Chapter 4 Recap
===============

Please note that the following are mainly points with some brief explanations to add to the notes you are compiling. The recap is an opportunity to revisit the work we have spoken about during class.

==
Overview
==

Chapter highlights:
-- Boolean expressions and the selection structure
-- The relational comparison operators
-- AND logic
-- OR logic
-- NOT logic
-- Making selections within ranges
-- Precedence when combining AND & OR operators

==
N.B. The following recap and associated notes are compiled from multiple sources. Some of the work mentioned in the following sections has been discussed in the previous chapter notes including the homework that you have been completing. Also, that some of the examples discussed in class closely resemble the content shared. Do your best to compare and relate the notes so that you have a much clearer picture of the content.
==

Boolean expressions and the selection structure
==

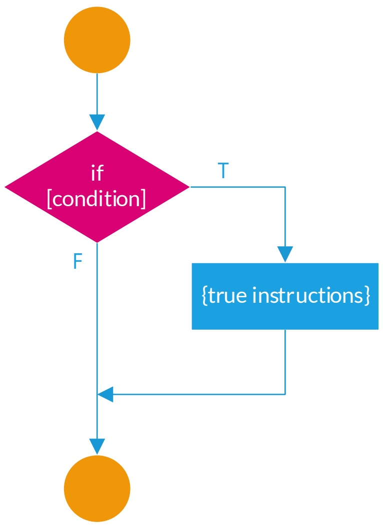
-- Boolean expressions - logical expressions - can only result in the values of true or false
-- The selection structure uses these types of expressions to process & evaluate information making a choice to decide if a set of instructions are executed or not.

-- Decision Logic / Selection Structure - is the same ...

-- ***Single alternative (unary) selection structure***
- if..then statement - Action is provided for only one outcome (if the result of the condition is true)

** *pseudocode*
```
if [condition] then
        {true statements}
endif
```

** *flowchart*



- In the above template code, the if..then statement only executes instructions if the (*result of the*) condition is true, if the (*result of the*) condition is false, no instructions within the if..then statement is executed. The program will continue processing after the endif command.

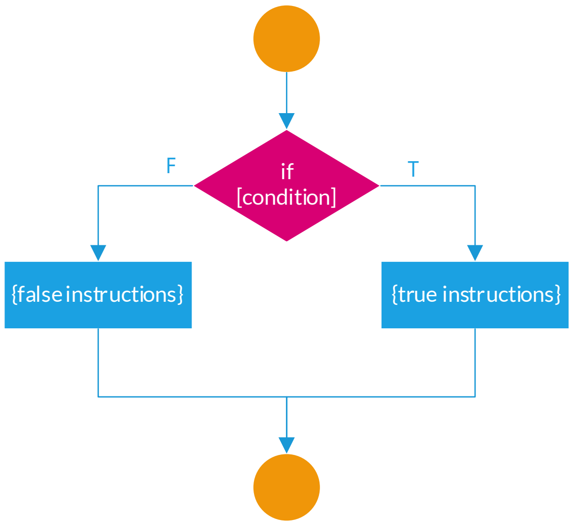
-- ***Dual alternative selection structure***

- if..then else - Provides an action for each of 2 possible outcomes

** *pseudocode*

```
if [condition] then
        {true statements}
else
        {false statements}
endif
```

** *flowchart*



- In the above template code for the if..then statement, if the (*result of the*) condition is true, then the instructions/statements coded in the true section will be executed and if the (*result of the*) condition is false, then the statements written after the else will be executed.

- Because of the introduction of the else 'block', this statement is able to deal with both possible outcomes of the condition in the if..then statement.

* Again, the decision logic structure tells the computer that 'if' the result of the condition is true, 'then' execute a set of instructions, or 'else' execute another set of instructions. *The 'else' part is optional, since if a program's logic does not require instructions to be executed when the condition is false, then it is not coded.*

==

The textbook presents an example (*Figure 4-3*) that utilizes modules and loops to show the if..then statement at work. Again, while, this is appreciated in the progress of the textbook, we will take it step-by-step.

We will for now, continue developing programs (sequential logic and now decision logic) within one main program. Later, re-introduce the lessons around modules, by going back to program and learning why and how to split a program into modules (*procedures and functions*).

And as we progress into the Chapter about loops, we will then again, revisit these examples shared in the textbook to better understand how a complete program is developed and the reasons why loops were a part of the programs in these previous chapters.

==

Here is the example that we had discussed in class and without the use of modules and loops.

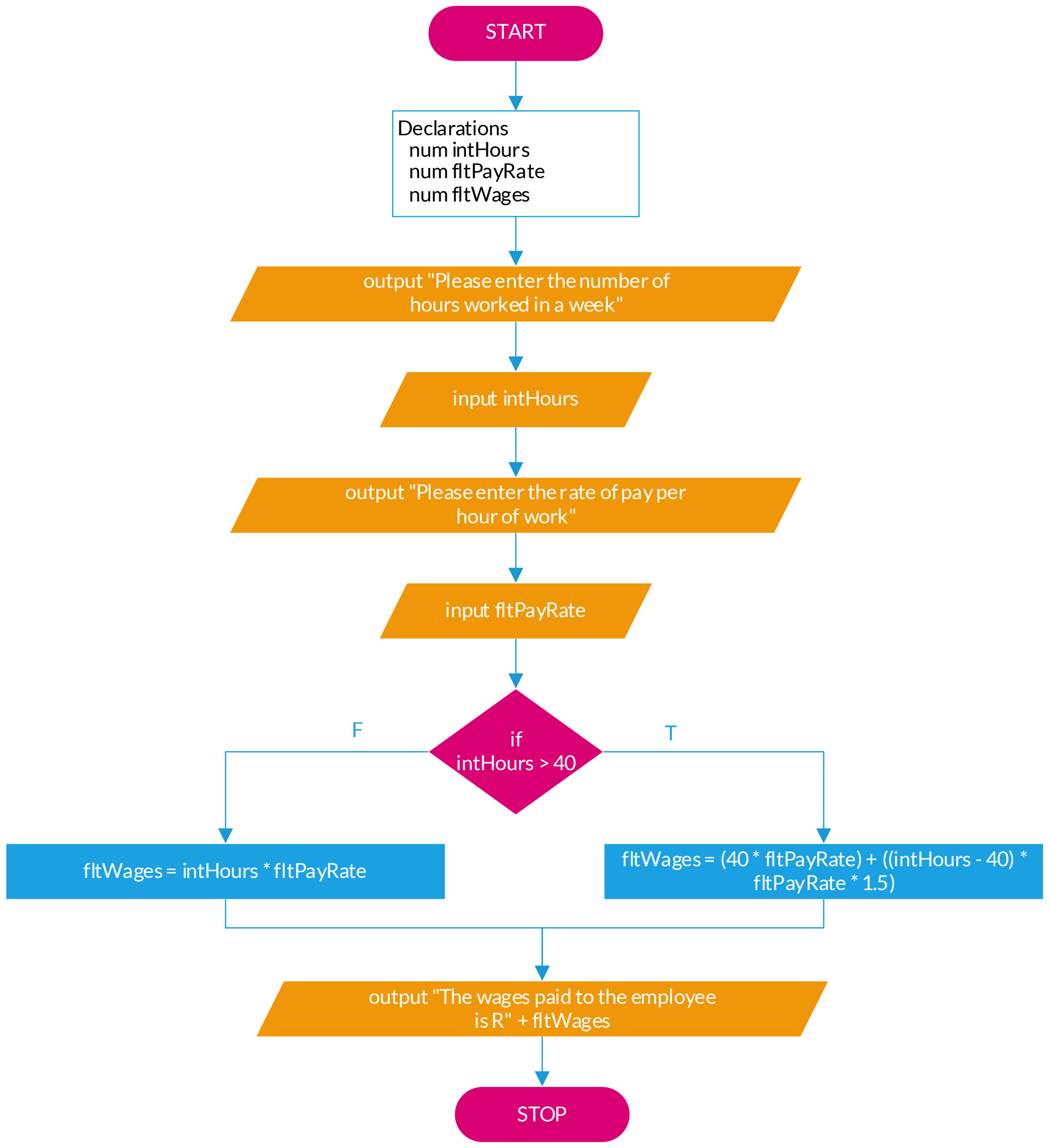
Calculate and display the wages paid to an employee based on the number of hours worked in a week and the rate of pay per hour. If the employee works for more than 40 hours in the work week (which is the allowable amount of time for the week), then they must be paid 1.5 times the rate of pay for the overtime hours worked. If the employee has worked up to 40 hours, then they could be paid a normal pay per hour.

