing the instruction, changes, although they all share the basic elements of a loop:

a. The counter of iterations.
b. The condition/conditions of iteration.
c. The procedure of instruccions to repeat.

The structure of a cycle for in Python is as follows: it starts with the "for" instruction, followed by the iteration condition ending in a colon ":", on a different line or lines and indented by a tab. the instructions to repeat are placed.

for “iteration condition”:
    procedure to repeat 1
    procedure to repeat 2
    procedure to repeat n

Indentation
The indentation of a tab refers to the space between the column where the for instruction and the statements to be repeated within the loop are located. The tab is a defined space between words generated by the Tab key on the keyboard.

This separation between the columns of the code blocks that make up a program is called indentation, which serves to facilitate visual identification of functional blocks of code in a program for the programmer or user. Today, it is also used for the same purpose by programming languages.

This screen was directly obtained from the software that is being explained in the computer for educational ends.

Iteration condition.

The part of the condition of iteration in the cycle allows you to repeat the procedure for each element contained in a set, for example, the code

```
for i in [1, 3, 5, 7, 9]:
   print(i, end=", ")
```

Show the numbers 1, 3, 5, 7 and 9 separated by a comma. Any procedure, command or parto f the program that generates a set of elements can be used as an iteration condition in a cycle for, for example:

```
for i in [Orange, Coconut, Tangerine, Apple]:
   print(i, end=", ")
```

List the four fruits separated by a comma.

One of the most used commands to generate the set of elements that control the iterations of the cycle for is “range()”. This instruction generates a set of numbers from a numerical sequence, where we can define the beginning, the end and the increment in this sequence. For example:

```
for i in range(5):
    print(i, end=", ")
```

displays the sequence of numbers 0, 1, 2, 3, 4. When only one parameter is used in the “range()” instruction, Python interprets it to be a unit increment and starts from 0. A second parameter defines the start of the range() statement succession:

```
for i in range(-2, 5):
    print(i, end=", ")
```

displays -2, -1, 0, 1, 2, 3, 4. A third parameter defines the increment or step of the sucession:

```
for i in range(-2, 5, 2):
    print(i, end=", ")
```

displays -2, 0, 2, 4.
Once this is understoof, it is easy to generate the program to calculate a multiplication from sums[2]:

```
For i in range (a):
        result = result + a
```

This generates the result $a \cdot a^1$.

2.2 The cycle or loop *while*

Another structure that allows generating iterative procedures is the while loop, which historically is also known as "do while" due to the syntax used in the first programming languages.

The "while" instruction allows you to repeat a procedure or series of instructions, **while** a condition is met or not, for example, the code:

```
day = 0
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
while day < 7:
  print("Today is " + week[day])
  day += 1
```

Displays the next message:

Today is Monday
Today is Tuesday
Today is Wednesday
Today is Thursday
Today is Friday
Today is Saturday
Today is Sunday

If someone wants to check the code of multiplication as addition, you can visit: https://www.programiz.com/python-programming/online-compiler/ and run the following code:

```
a = 10
result = 0
for i in range (a):
        result = result+ a
print(result)
```

The instruction “day += 1” increments the day value by 1. This example is useful to recommend not to use special characters when defining variables; the variable “dia” could have been named “día” more correctly, however, this may make it difficult for colleagues from other countries to interpret our code, or worse, the compiler may not identify the symbol “í” and mark an error that would take us a little time to identify and correct.

In the case of character strings, such as "Wednesday", there is no problem, since it is a sequence of symbols that the language will only display without the need to interpret it.

Back to the “while” instructiont, we can see that the syntax is very similar to that of a cycle for: instruction followed by the iteration condition ending in a colon.

```
while day < 7:
```

And the block of instructions to repeat is placed one tab of distance from the instruction “while”.

2.3 Differences between while and for

As mentioned before, it is possible to obtain the same result with both instructions with enough imagination and creativity, however, the "while" instruction is designed to define more complex iteration conditions, since greater than, less than and equality conditions that can be used, together with the basic logical operators of union, intersection and negation (described in topic 6), which gives compound conditionals. While the for loop is designed to work with lists, vectors, databases, etc., that is, data arrays or data sets.

As mentioned before, it is possible to generate a procedure with any of the two instructions, one of them will provide a smaller or more efficient code, computationally speaking, and it is the job of the programmer to be able to identify that operational advantage in his implementations, although it is not always usually clear and it usually depends a lot on the experience and skills of the programmer.

Teaching notes for the teacher that gives the topic:

3. Nesting

Nesting is a strategy to simplify or compact codes that require the repetition of repetitions. This strategy does not offer any computational efficiency when executing the instructions, but it does offer efficiency in lines of code, which allows the program to be stored in less memory and consume less computing resources when compiling.

It also offers a visual advantage for experienced programmers, making it easier for different members of a team to understand the programs.

