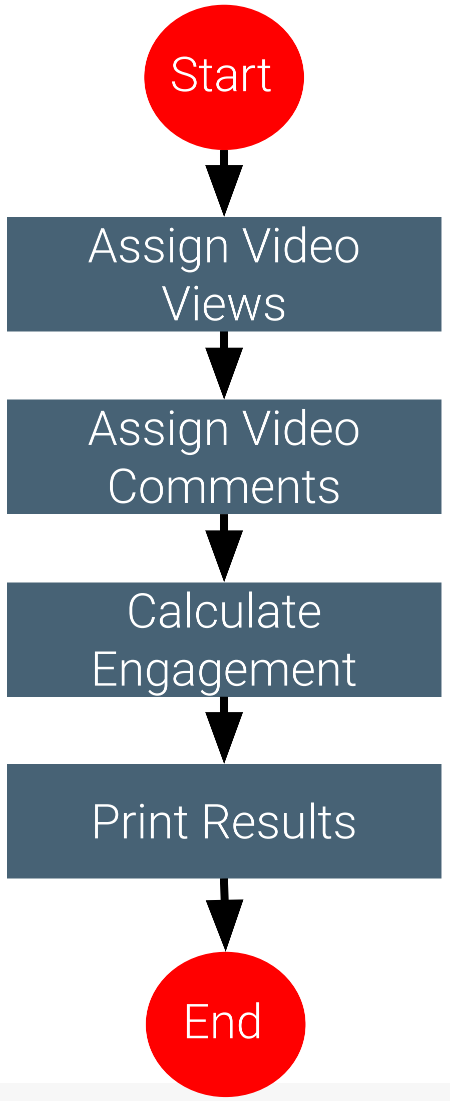
DATA SCIENCE 101:
DECISION STRUCTURES & BOOLEAN LOGIC

SEQUENCE STRUCTURES

- What we have been programming so far is known as a **sequence structure**
- Sequence structures are sets of statements that execute in the order in which they appear
- Unfortunately not all programs can be written this way, as there are times where we need to deviate from a linear structure and based on certain conditions



EXAMPLE: CALCULATING OVERTIME PAY

- If a worker works more than 30 hours in a week we have to pay overtime pay
- The overtime pay rate is 1.5 times of the worker’s hourly rate
- This additional rate is only applied to hours worked above the 30 hour limit
- Our current Python toolset doesn’t give us the ability to address this kind of problems

SOLUTION: CALCULATING OVERTIME PAY

What we need is a decision construct that:

- Allows your program to “ask a question” and respond accordingly
- Simplest form – perform an action only if a certain condition exists
- If the condition is not met, then the action is not performed

TYPES OF DECISION CONSTRUCT

- There are three general of selection statements that allows us to represent different types of decisions:
  - `if`
  - `if else`
  - `if elif else`

BOOLEAN EXPRESSIONS

- Writing a condition
- Boolean expressions
- Relational Operators
- Conditional Operators
- Boolean Operator Tips

WRITING A CONDITION

- The trick to writing a selection statement is in constructing a condition that matches the question you are trying to ask the computer
- All selection statements must have a condition to “test”
- Think of conditions as “yes or no” questions. They can only be answered by one of two options – “True” or “False”

EXPRESSING CONDITIONS USING BOOLEAN EXPRESSIONS

- In programming, conditions are expressed through the use of Boolean expressions which gives either a **`True`** or **`False`**.

```
                 an expression that gives
                     True or False

if Condition:
    Statement
    Statement
    Statement
```

TOOLS FOR FORMING BOOLEAN EXPRESSIONS

- Boolean expressions are generally formed using the following two types of operators:
    - Relational operators
    - Conditional operators

RELATIONAL OPERATORS

Operator	Example	Description
`>`	a `>` b	`True` if a is greater than b
`>=`	a `>=` b	`True` if a is greater than or equal to b
`<`	a `<` b	`True` if a is less than b
`<=`	a `<=` b	`True` if a is less than or equal to b
`==`	a `==` b	`True` if a is equals to b

- ALL Boolean expressions boil down to `True` or `False`
- Programmers often say that the expression “evaluates” to `True` or `False`