** *pseudocode*

```
0. Start
1. Declarations
        num intHours
        num fltPayRate
        num fltWages

2.      output "Please enter the number of hours worked in a week"
3.      input intHours

4.      output "Please enter the rate of pay per hour of work"
5.      input fltPayRate

6.      if intHours > 40 then
                fltWages = (40 * fltPayRate) + ((intHours - 40) * fltPayRate * 1.5)
        else
                fltWages = intHours * fltPayRate
        endif

7.      output "The wages paid to the employee is R" + fltWages

8. Stop
```

Small note: Please note that none of the code within the if..then statement are assigned instruction numbers. The code is indented to help read the program and to realise that all the code for the if..then statement is one entire instruction.

** *flowchart*


START
Declarations
num intHours
num fltPayRate
num fltWages
output "Please enter the number of hours worked in a week"
input intHours
output "Please enter the rate of pay per hour of work"
input fltPayRate
if intHours > 40
F
T
fltWages = intHours * fltPayRate
fltWages = (40 * fltPayRate) + ((intHours - 40) * fltPayRate * 1.5)
output "The wages paid to the employee is R" + fltWages
STOP

Conditions [Logical and/or Boolean Expression]
==

In the following sections, the notes are about the use of the relational & logical operators. It is important that we have a clear understanding that these sections are an opportunity to learn how to develop the logical expressions needed for the conditions in an if..then statement.

The condition can be one of 4 variations:

-- 1 -- Relational Expression

- Expression that uses relational operators: <, >, >=, <=, =, and <>
- Operands can be either numerical, character or the string data type
- Both operands are of the same data type

- *intHours > 40*
- *fltSalary >= 15000*
- *intNumber1 > intNumber2*

-- 2 -- Logical / Boolean Expression

- Expression uses logical/boolean operators: NOT, AND, OR
- Operands are of a boolean data type

- *blnCheckRegistration AND blnCheckVerificationEmail*

-- 3 -- Mix of mathematical and/or relational and/or logical operators

- Since the result of a condition is required to be either true or false, we know from the order of precedence of operators that a mix of operators will still produce a logical answer, because mathematical operators will be dealt with first, then relational and lastly logical.

- *T + 10 <= W (T and W are numeric data types)*
- *R < 90 OR S > 100 (R and S are numeric data types)*
- *intHours - 40 > 0 (intHours is a numeric data type)*

- *Side Note – (repeat) even though we know the order of operators,* ***making use of brackets within an expression does help to better read the expression***
- ***(intMark >= 90) AND (intMark <= 100)***

-- 4 -- A variable of a boolean data type

- Since a condition must produce a result of either true or false & a variable that is of a boolean data type must contains either true or false, then we may create an expression that is only just the variable

```
        if blnLoginStatus then
                {true instructions}
        else
                {false instructions}
        endif
```

- In the above statement - if the variable contains the value of 'true' (*then the result of the condition is true*), then the instructions that are coded in the true section will be executed and if the variable had the value of false (*then the result of the condition is false*), then the instructions in the false section would have been executed.

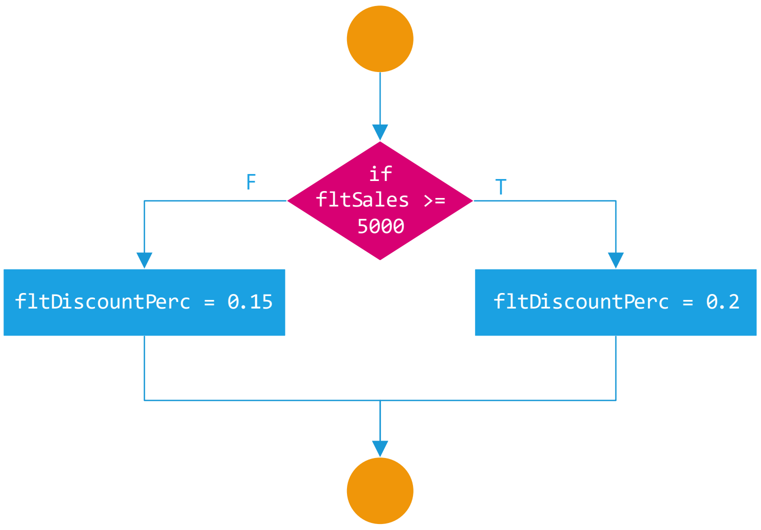
The relational comparison operators
===

```
-- Relational operators
        - Introduced in the - recap of Chapter 2 - notes, along with all other operators
                -- < less than,
                -- > greater than,
                -- <= less than or equal to,
                -- >= greater than or equal to,
                -- = equal to
                -- <> not equal to

        - Used to build logical expressions

        - An important reminder of the data types of the operands when dealing with these types of
         operators (operands can be either, integer, float, character, or string)
        - Relational operators will always produce a logical result (true or false)
        - The use of the operators will depend on the requirements of a questions posed
        - We need to use the operators efficiently
        - Test data will be used to evaluate our choices so that we are able to see all possible
        outcomes of our condition
```

        ** *pseudocode*

```
        if fltSales >= 5000 then
                fltDiscountPerc = 0.2
        else
                fltDiscountPerc = 0.15
        endif
```

        ** *flowchart*



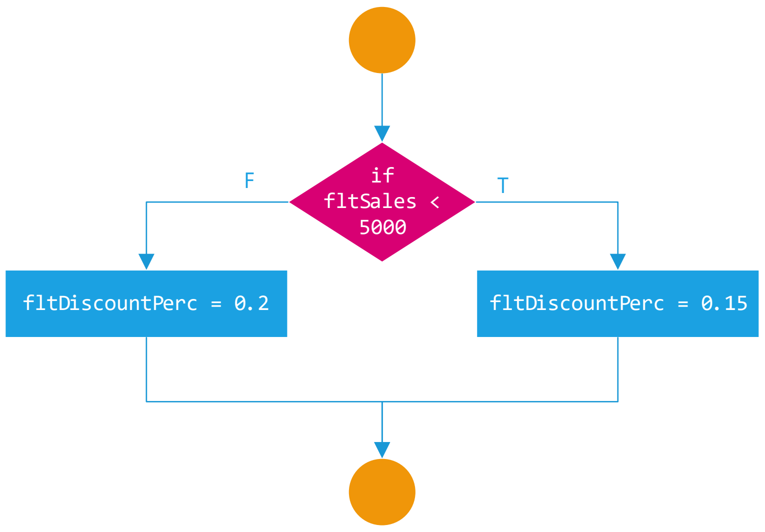
- Thinking in the **negative** is a version of the code that we are going to have practice as we develop more programs. We are more use to the idea of reading a question and developing a condition, looking at the positives. But sometimes it is important to consider that thinking of the negative is a better choice for the efficiency of the program.

- Consider the above code - there is a 20% discount awarded to a user if they make a sale of R5000 or more. But what if after investigations of the data (interviews with the client), we learn that the 20% discount is not consistently awarded. Then we would consider the following:

** *pseudocode*

```
if fltSales < 5000 then
        fltDiscountPerc = 0.15
else
        fltDiscountPerc = 0.2
endif
```

** *flowchart*



- This is going to be a choice you as a developer is going to have to make when developing solutions. Even though both versions are correct, when this piece of code is executed, we will recognize the time it takes to process information using one method or the other.

- Just to offer a little more that was shared in the slides/textbook

Example: A major retail store offers a discount on the sales amount for a customer depending on their customer code. If a customer is assigned the customer code '1' then a 50% discount is given and if they do not have a code, then a 25% discount is applied.

***EDIT** – I wanted to add a little more to the story*
- If the customer is assigned a code of '1', then they are given a 50% discount and if they are assigned a code of '0', then they are awarded a 25% discount.

** *pseudocode*
- separate sequential (*straight-through logic*) if..then statements

```
4.      if intCustomerCode = 1 then
                fltDiscount = 0.5
        endif

5.      if intCustomerCode = 0 then
                fltDiscount = 0.25
        endif
```

- In the above solution, if the customer code is '1' or '0' then they receive a discount, but there is no discount awarded for a customer without a code

- Another **_*edit_** to the story - Let's say that there are only 2 codes given to customers. Customers without a code are not allowed to shop at this store.