The strategy is very simple, it consists of placing a loop (for or while) inside another loop (for or while) respecting the syntax of these cycles and establishing the hierarchy of the procedures based on the indentation:

```
Loop1 condition1:
        Loop2 condition2:
                Instructions for repeating the loop 2.
        Instructions for repeating the loop 1.
```

An example that illustrates the advantages of nesting is the algorithm that displays the multiplication tables from 1 to 10. Commonly, we would make the code that

makes the table of 1, another code that makes the table of 2 and so on until the table of 10:

```
for j in range(1, 11):
        print(1 * j, end=' ')

for j in range(1, 11):
        print(2 * j, end=' ')
.
.
.
for j in range(1, 11):
        print(10 * j, end=' ')
```

However, it is evident that it consists on a repetition of cycles for, so, if we use the nesting strategy, the resultant code is:

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i * j, end=' ')
    print()
```

Both codes, writing the first one complete, produce the same result:

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

And, strictly talking, both codes do the same number of operations, however, it is clear that the second one is more compact and, with a little experience, it is even easier to understand.

The Ozaria platform provides several more visual and intuitive examples.

This screen was directly obtained from the software that is being explained in the computer for educational ends.

Teaching notes for the teacher that gives the topic:

4. Hybrid nestings

Something very logical is to ask if it is possible to use while loops within for loops or vice versa, and the answer is yes, in fact, this type of strategy is known as **hybrid nesting**.

Using conditionals, while or for loops within a different conditional or loop is very common and useful when generating code that meets our requirements.

Teaching notes for the teacher that gives the topic:

5. Applications

The applications of iterative processes and all their variants are very diverse and are perfectly illustrated in the activities of the platform, that is why they will not be discussed here.

Module 2

Introduction

The way in which the human being represents information, stores it and uses it for decision making depends directly on both their needs and the tools they have to do so. Initially, the human being only had the need to quantify the food that he collected, for which, with integer numbers it was more than enough.

Later, their culture evolved and man invented the concept of sharing or lending, thus the need to represent a debt gave rise to negative numbers, while the ability to cut or divide food gave rise to the need to use fractional numbers.

During all this time, the human used his fingers to quantify, generating a **decimal numeration**. To facilitate the way of representing quantities, the human being developed a series of symbols or digits (0, 1, 2,..., 9) that, depending on the position of each symbol within a quantity, represents its value; the first position represents the units, the second position is the tenths, and so on.

This type of numbering or way of representing quantities is known as **digital numbering**, since digits are used in different positions to represent the desired information.

In a parallel way, the philosophers, mainly the Greeks, began trying to understand the different concepts and elements of human nature, among several currents, Aristotle began to break down human nature into basic truths that could only be true or false, this philosophical current laid the foundations for the processing of information represented by quantities that can only take two values: true or false.

We had to wait until the appearance of the vacuum tube and, later, the transistor, so that the foundations laid by philosophical thought were used in the representation and processing of information, giving rise to the **digital** age.

The term digital is associated, today, in a very simplistic way with the representation of information from binary digits, this is due to the fact that current computing systems are mostly **binary digital**, that is, they represent the information in the form of digits that can take only the value 0 or 1.

The first functional electronic computer (ENIAC) was a decimal digital computer, which represented information with numbers from 0 to 9, it was the Hungarian-American mathematician John von Neumann who realized the advantage of

representing quantities in binary form and participated in the design and manufacture of the first binary electronic-digital computer, the MARK 1.

The reason for which Neumann decided that it was more efficient to represent quantities in binary form was, and still is, because of the way binary digital computers are built. The basic processing units of a binary digital computer were vacuum tubes and today transistors, which can be visualized as switches that allow or not the flow of electrons through them.

Therefore, a transistor can have a value of 0 if it does not allow the passage of electrons and 1 if it allows it. The great combination of 0 and 1 in different positions allows current computers to represent and process information massively and very quickly, which is known as the digital age, or more properly, the **binary digital** age.