IN-CLASS EXAMPLE: EVALUATE - RELATIONAL

```
#Given these variables
a = 100
b = 5
c = -5
d = 5
```

```
#Evaluate these expressions
a > b
b < c
b >= c
c <= d
a == b + d
d <= a + c
c != b
```

IN-CLASS EXAMPLE: RELATIONAL OPERATORS

With Relational Operators

```
see_you_again = 5000
despacito = 15000

if see_you_again < despacito:
   print('See You Again has less view!')
```

CONDITIONAL OPERATORS

Operator	Example	Description
`and`	`a and b`	`True` if a and b are both true
`or`	`a or b`	`True` if either a or b are true

- Often used in conjunction with a relational operator

LET'S EVALUATE THESE MENTALLY AND TRY ON JUPYTER

```
#Given these variables
a = 100
b = 5
c = -5
d = 5
```

```
#Evaluate these expressions
a > b and b > c
a > b and b < c
a > b or b < c
a >= b + d and a > c
c == b or 2b == d - c
```

IN-CLASS EXAMPLE: RELATIONAL AND CONDITIONAL

With Relational & Conditional Operators

```
see_you_again = 5000
despacito = 15000
havana = 3000

if see_you_again < despacito and Havana < despacito:
   print('despacito has the most number of views!')
```