- The following if..then statement would be a better option - the code is a little simpler and efficient

4.
```
if intCustomerCode = 1 then
        fltDiscount = 0.5
else
        fltDiscount = 0.25
endif
```

- In the above solution - if the customer code is not '1', then it is obviously '0' and the else/false section works for the story we have been given

- Now while the above code for the if..then/else statement is decent, there may be variations based on how we interpret the story. *The possibility in the future is conducting interviews with the managers and owners of the business to get a better understanding of the problem and as a programmer, making the required decisions to get the more efficient solution into production.*

- Take note of the slight differences in the coding of the above statement
- We could have also opted for the following version:

```
if intCustomerCode = 0 then
        fltDiscount = 0.25
else
        fltDiscount = 0.5
endif
```

- Now consider the following (again) - thinking in the negative, is not thinking of the wrong thing, but just changing how we interpret the condition - sometimes the negative comparison is better understood for the development of the program as a whole

```
if intCustomerCode <> 1 then
        fltDiscount = 0.25
else
        fltDiscount = 0.5
endif
```

- It is important to do the required research into the problem and the business to guide us into making the right decisions for the solution when it’s put into production

----
*** Errors we need to be aware of*
-- Deciding on which relational operator to use is going to come with practice and more examples.
-- It is important to investigate and question the problem so that you fully understand your code

-- If we choose the wrong relational operator, it could end up that our solution does not solve the problem correctly. (*Producing logical errors*)

-- ***Example*** - Write a program that selects all employees over the age of 30
- Does this include 30 years of age?
- What is the intention with the condition?
- How is it going to be used?
- Can I investigate further?
- Can I talk to someone who knows?

Is it meant to be?
`if intAge > 30 then` |OR| `if intAge >= 30 then`

-- There are other trivial expressions that sometimes we need to question, ***why***?
-- Testing if a BIG number is greater than a small number (but you already know the answer)
- `if intLargeNumber > intSmallNumber then`

-- or vice-versa
- `if intSmallNumber < intBigNumber then`

-- important to re-iterate the question, what was your intention knowing that the answer to these statements is obvious

----

AND logic
==
-- *Aside from using AND in a logical expression using 2 boolean operands, the following information relates to the idea of using the AND operator to combine (compound) a condition*

-- When we come across multiple conditions that relate to one outcome, then the development of the expression will require us to combine multiple expressions.
-- These expressions could be joined using the AND logical operator

```
        ** expression
        (fltSales >= 5000) AND (blnWholesaleCustomer = true)
```

-- When the AND operator is used, both operands (*and in this case, the expressions on either side of the AND operator*) must be true for the result to be true. If either operand results in being false, then the end result is false. (Remember the notes in recap of Chapter 2 about the AND operator)

-- When we decide to nest multiple decisions into one logical expression, either of the pieces can be first. However, performance can be improved by testing the expressions in a proper order.

-- These can speed up processing time, because if the first operand results in being false, technically there is no need to test the second operand, because the result of the logical expression is going to be false

----
OR logic
==
-- *We may once again use the OR operator in a boolean expression with the operands being boolean variables and taking note that the following information is on how to use OR to combine conditions.*

-- Similar to the above section about the AND operator, we again deal with an opportunity of multiple conditions that are related to an outcome, but in this version, the OR logical operator will result in being true, if either of its operands are true.

-- Speeding up processing time is again considered, and it is the order of the operands that needs to be looked at while using the OR operator. If the first operand results in being true, then the result of the whole expression is true.

```
        ** expression
        (fltSales >= 5000) OR (blnRetailCustomer = true)
```

NOT logic
==
-- Reverses the value of the logical expression & the NOT operator only has one operand

```
        ** expression
        NOT (intAge < 18)
        // will change the result of the expression - if true, then false and vice-versa
```

-- It is difficult to imagine when and where would the NOT operator be useful. It will take practice and a thought that sometimes the logical expression needs to be thought of in a different way in order for the development of the if..then statement (in this case) to be better understood.

```
        ** pseudocode
        if NOT (intEmployeeDepartment = 1) then
                output "This employee is not in Department 1"
        endif

        instead of

        if intEmployeeDepartment <> 1 then
                output "The employee is not in Department 1"
        endif
```

May not be the best use case of the operator, but there is a difference that would be appreciated, if we knew more about the application/development of the solution.

-- When we do choose to use the NOT operator, we are likely to use it on a boolean variable

----
Multiple if..then..else statements
===
-- Within the discussions around decision logic, one program does not always contain just one if..then statement. There are times where we are expected to execute and deal with multiple decisions.

There are 3 that we need to take note of:

1. Straight-through logic – multiple decisions are processed sequentially, one after the other (part of the program). There is no 'else' (quite likely), and this means that if the condition in the if..then statement is false, then the current statement is left, and we immediately move on to the next instruction in the program.

Example: Depending on the age of a person, if they are older than 16 then they are allowed to apply for their learner's license and if they are older than 18 then they are allowed to vote in the elections.

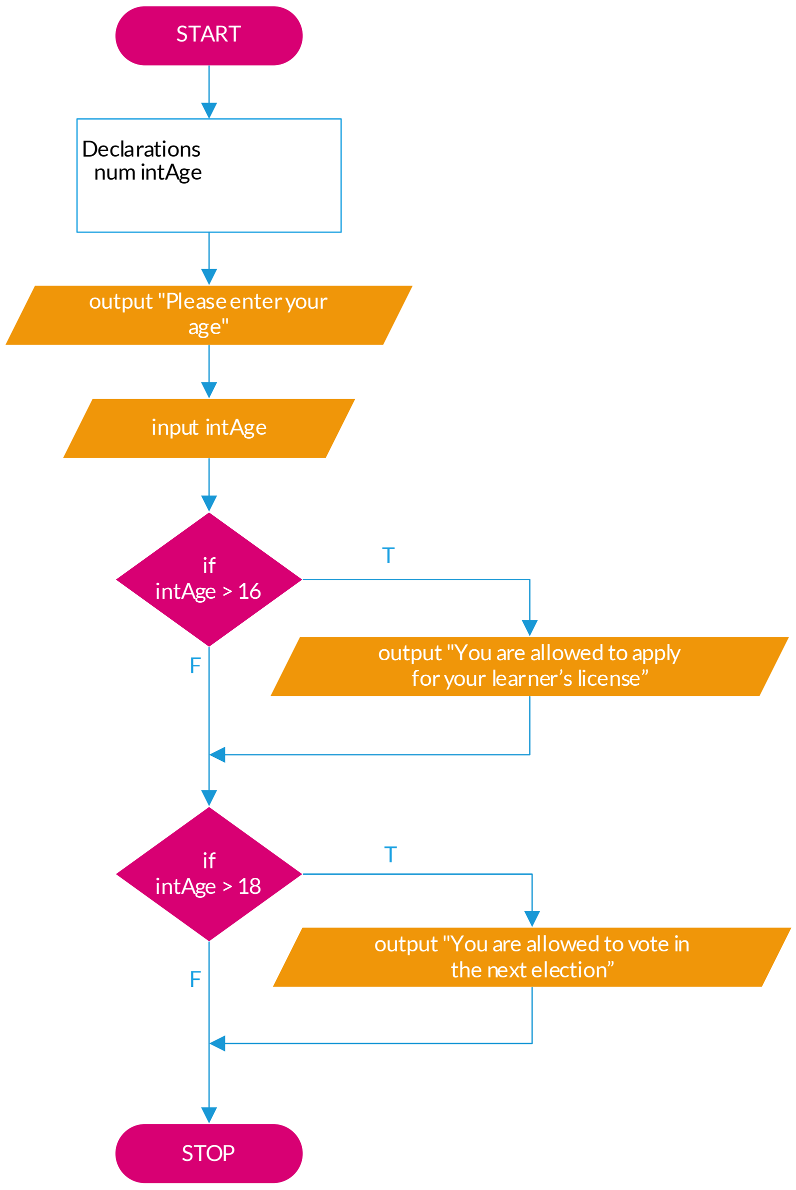
** *pseudocode*

```
0. Start
1. Declarations
        num intAge

2.      output "Please enter your age "
3.      input intAge

4.      if intAge > 16 then
                output "You are allowed to apply for your learner's license"
        endif

5.      if intAge > 18 then
                output "You are allowed to vote in the next election"
        endif

6. Stop
```

* Take note that each coded if..then statement must be executed - This example has been discussed in class, where we assumed that we could have developed the above solution using an *if..then..else statement*, but after discussions, learnt that there are errors in our logic and this example would be better suited to remain as separate if..then statements (*straight-through logic - executed sequentially*)

* N.*B.* Both decisions are contingent on age of the person, however, the 2 resulting messages take place at different ages.

*** flowchart*


START
Declarations
num intAge
output "Please enter your age"
input intAge
if
intAge > 16
T
output "You are allowed to apply for your learner's license"
F
if
intAge > 18
T
output "You are allowed to vote in the next election"
F
STOP

* Another point to repeat, is when there are multiple decisions to be made and we will proceed to code separate if..then statements first, and while the thinking may lean towards a possibility of combining the if..then using an else block, we could come to the realisation that each of the decisions are independent of one another, so best to keep them separate from one another.