Teaching notes for the teacher that gives the topic:

6. Compound conditionals

It is very natural to use all the reasoning tools developed by ancient philosophers about binary premises for understanding why the information inside a computer is represented from 0 and 1.

Ancient philosophers developed a whole methodology of logical reasoning, structured from simple premises and logical combinations that were later applied to computer science.

Within computer science, the IF conditionals represent the basic operation for decision making, the premises or syllogisms, and the combination of these conditionals from logical operators allows the generation of much more complex functions, in fact, the microprocessor of a computer is capable of generating the most complex imaginable function, from combinations of only three basic logical operations.

The **basic logical operations** are union, intersection, and negation, each of which produces a binary result from the combination of basic conditionals.

The intersection

The logical **intersection** operation is more commonly known as the “AND” operator and basically fulfills the function indicated by its grammatical use. For example, a mother may condition her son's attendance at a party if he completes his homework; another mother may condition attendance if he cleans his room.

If you do your homework = you go to the party.
If you clean your room = you attend the party.

A stricter mom can use the intersection operation to generate a compound conditional:

If you do your homework AND clean your room = yo go to the party.

In this case, the son must successfully complete both actions to attend. Failing one or both of them will cause him to not be able to attend the party.

This reasoning can be expressed in a more graphic and easier way through a **truth table**, for which we will take the following considerations:

Let us call the premise “If you do your homework” as input “a” and “you clean your room” as input “b”, so the consequence “go to the party” is generated from the intersection of both inputs “a&b”. A value of 0 represents false, that is, not meeting any of the conditions or not having authorization to attend the party, while 1 is true and implies having met the condition or being able to attend the party.

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

The union

The logical **union** operation is known as the “OR” operator and basically fulfills the function indicated by its grammatical use. Again, using the example of attending the party, a more compliant mother can state the following compound condition:

If you do your homework or clean your room = you attend the party.

Thus, the son will be able to attend the party if he meets any of the two conditions, and even both. The only situation in which he will not be able to attend is if he does not comply with any of them.

The truth table for the union is as follows:

a	b	a o b
0	0	0
0	1	1
1	0	1
1	1	1

Negation

The negation is the only logical operation that affects only one condition and basically inverts the value of this condition. For example, the condition “if you clean your room” changes to the condition if you don't clean your room, so the state that was 0 is now 1 and vice versa.

The negation of a premise is represented by using a line on the premise $\overline{clean} = not\ clean$ or with an appostrophe $clean' = not\ clean.$

a	a'
0	1
1	0

Part of the confusion can be due to the incorrect identification of the parts of a sentence, for example, the sentence "if you don't clean your room, you don't come to the party" has a double negative, however, these negatives are in different parts of the sentence.

The first one is in the conditional and the second in the consequence, therefore there is no ambiguity in its interpretation.

Teaching notes for the teacher that gives the topic:

7. Binary logic

Once the basic logical operations are known, it is possible to generate combinations between them to control the behavior of a system or obtain a specific wanted output, for example, consider the electric cutter illustrated below.

For safety, the machine internal digital computer must not allow the blade to work when the operator's fingers are in the path of the blade, so the designer adds a pair of buttons on each end labeled as “cut”, so the blade will be activated if button 1 **&** button 2 are activated, forcing the user to use both hands to activate the buttons and thus ensure that they are not in the path of the blade.

We can make this activation condition more complex to make the system safer, adding a switch that indicates whether the transparent cover is up or not, so our premises will be a = button 1 pressed, b = button 2 pressed and c = cover up: this way the blade will activate if button 1 is pressed & button 2 is pressed & the cover is not up.

All this description can be expressed more compactly through a logic function.

Logical function and truth table

A logical function is the set of logical conditions that generate a true consequence, for example, the logical function that expresses the combinations necessary to activate the blade in the previous example is a&b&c', or more compactly, abc’.