*** code*

```
if x > 100 then
        x = 0
endif

if y < 250 then
        y = 0
endif
```

*** **AGAIN**: a repeated statement made in class and here in this document. It does not mean that because the statements are within one program and deal with one story that we are required to combine the if..then statements. In the above example, the dealing with 'x' have nothing to do with the dealing of 'y'.*

2. Positive Logic - allows the flow of processing to continue through when the result of condition is false. If the result of the condition is true, then there are instructions to execute, and we leave the nested block of if..then statements.

3. Negative Logic - allows the flow of processing to continue through when the result of the condition is true. The opposite of the above point, if the result of the condition is false, then there are instructions to execute and once again, leave the block of nested if..then statements.

Point #2 & #3 above provides us with an interesting opportunity, that not all the instructions are processed or need to be processed in a given scenario. Once an instruction is executed, then the computer will move on to the next instruction. It should also be noted that this now means that we are going to be developing **nested if..then statements** (one if..then inside of another)

Examples that showcase the ideas of positive and negative logic will be presented in the below section after the notes from the slides/textbook about "*Making selections within ranges*".

----

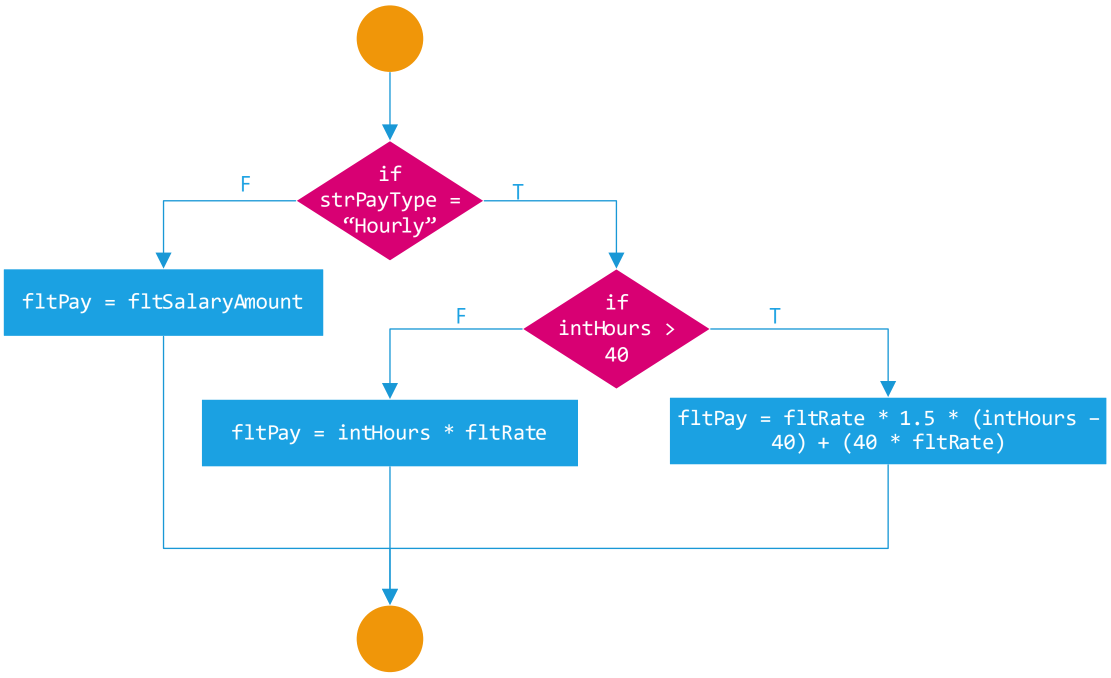
4. Aside from points #2 & #3 presenting us with an opportunity to deal with nested if..then statements, it does not mean that when we do develop nested if..then statements that we are always developing either positive or negative logic. There are times when we are required to create a nested if..then statement, because the solution for a particular piece of the program depends on the efficiency of the code.

Consider the following example, you are required to calculate and output the pay for an employee. If the pay type of the employee is “Hourly”, then you are required to calculate the wages for the number of hours worked. If the “Hourly” employee has worked for more than 40 hours in a week, then you are required to pay them at 1.5 * rate of pay for the overtime hours. If the employee type is not a “Hourly”, then please assign the standard salary amount to be paid.

** *pseudocode*

```
if strPayType = "Hourly" then
        if intHours > 40 then
                fltPay = fltRate * 1.5 * (intHours - 40) + (40 * fltRate)
        else
                fltPay = intHours * fltRate
        endif
else
        fltPay = fltSalaryAmount
endif
```

** *flowchart*



if strPayType = “Hourly”
F
T
fltPay = fltSalaryAmount
if intHours > 40
F
T
fltPay = intHours * fltRate
fltPay = fltRate * 1.5 * (intHours – 40) + (40 * fltRate)

Making selections within ranges

===

-- Within the lesson about decision logic, there is an opportunity to learn about how to deal with conditions that must satisfy looking at a range of values

- **Example:** Sales must be between R500 & R1000 (including both amounts stated) to receive a 5% discount

-- Developing separate logical expressions might seem like the right thing to do, but it is not the most efficient & possibly incorrect

** *pseudocode - using the above example*

```
if fltSales >= 500 then
        fltDiscount = 0.05
endif

if fltSales <= 1000 then
        fltDiscount = 0.05
endif
```

- Taking note of the above code - the first if..then looks to be correct. What if the user enters 1500, then the 1st condition will result in being true and will set the discount percentage to 0.05, but there is a limit that the sales amount needs to be below 1000 as well. This 1500 is then not within the range of 500 to 1000 and therefore should have never been assigned the 5% discount value.

- The same can be said about the second if..then statement. If the user enters 400, the value being less than 1000 and discount once again is set to 0.05. But the sales amount supposed to be above 500.
- I trust you are able to see the problem.

-- We should consider combining the conditions

** *pseudocode*

```
if (fltSales >= 500) AND (fltSales <= 1000) then
        fltDiscount = 0.05
endif
```

- This condition/logical expression now caters for the range correctly - the sales amount has to be between 500 and 1000 in order to receive the discount through using a logical operator. Also take note of the notes shared earlier about the use of the AND operator, both operands are in this case, relational expressions, that will produce either a true or false result, which will then be used to evaluate the whole expression with the AND operator.

-- Look through the example presented in the textbook - Figure 4-19

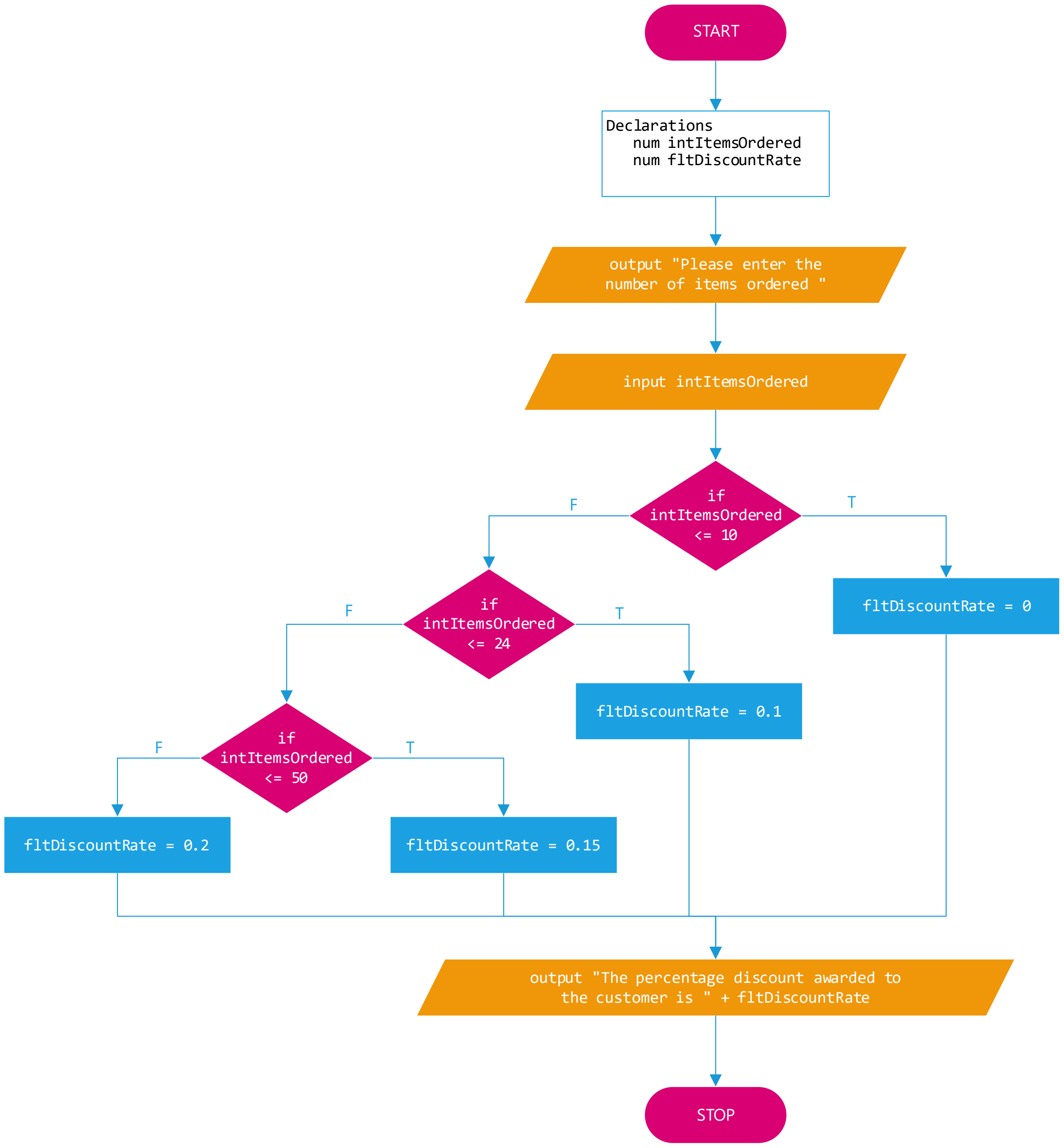
- A discount percentage is awarded based on the number of items ordered

Items ordered	**Discount Rate (%)**
0 to 10	0%
11 to 24	10%
25 to 50	15%
51 or more	20%

** *pseudocode (complete program)*

```
0. Start
1. Declarations
        num intItemsOrdered
        num fltDiscountRate

2.      output "Please enter the number of items ordered "
3.      input intItemsOrdered

4.      if intItemsOrdered <= 10 then
                fltDiscountRate = 0
        else
                if intItemsOrdered <= 24 then
                        fltDiscountRate = 0.1
                else
                        if intItemsOrdered <= 50 then
                                fltDiscountRate = 0.15
                        else
                                fltDiscountRate = 0.2
                        endif
                endif
        endif

5.      output "The percentage discount awarded to the customer is " + fltDiscountRate

6. Stop
```

** *flowchart*


START
Declarations
num intItemsOrdered
num fltDiscountRate
output "Please enter the number of items ordered "
input intItemsOrdered
if intItemsOrdered <= 10
F
T
fltDiscountRate = 0
if intItemsOrdered <= 24
F
T
fltDiscountRate = 0.1
if intItemsOrdered <= 50
F
T
fltDiscountRate = 0.2
fltDiscountRate = 0.15
output "The percentage discount awarded to the customer is " + fltDiscountRate
STOP

** Errors

-- In the following section in the textbook and the slides, when we consider developing this type of logic, it is important to recognize and avoid possible errors while coding. Some errors may result in not being a problem, but for a programmer, the code is not as efficient as it could have been.

-- Since you are dealing with a range of values, avoid checking that values that do not occur.

-- (In the future) requiring prior knowledge of the data we are dealing with, will help better develop the series of conditions.

-- Reconsider the need for a decision, if there is only one possible outcome.

-- If the result is an obvious one, please question the need for a decision again - by that we mean, do we need an if..then statement here if the result of the condition is going to always be true.

-- When dealing with the development of the conditions, especially with the range of values, it should be noted in the example above - even through a range is stated, it is not necessary to test both the limits since the preceding or following condition is likely to deal with those values as required. This would take some practice to understand ...

*** side note - our current prescribed textbook discusses this version (Making selections within ranges) of nested if..then statements but does not use the term, positive logic, and while the logic about to discussed is similar, there is a little more that needs to be understood and in-turn, the knowledge needs to flow from understanding positive logic to then understanding negative logic*

Positive Logic (Multiple nested if..then statements)
===

-- Easier to deal with this type of logic in decision making because it is close to our way of thinking (thinking in the positive).

-- Using nested if..then statements - if the result of a condition is true, instructions are processed and if the result of the condition is false, then another decision is executed. (Until you reach the end of the list conditions – the last true instruction will be executed in the previous if..then statement’s false section)

-- When we use this type of logic, then no more decisions are executed once the result of condition is true.

-- Let's look at another example:

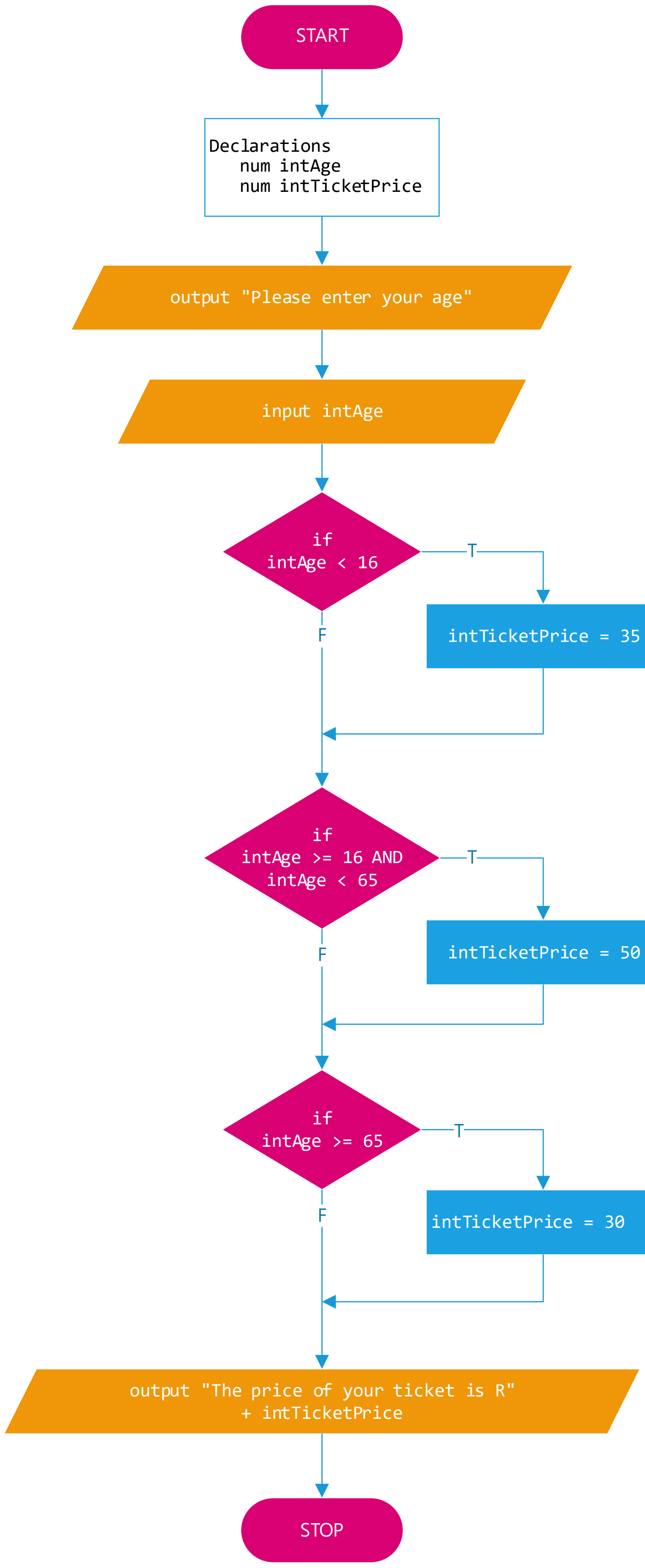
A cinema decides to charge their customers based on their age.
If they meet the following requirements, then the proceeding charge applies for their ticket.