Another way to represent the behavior of a logical system is through its truth table, which is a set of rows and columns where all the possible values that the conditions can take and the corresponding consequences for these combinations in the conditionals are expressed.

The truth table for the cutter example is as follows:

a	b	c	a&b&c'

0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

The procedure for generating a truth table is as follows:

1. Generate the table with the possible combinations for the *n* conditions involved in our logic function.

For the example of the cutter n = 3 conditions, the number of possible combinations is calculated with the formula:

combinations $=2^n$, for n = 3 we have $8 = 2^3$ combinations

The easiest way to generate all the possible combinations is to generate $8 = 2^3$ zeros followed by 2^{n-1} ones in the first column. The second column alternates 2^{n-1} zeros with 2^{n-1} ones a couple of times. This process is repeated for 2^{n-3}, 2^{n-4}, etc. until reaching the 2^0 limit, where it will alternate between just 1 zero and 1 one.

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

In the next column we add the first operation a&b which, for simplicity, can be expressed just as ab:

a	b	c	ab

0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Then, we generate the second c' operation:

a	b	c	a&b	c'
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

Finally, we evaluate the last operation: a&b&c'.

a	b	c	a&b	c'	abc'
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	0

Teaching notes for the teacher that gives the topic:

8. Boolean algebra

Boolean algebra is a series of theorems that allows changing the form of expressing a logical function by an equivalent function. This equivalent function may simply be different, or it may have some kind of advantage, such as reducing the number of logical operations, reducing the number of variables, or simply isolating a term to assess its importance in the function.

Equivalent function

Two functions are said to be equivalent when they both share the same maxterms.

Maxterms

The maxterms of a function are the combinations of individual logical conditions that produce a true output, for example, the maxterms of the function “a or b” are ab’ + a’b + ab, with a’b + ab = a’b or ab.

a	b	a+b	
0	0	0	a’b’
0	1	1	a’b
1	0	1	ab’
1	1	1	Ab

As a simple illustrative example, we Will generate the truth table for the function ab’ + a’b + ab.

a	b	ab’	a’b	ab	ab’ + a’b + ab
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	0	0	1	1

As you can see, each term of the function generates a unique true output and there are no repeated terms, therefore, the number of maxterms in which a function can be decomposed is equal to the number of times that the original function takes the value of 1.

Now we can say that the functions a+b and ab' + a'b + ab are equivalent, or a+b = ab' + a'b + ab, therefore, there must be a series of theorems and procedures that allow to pass from one expression to another. These theorems and properties form the basis of Boolean algebra and are listed below:

First, Bolean algebra must satisfy the properties of:

1. Commutativity.

The order of the operands does not alter the:

$$x+y=y+x \qquad xy=yx$$

2. Neutral element.

There is one element that does not alter the operation:

$$x+0=x \qquad x*1=x$$

3. Distributivity.

$$x(y+z)=xy+xz \qquad x+yz=(x+y)(x+z)$$

4. Asociativity.

$$x+(y+z)=(x+y)+z \qquad x(yz)=(xy)z$$

5. Complementary element.

The operation with the complement results in the neutral element.

$$x+x'=1 \qquad xx'=0$$

Some of the properties give, as a consequence, the following **theorems**:

Theorem 1. Idempotence

The operation of this term ends with the same result in the original term.

$$x+x=x \qquad x*x=x$$

Demonstration:

x	x	x+x
0	0	0
1	1	1

x	x	x*x
0	0	0
1	1	1

Look that, even when there are two operands, both are the same so, we can $2^1 = 2$ combinations.

Theorem 2. Identity

The operation with the null element (or identity) results in the same null element.

It should be remembered that the null or identity element depends on each operator, thus, for the union operation, the identity element is 1 and for the intersection operation it is 0.

x+1=1 x*0=0

Demonstration:

x	1	x+x
0	1	1
1	1	1

x	0	x*x
0	0	0
1	0	0

Theorem 3. Absorption

When a variable appears in both operands, the result is the same variable.

x+xy=x x*(x+y)=x

Demonstration:

It can be demontrated using the truth table:

x	Y	xy	x+xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

x	Y	x+y	x(x+y)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

We can also start using the theorems before and simplify the functions using algebra:

x+xy = x(1+y) (by idempotence)

x(1+y) = x(1) (by identity)

x(1) = x (by identity)

In a similar way:

x(x+y) = x(1(1+y))

x(1(1+y)) = x(1(1))

x(1(1)) = x

It is very important to mention that, despite the similarity that exists with elementary algebra (the one we commonly know), Boolean algebra has its own rules and theorems that are not always analogous to those of elementary algebra, therefore, the properties which we are used to, are not always valid and we must be careful.

Theorem 4. DeMorgan

The negation of an operation is the same that applying the opposite operator with the negated operands.