Age	**Ticket Price**
Age < 16	R 35
Age >= 16 and Age < 65	R 50
Age >= 65	R 30

** *algorithm / pseudocode*

```
0. Start
1. Declarations
        num intAge
        num intTicketPrice

2.      output "Please enter your age"
3.      input intAge

4.      if intAge < 16 then
                intTicketPrice = 35
        endif

5.      if intAge >= 16 AND intAge < 65 then
                intTicketPrice = 50
        endif

6.      if intAge >= 65 then
                intTicketPrice = 30
        endif

7.      output "The price of your ticket is R" + intTicketPrice

8. Stop
```

```
** flowchart
```


START
Declarations
num intAge
num intTicketPrice
output "Please enter your age"
input intAge
if
intAge < 16
T
F
intTicketPrice = 35
if
intAge >= 16 AND
intAge < 65
T
F
intTicketPrice = 50
if
intAge >= 65
T
F
intTicketPrice = 30
output "The price of your ticket is R"
+ intTicketPrice
STOP

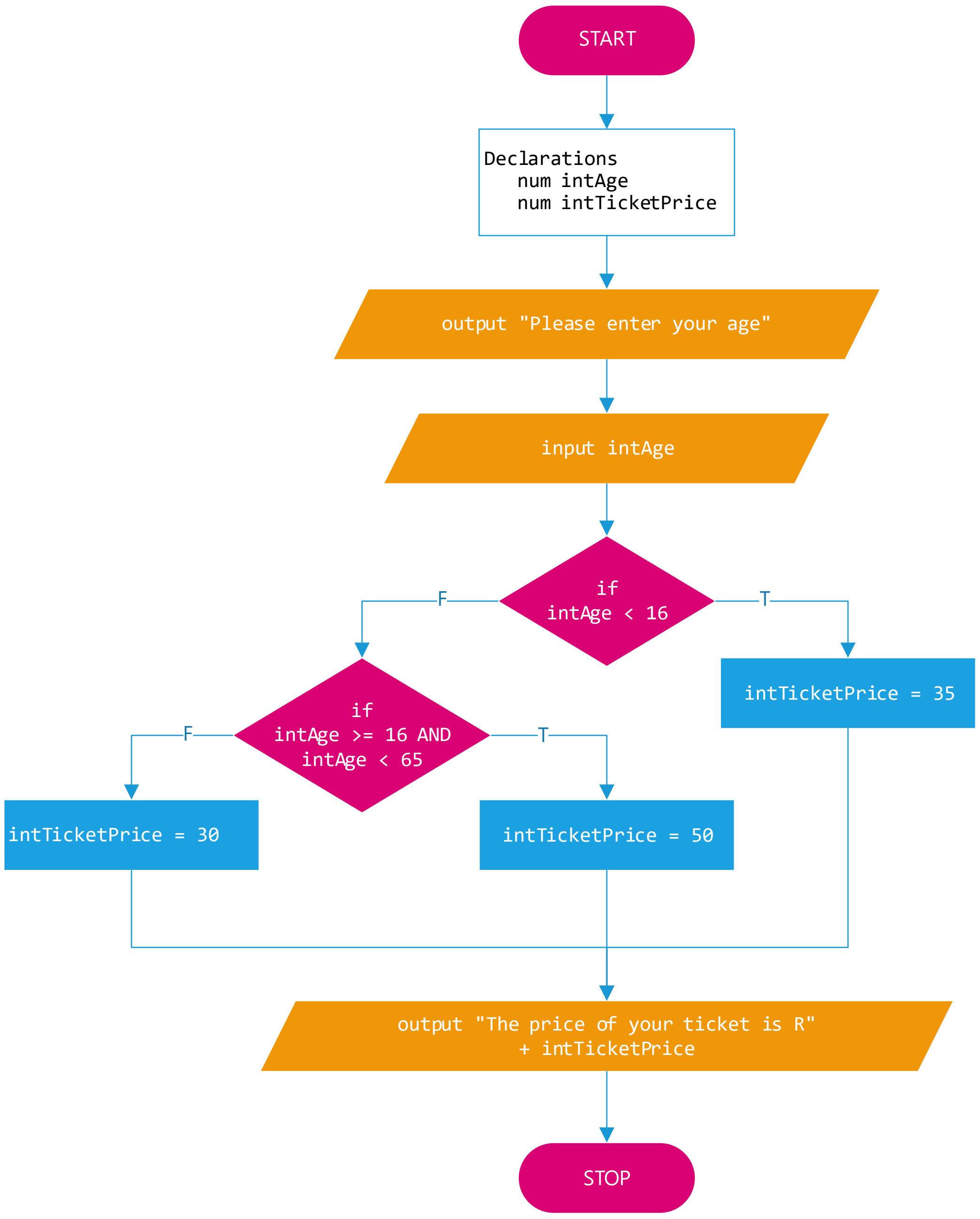
- The above example shows us that executing each of the conditions separately, in their own if..then statements is a decent solution, but the coding can made to be more efficient, restructuring it using positive logic.

- Please take note of the differences between the above code and now this new version below.

*** algorithm / pseudocode*

```
0. Start
1. Declarations
        num intAge
        num intTicketPrice

2.      output "Please enter your age"
3.      input intAge

4.      if intAge < 16 then
                intTicketPrice = 35
        else
                if intAge >= 16 AND intAge < 65 then
                        intTicketPrice = 50
                else
                        intTicketPrice = 30
                endif
        endif

5.      output "The price of your ticket is R" + intTicketPrice

6. Stop
```

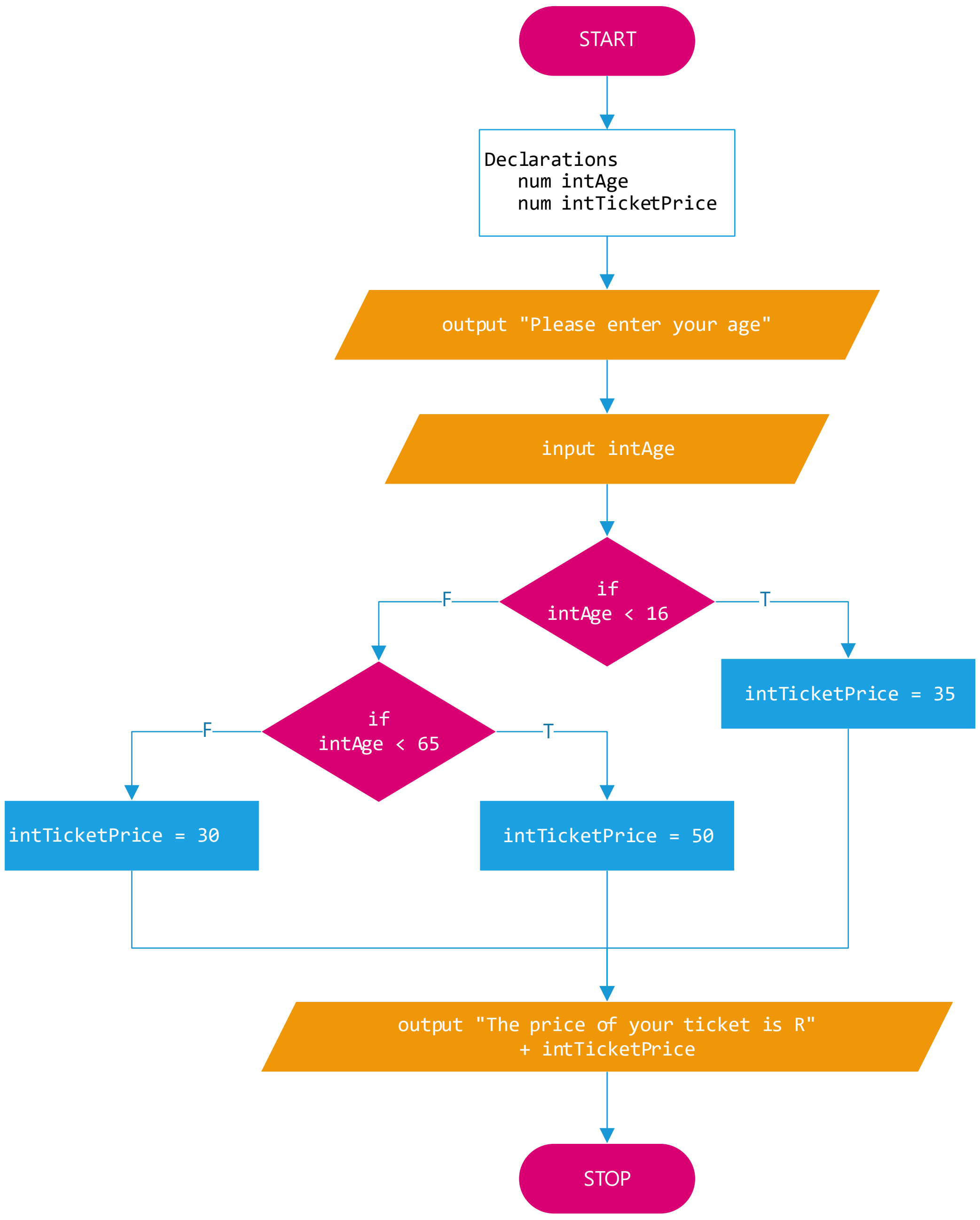
** *flowchart*



- Notice that there is one less decision being processed. There is no need to process the third decision like in the previous example, simply because the resulting false of the original 2nd condition (now nested if..then statement) is the same as if executing & getting a true on the original 3rd condition.(*a little confusing, but hopefully when you remember what we had discussed in class, this is now better understood*)

- a further edit can also be made with the now nested if.then statement, if the 1st condition presents a false result, then that means that the age is actually >= 16 and this would mean that testing that part of the condition/expression (intAge >= 16) is now again not necessary.

```
0. Start
1. Declarations
        num intAge
        num intTicketPrice

2.      output "Please enter your age"
3.      input intAge

4.      if intAge < 16 then
                intTicketPrice = 35
        else
                if intAge < 65 then
                        intTicketPrice = 50
                else
                        intTicketPrice = 30
                endif
        endif

5.      output "The price of your ticket is R" + intTicketPrice

6. Stop
```


START
Declarations
num intAge
num intTicketPrice
output "Please enter your age"
input intAge
if intAge < 16
F
T
intTicketPrice = 35
if intAge < 65
F
T
intTicketPrice = 30
intTicketPrice = 50
output "The price of your ticket is R" + intTicketPrice
STOP

-- Some examples when presented will not immediately use words or the actual relational operators when discussing the range.