(x+y)'=x'y' (xy)'=x'+y'

Demonstration:

x	Y	(x+y)'	x'y'
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

x	Y	(xy)'	x'+y'
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Finally, the use of these properties and theorems allows expressing equivalent functions, as an example, we will return to the case of the maxterms of the OR function.

ab+a'b+ab'

Applying distributivity, we can rewrite the function as:

b(a+a')+ab' = b(1)+ab' = b + ab'

Applying distributivity again, we can rewrite the function as:

b + ab' = (b+a)(b+b') = (b+a)(1) = b + a

All the previous expressions are equivalent and, depending on the purpose, they can be more or less convenient. This is the importance of developing the ability and intuition necessary to identify equivalent terms and the properties or theorems that can be applied for their conversion.

Teaching notes for the teacher that gives the topic:

9. Karnaugh maps

Karnaugh maps are graphical representations where the maxterms of a logical function are located. The fact of being a graphic distribution allows, on many occasions, to visually and much more easily identify the terms that share elements in common and, therefore, it is likely to simplify these terms.

Karnaugh's mapping methodology includes a series of steps and rules that systematize the creation of maps, the identification of simplified terms, and how to simplify them. This methodology, although valid for any number of variables, is mainly applied to functions with 2, 3 and up to 4 variables; For the case of 5 or more variables, the size of the maps and the possible interactions between terms makes it inefficient and makes its application very difficult.

The first step consists in creating the map, this is, generating the graphical distribution of the maxterms. Depending on the number of variables, this distribution is a 2x2, 2x3, or 4x4 matrix, where each position in this matrix represents a possible combination of terms:

	a	a'
B		
b'		

	ab	a'b	a'b'	ab'
C				
c'				

	ab	a'b	a'b'	ab'
Cd				
c'd				
c'd'				
cd'				

Each cell in this arrangement, represents the combination of the terms in the corresponding row and column:

	a	a'
B	ab	a'b
b'	ab'	a'b'

	ab	a'b	a'b'	ab'
C	abc	a'bc	a'b'c	ab'c
c'	abc'	a'bc'	a'b'c'	ab'c'

	ab	a'b	a'b'	ab'
Cd	abcd	a'bcd	a'b'cd	ab'cd
c'd	abc'd	a'bc'd	a'b'c'd	ab'c'd
c'd'	abc'd'	a'bc'd'	a'b'c'd'	ab'c'd'
cd'	abcd'	a'bcd'	a'b'cd'	ab'cd'

Then, the maxterms of the function must be located within this array, for example, the map of the function a'bc'd + a'b'c'd' + a'b'c'd + a'b 'c'd' is as follows:

	1	1	
	1	1	

Finally, sets of 2, 4 or 8 are located within the map and the final result is the common terms of these arrays. For the indicated example, there is a single group of 4 ones and the only common terms between these elements are: a'c', thus, we can say that a'bc'd + a'b' c'd' + a'b'c'd + a'b'c'd' = a'c'.

	1	1	
	1	1	

a'bc'd + a'b'c'd' + a'b'c'd + a'b'c'd'

It is clear that, for the previous example, the common elements can be identified directly from the original function, however, for more complex functions, the graphical distribution helps a lot to identify groups with common terms.

Teaching notes for the teacher that gives the topic:

10. Applications

Once more, the applications about compound conditionals and their simplifications are shown in the Ozaria platform exercises.

Teaching notes for the teacher that gives the topic:

11. Functions

The term **function** is used in several areas of knowledge. A mathematical function establishes a relationship between two sets, while a logical function establishes the logical conditions necessary to have a true result.

In a similar way, within computer science, the term "function" has a very specific definition. Basically, a function is a set of instructions that carry out a specific procedure for a series of perfectly identified parameters or values. Although, within a program this is something very common, the particularity of the function lies in how continuously it is necessary to repeat this entire procedure.

For example, suppose you create a program that stores a user's name, address and age; and eventually it responds with a personalized greeting. The procedure is pretty easy:

Pseudocode:

Start a counter n=1

Create a list type variable named names.

Create a list type variable named addresses

Create a list type variable named ages.

Ask the user for his name and store the input in the names variable at index n.

Ask the user for his address and store the input in the addresses variable at index n.

Ask the user for his age and store the input in the variable ages at index n.

Write “Hello:” names[n]

As you can see, this code is very simple, however, the fact of not knowing the exact number of individuals that are going to be registered makes it difficult to place this code inside a loop to repeat it indefinitely.

Note that it is possible to place it in a while loop, where the repetition condition is to ask the user if there are more pending records, however, let's consider that this program is part of a much larger one, where we can see the number of records up to time, check the data of a specific record, search record by name, etc.

This makes it much more difficult to implement a loop that repeats the registration process every time it is needed. The most logical and intuitive solution is to set aside the specific code for the registry and use it every time it is necessary, this is precisely a function; a name is given to the code set, it is stored separately and it is called when needed.

This significantly reduces the lines of code required to repeat the process over and over again.

The way to generate a function in Python is as follows:

It should start with the "def" command, which tells Python that a function is to be defined. Then, we place the name we want for this function; we must be aware of not duplicating names or using names already defined within the same language.

All the parameters that the user must provide to carry out the procedure are placed inside a parenthesis; the number of parameters directly depends on the purpose of the function.

There may be functions where no parameters are required or functions where several of them are needed. Finally, a colon is placed and in the following lines the instructions that form the function are added.

Do not forget to respect the indentation of belonging. At the end, if the function must return one or several results, the "return" instruction is used, followed by the variable or variables whose values the function must return to.

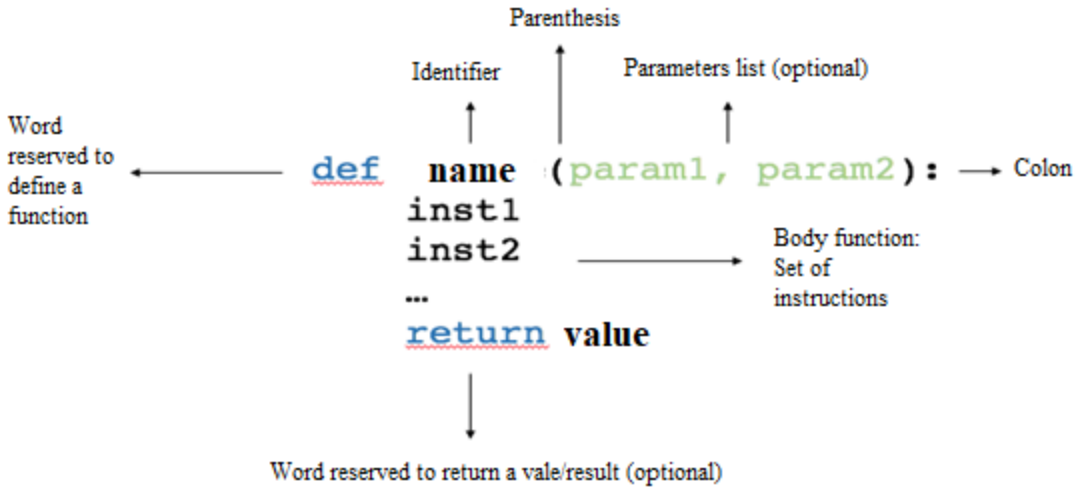


Figure 1. To generate a function in Python.

For example, the jumpArround function created in the Pi jump activity has no input or output parameters and it just makes that "*noodles*" go up, then left and then down:

This screen was directly obtained from the software that is being explained in the computer for educational ends.

Module 3

Introduction

The main reason for learning to program and code in any language is to generate programs and applications that help people to simplify tasks or solve problems, beyond creating new difficulties for them.

Therefore, the way in which the programmer plans that the user is going to interact with his programs is of vital importance and it is worth starting to train young people in criteria and standards of functionality, operability and simplicity that allow them to plan and anticipate the needs, deficiencies and special situations that the user may encounter while using its programs, in order to help, understand and facilitate their experience.

Teaching notes for the teacher that gives the topic:

12. Interaction with the user

Strictly talking, a user interface is any means by which the user can interact with any technological element, thus, this includes both physical and non-physical elements. For example, a keyboard, microphone, or touch screen are physical **hardware** interfaces, while menus, screens, toolbars, etc. are **software** interfaces and, of course, there can always be elements that share both **software-hardware** characteristics, for example, the buttons of an application that interact with the touch screen, but they are designed from the code.

Both types of interfaces are very important for the final product. An ergonomic and functional design in both types of interfaces can go a long way in making a program truly solve a problem or simply become a bigger problem.

Experts in product design, industrial design, marketing, etc. specialize in the physical characteristics of an interface and, although there are also experts in charge of monitoring the ergonomics and functionality of software interfaces, it is always important that a programmer has good design habits that makes this task easier.

It is important to highlight that, for our case, the area of interest is software interfaces and, with this in mind, we consider it necessary for the student to develop empathy with the user, to understand the way in which he is going to interact with his programs and to always think about the users' needs when designing his interface, looking for making it ergonomic and functional.

Teaching notes for the teacher that gives the topic:

13. User interface

The user interface or UI is the set of instructions, functions and procedures generated specifically to interact with the user. Although a program can be perfectly generated to function and fulfill its purpose, it is possible that very specialized knowledge is required, either on the specific subject of the program, or that it requires mastery of programming concepts.

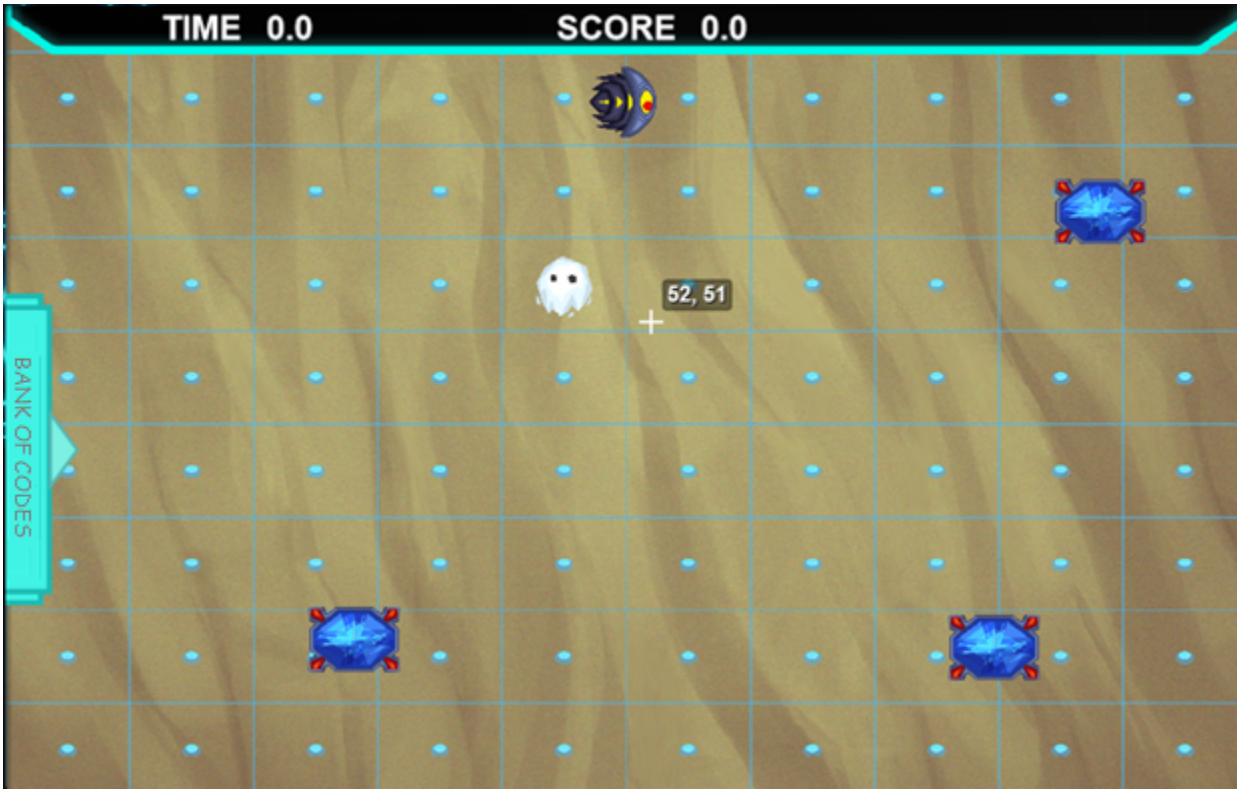
It is the responsibility of the programmer to understand the level of knowledge and mastery of concepts, both of the topics of the program and of programming, as well as of the sector of the people to which his program is directed and to do everything possible to correct or reduce the possible deficiencies of the user.

For example, nowadays, it is not necessary for a person to know about augmented reality or artificial vision, much less about the memory or processor of their device so that they can use the filters with their cell phone camera and have fun without worrying about the lighting, exposure time, GPU capacity, etc. This is thanks to the

fact that the programmers understood all this and simplified the process for the user.