Example

(Sales less than or equal to 2000 - Commission is 2%)
Sales 2001 - 4000 - the Commission is 4%

- The development of the condition now requires you the developer to decide on the correct operator

```
...
if (fltSales >= 2001) AND (fltSales <= 4000) then
        fltComm = 0.04
...
```

- Again, in similar style of the coding knowledge from the above notes, we could drop off one part of the combined condition (this being the 2nd condition), similar to the logic of the previous example program, because the previous condition would have dealt with commission for anything less or equal to the R2000

```
if fltSales <= 4000 then
        fltComm = 0.04
...
```

-- With the development of every program, we are required to input test data for every possible path to view that the correct results are being processed.

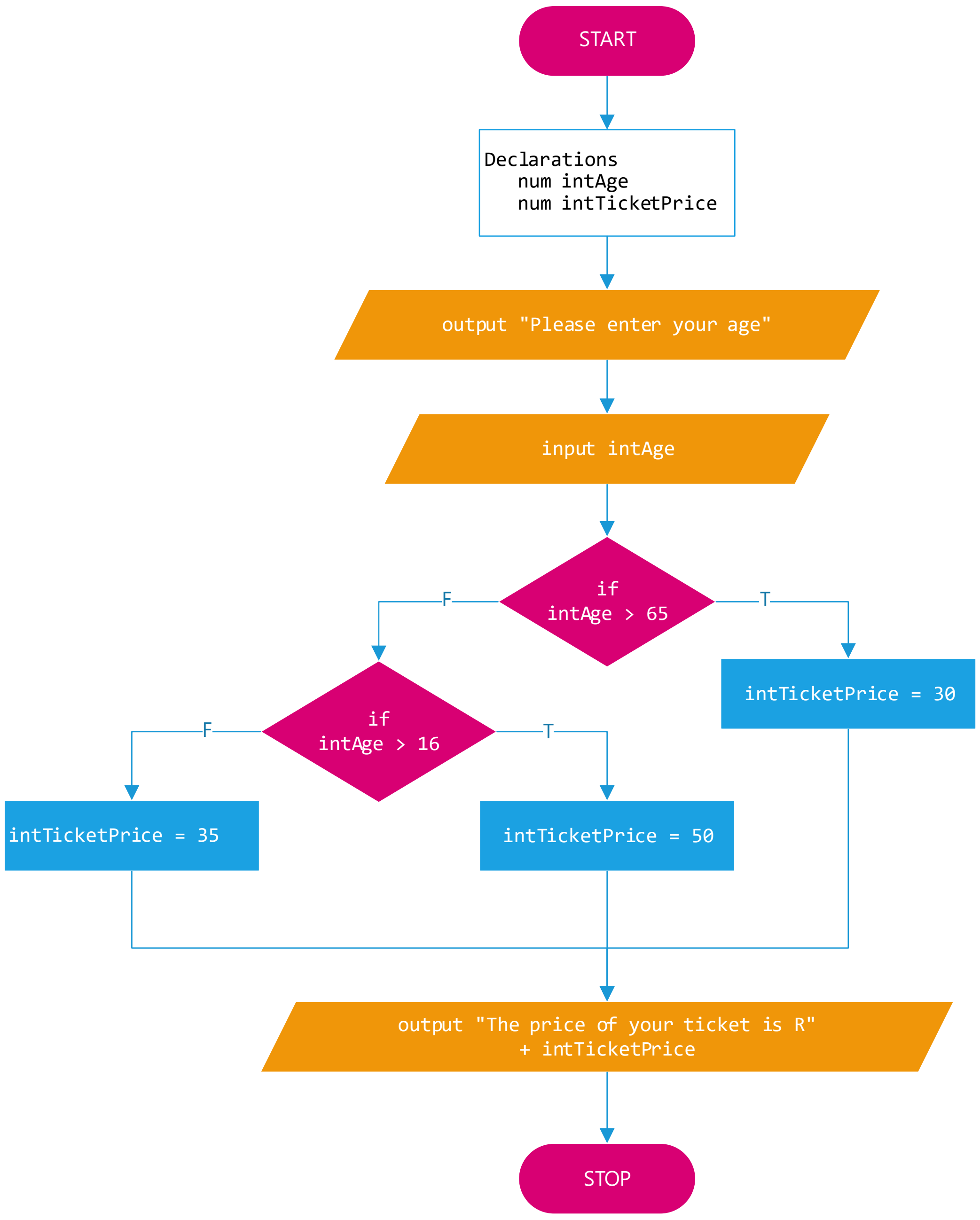
- In the above example using the age and ticket price, I would need to test the following ages: 15, 16, 20, 64, 65, 70 as an example - by testing all possibilities of input, I can then see if the right ticket price is being displayed for the customer.

-- Taking note of the order of the if..then conditions, please remember that we tend to follow an order based on how we were presented with the data, in the example, from lowest age to the highest. But we do have an opportunity to reconsider the order. Usually putting the condition that constantly results in being true will be processed first. (*This will take some practice to understand*)

The following example is using positive logic, but this time processing the conditions in reverse order

*** algorithm / pseudocode*

```
0. Start
1. Declarations
        num intAge
        num intTicketPrice

2.      output "Please enter your age"
3.      input intAge

4.      if intAge > 65 then
                intTicketPrice = 30
        else
                if intAge > 16 then
                        intTicketPrice = 50
                else
                        intTicketPrice = 35
                endif
        endif

5.      output "The price of your ticket is R" + intTicketPrice

6. Stop
```



** Again - in the above example, we are still following the rules of positive logic, but the order of the conditions is reversed, because we may have done some research and considered that this would be more efficient to code.*

Negative Logic (Multiple nested if..then statements)
===
-- Generally, this version of logic for nested if..then statements is a little difficult for us to comprehend, because we do not immediately think in negative terms.

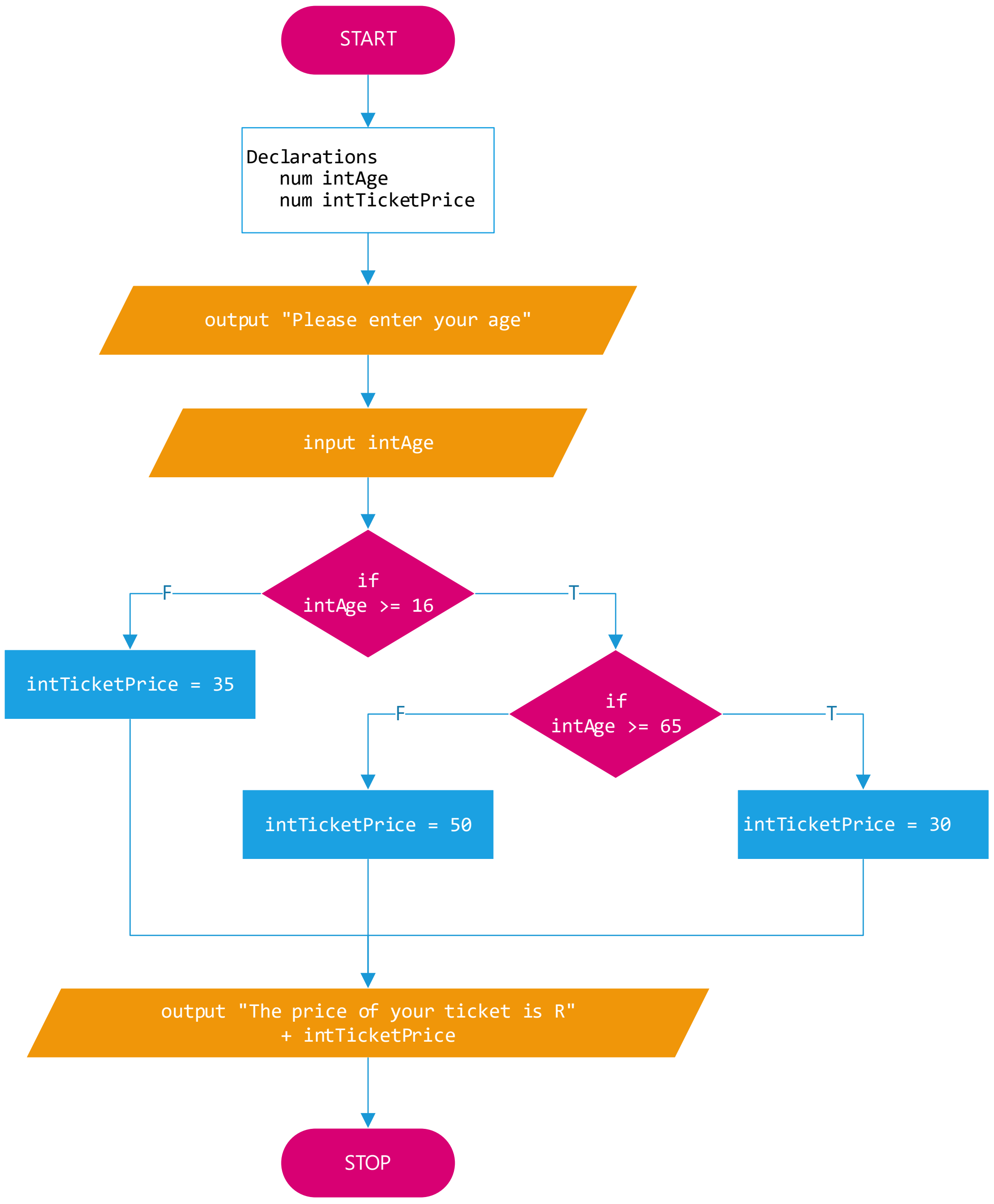
-- Here we are expected to tell the computer to continue processing if..then statements if the condition is true, but if the condition results in being false, then process an instruction. (*The opposite of positive logic*)

-- Using the same example of the age and ticket price and the original order of conditions that were presented to us, please take note of the following code:

** *pseudocode*

```
        0. Start
        1. Declarations
                num intAge
                num intTicketPrice

        2.      output "Please enter your age"
        3.      input intAge

        4.      if intAge >= 16 then
                        if intAge >= 65 then
                                intTicketPrice = 30
                        else
                                intTicketPrice = 50
                        endif
                else
                        intTicketPrice = 35
                endif

        5.      output "The price of your ticket is R" + intTicketPrice

        6. Stop
```

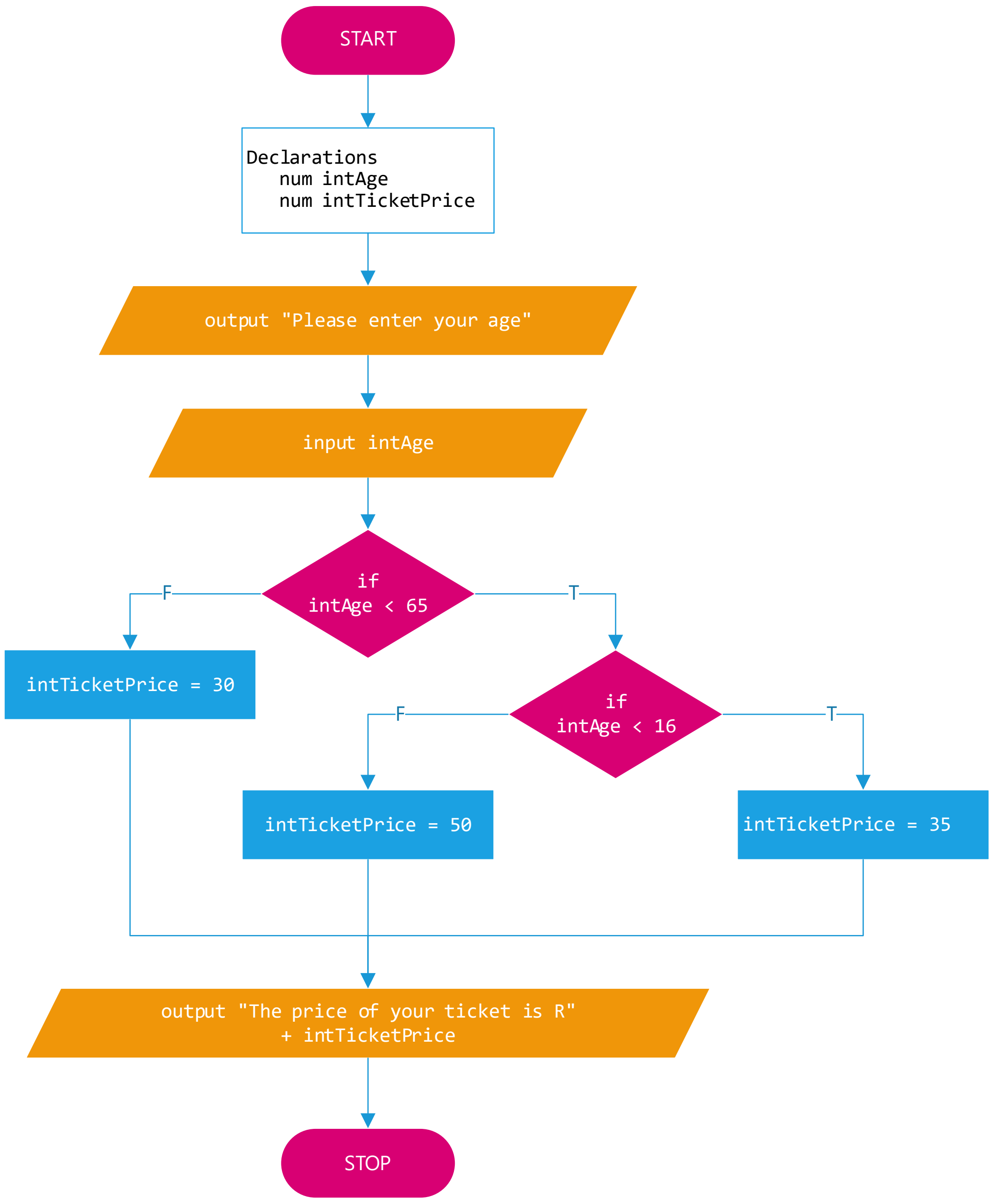
** *flowchart*



- Take note that the instructions (the ticket price) are only dealt with if the condition is false (until the last remaining instruction is executed in the true section of the last if..then statement)

*** pseudocode - in reverse order*

```
        0. Start
        1. Declarations
                num intAge
                num intTicketPrice

        2.      output "Please enter your age"
        3.      input intAge

        4.      if intAge < 65 then
                        if intAge < 16 then
                                intTicketPrice = 35
                        else
                                intTicketPrice = 50
                        endif
                else
                        intTicketPrice = 30
                endif

        5.      output "The price of your ticket is R" + intTicketPrice

        6. Stop
```



```
----------------------------------------------------------------------------------------------------
Which decision logic do we use?
====================================================================================================

-- By answering the following question, we would be better able to analyze which type of decision logic
we would employ in our solution making it the most efficient solution as possible for the problem we
are faced with.

        - Which type would make the solution most readable?
        - Which type would make the solution the easiest to maintain or change?
        - Which would require the fewest tests when you don't know anything about the data?
        - Which would require the fewest test when you are given some data?
```

Precedence when combining AND & OR operators
===
-- When we make the decision to combine multiple conditions into a compound condition, we are going to have to decide on the correct use of the AND & the OR operators and if needed, might they both be needed in the condition. If this is the case, then knowing the order of precedence of the operators once again will help in the development of the complete condition.

-- *AND comes first - then OR*

consider the following example from our textbook Figure 4-23
** *pseudocode*

```
        if chrRating = 'G' then
                if intAge <= 12 then
                        output "Discount applies"
                else
                        if intAge >= 65 then
                                output "Discount applies"
                        endif
                endif
        endif
```

- The above code is using nested if..then statements (*may seem like the positive logic idea we read and learnt above – but it is not*) - but if we take a closer look at the code and what is being coded. we could make it more efficient by using logical operators to deal with the end result that we would still expect to see.

```
        if (chrRating = 'G') AND ((intAge <= 12) OR (intAge >= 65)) then
                output "Discount applies"
        endif
```

* *While the brackets () are not immediately needed - just using them to clearly identify each of the pieces of this new compound condition*

** Errors

We need to learn how to effectively use the right logical operator at the right time.

It is also not only good enough to know the truth tables of the logical operators - knowing the combinations of true and false and the resultant values.

It needs to be said that we need to practice multiple examples and make the necessary errors to help better understand how and when a specific logical operator will aid us in developing a well-formed logical expression.

- *Textbook - Figure 4-15*

There is so much more to learn, with the development of various versions of if..then statements. This will come through the experimenting with examples that we are presented with.