Specifically, the user interface is all the instructions that receive or provide information from or to the user. Nowadays, since human beings have a much greater affinity for visual and striking elements, it is very common to do this interaction through graphical elements so, several programs have **a graphical user interface or GUI**, for its acronym.

In the final challenge of this course, the student will develop his own graphical interface using the tools generated by the supplier. Here is another example of a good interface, which makes the process much easier and allows the student to focus only on the main part:



This screen was directly obtained from the software that is being explained in the computer for educational ends.

Clarity

A good interface gives information accurately to prevent the user from making interpretation errors during the interaction.

Conciseness

A good interface provides only the information the user needs and does not distract the user with useless or irrelevant information.

Coherence

This feature is what makes an interface intuitive, that is, if an action generated a response under one situation, it is expected that the same action will generate a similar reaction in other circumstances, allowing the person to create patterns of use in a simple and practical way.

Flexibility

It is to allow the user to modify or personalize the way of interacting. It helps a lot to reduce the learning time that the user requires to learn how to use our program.

Visual appeal

It is always important to have a balance between visual details and conciseness. Visuals that are too flashy can draw the user's attention to unimportant details and distract them from elements that do require their attention.

Although a GUI is very useful to simplify the interaction with the user, it is not the only way to interact with it, in addition to the fact that generating interactive graphics in a programming language is usually one of the most difficult skills to acquire, since it requires a lot of time and experience.

Teaching notes for the teacher that gives the topic:

14. Ergonomics and functionality

In August 2000, the Council of the International Ergonomics Association (IEA) agreed on a definition that has been adopted as “official” by many entities, institutions and standardized organisms.

For the specific Mexican case in the official Mexican standard NOM-036-1-STPS-2018:

“Ergonomics (or the study of human factors) is the scientific discipline that deals with the **interactions between humans and other elements of a system**, as well as the profession that applies theory, principles, data, and methods to design in order to optimize the well-being of the human beings and the overall result of the system” (Official Diary of the Federation, 2018).

Ergonomics takes into consideration physical, cognitive, social, organizational and environmental factors, but with a "holistic" approach, in which each of these factors should not be analyzed in isolation, but rather in their interaction with the others.

For the specific case of computer systems, ergonomics focuses on the physical characteristics and capabilities of human beings who interact with computer systems. These human characteristics can be divided into three main groups:

1. **Physical limitations:** not all users are physically equal. For example, if the interface is designed only for tall or right-handed individuals, many will have trouble using it and will be affected.
2. **Cognitive skills:** Not all users are experts or have high cognitive skills, so it should not be necessary to handle a lot of information to use an interface. For example, many systems use the term address instead of URL for making it easy to understand.
3. **Emotional needs:** the interface must not confuse the user, neither about how to start using it nor when an error occurs. Confusion leads to frustration, which eventually turns into stress, producing long-term emotional damage.

Due to these characteristics, the biggest challenges facing ergonomics are related to the diversity of users. Some aspects are quite controllable and measurable, such as age and habits, but others are very difficult to manage. People's expectations and their level of knowledge are varied and their research is complex.

This variety in potential users also makes it difficult to assess the ergonomics of a system or interface, that is why specialized tools have been developed, such as user tests and usability tools, in order to assess and quantify the benefits of a system.

It is clear that when talking about aesthetic or visual issues there are subjectivities that can make it difficult to design a good UI, especially when working in interdisciplinary or intercultural teams.

Therefore, some experts have taken on the task of defining international and global parameters on good practices to develop user interfaces, perhaps the best known and followed standard is the ISO 9241 norm, which focuses on the ergonomics of the interaction between the person and the system, specifically, in aspects such as ease of communication and dynamism.

It is worth mentioning that these standards are not subjective or dictated arbitrarily, they are the result of research, studies and field tests of the different elements that can make up a UI.

Within a web page, application, program or any digital tool, the design of the user interface must guarantee that it is able to adapt to the task, have error tolerance, customize, provide enough level of control, adapt to learning, be self-descriptive and, above all, conform to user expectations.

The design of a user interface is helped by many other areas of knowledge, for example, graphic design, psychology, etc.

Therefore, a good UI respects the principles of consistency and graphic quality, such as color codes to support user tasks, minimalism, distribution, Gestalt and standardization.

Teaching notes for the teacher that gives the topic:

15. Final product

Finally, the conjunction of algorithms and user interfaces generate the final product. Although the success of this depends on many variables beyond the control of designers and programmers, there are basic elements that must be respected to increase the probability of success.

Consistency

The design must be consistent throughout the application, both in the sequences of actions and in the terminologies and conventions of the platform. This helps people recognize elements and understand their hierarchy and usefulness.

Efficiency

The interface must allow efficient use of the user's time, loading and presenting the content within an acceptable time. The longer you make someone wait, the more stress builds up on them.

There are certain ranges for the actions, each one communicates to the user that something different is happening. You should also include features for advanced users, for example, accelerators or shortcuts.

Design

Ideally, the design should be attractive. This helps the user to consume the data and minimize stress, achieving the “feel good” effect. The principles of contrast, repetition, alignment and proximity are natural to humans and are felt as familiar.

Memory

The memory usage of users should be minimized. All information needed to perform a task should be presented or required in a simple way.

Contextual Help

Help resources should be visible and always available, without interrupting navigation.

Structure and space

The interface must consider the position of the hands, the mobility of the fingers and the visual field of the users on desktop computers, tablets and smartphones. Also, buttons, fields, and dropdowns should be easy to activate across devices.

In addition, it is important to offer visual *feedback* on the status of the elements, indicating if they are active, pressed, selected or loading, among others. This informs the user that the system has received their order and is generating a response.

One of the keys to place technology at the service of people is that interfaces are designed thinking about them. All the functionalities and advantages offered by the Internet and digital platforms cannot be accurately used if users are not able to use the sites optimally.

To conclude, review on the Canvas platform the methodology, the syllabus and the evaluation to which the delivery of each challenge corresponds, as well as the course policies for the semester or four-month modality; there is a tab for each within Evaluation.

Also, from the Canvas platform there is the Challenges section, where the instructions for each of them will be reviewed, as well as their deliverables.

Grades will be uploaded on the Canvas platform.

Teaching notes for the teacher that gives the tournament (only biannual):

Below are the instructions for the tournament:

Objective

Promote the learning and application of advanced technologies in solving mathematical problems, while encouraging collaboration, creativity, and a competitive spirit among students.

Participant selection

The top participant from each group (or the top two participants from each group, in the case of smaller campuses) in the Final Challenge of the Information Technologies II class will be selected to participate in the tournament.

Before the tournament

- Students have the opportunity to make improvements to their applications before the tournament.
- Teachers must evaluate the final challenge to select the student(s) who will represent the group.

- The campus, through the Leader of Teachers, will define the teachers who will form the panel of judges.

Tournament development

- Participating students will present their applications to a panel of judges composed of Technology and Mathematics teachers, selected by the campus.
- Each presentation will include a live demonstration of the application, an explanation of the development process, and a question-and-answer session.

Evaluation criteria

- It will be evaluated using the same rubric as the Final Challenge: Functionality, Robustness, Versatility, User interface, and Aesthetic aspect.

Awards

- The student who achieves the highest score will be the winner of the campus tournament and will receive a digital badge, along with recognition on the campus's social media (with prior authorization for image use).

To learn about the methodology, dates of implementation of the final challenge and tournament (only biannual), as well as the instructions, click **here**.

Bibliographical references

Diario Oficial de la Federación. (2018). *NORMA Oficial Mexicana NOM-036-1-STPS*. Recuperado de https://dof.gob.mx/nota_detalle.php?codigo=5544579&fecha=23/11/2018#gsc.tab=0