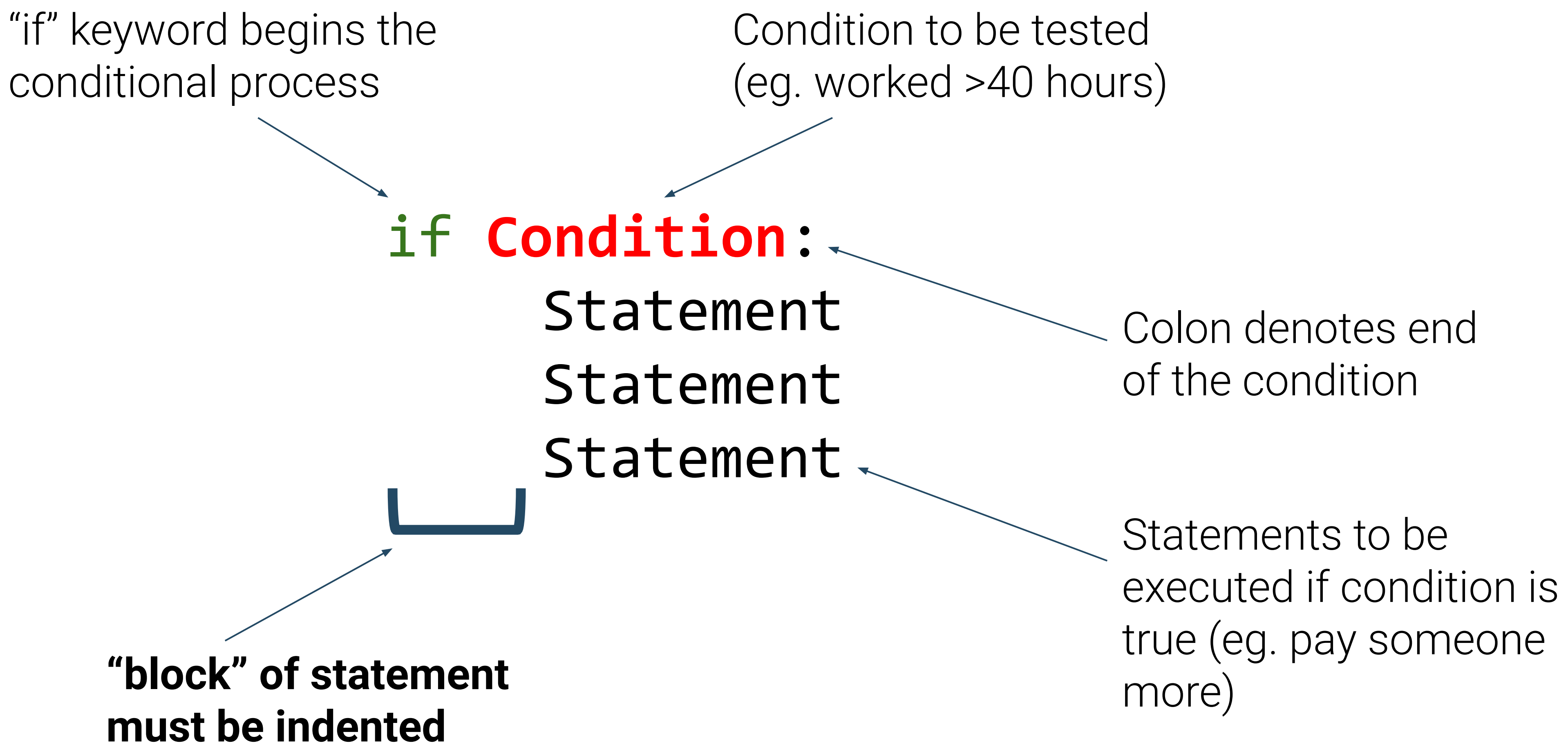
BOOLEAN OPERATOR TIPS

- Don’t confuse `==` with `=`
  - `=` is used for assigning values to variables
  - `==` is used for testing if two values/variables are identical
- Use `!=` if you want to test if two values are different
- The `<=` and `>=` operators test for more than one relationship
  - `<=` tests to see if a value is less than OR equal to another
  - `>=` tests to see if a value is greater than OR equal to another

`if` CONSTRUCT

- `if` general
- `if` specific form

GENERAL FORM OF DECISION CONSTRUCT



“if” keyword begins the conditional process

Condition to be tested (eg. worked >40 hours)

```
if Condition:
    Statement
    Statement
    Statement
```

Colon denotes end of the condition

Statements to be executed if condition is true (eg. pay someone more)

“block” of statement must be indented

SPECIFIC FORM OF “IF” DECISION CONSTRUCT

```
engagement_score = 0.8
if engagement_score >= 0.5:
   print('Good engagement!')
```

A video is considered ‘Good engagement’ if the score is equal or above 0.5

SPECIFIC FORM OF “IF” DECISION CONSTRUCT

- Basically, whether the action enclosed within the if statement is executed, is conditional upon whether the expression returns a **`True`** or **`False`**

```
if True:
    print('this value is true')


if False:
    print('this value is false')
```

IN-CLASS PRACTICE: IF STATEMENT EXAMPLES*

- Try the following to consolidate your understanding of the "IF" decision construct

Example 1

```
value = 0
if value <= 0:
    print(value)
```

Example 2

```
value = 100
if value == 0 or value >
100:
    print(value)
```

Example 3

```
value = 0
if value != 0:
    print(value)
```

Example 4

```
lessonIsFun = True
if lessonIsFun :
    print('I am getting the hang of this')
```

Example 5

```
if not lessonIsFun :
    print("oh no...")
```

IN-CLASS PRACTICE: USING CONDITIONAL OPERATORS*

```
#The following are details of a youtube video, make use of
conditions and if structure to check the video is from
Singapore, and if the views are more than 150000. If it’s
true, print out that “This is Singapore’s most popular
video.”

country = “Singapore”
views = ‘6732948’
```

IF-ELSE CONSTRUCT

- Simple decision construct
- General form of IF-ELSE construct
- Specific form of IF-ELSE construct

SIMPLE DECISION CONSTRUCT

- The selection statements we have been writing so far have only allowed us to create a single alternate branch of execution
- There are many times when we need to create multiple branches of execution based on the value of a Boolean expression

GENERAL FORM OF IF-ELSE CONSTRUCT

- The IF-ELSE structure allows you to perform one set of statements if a condition is true, and another if it is false

```
if Condition:
    Statement
    Statement
    Statement
else:
    Statement
```

Statements to be executed if condition is true (eg. pay someone more)

Statements to be executed if first condition is not true

SPECIFIC FORM OF “IF-ELSE DECISION CONSTRUCT

```
engagement_score = 0.8
if engagement_score >= 0.5:
   print('Good engagement')
else:
   print('Need improvement')
```

‘Good engagement’ will only be printed if score is equal or above 0.5

‘Need improvement’ will only be printed if score below 0.5

IN-CLASS PRACTICE: IF-ELSE + CONDITIONAL OPERATORS*

```
#The following are details of a youtube video, make use of
conditions and if structure to check the video is from
Singapore, and if the views are more than 150000. If it’s
true, print out that “This is Singapore’s most popular
video.” Else, print out that “this is another normal video

country = “Singapore”
views = ‘2390’
```

IF-ELIF-ELSE CONSTRUCT

- General form of IF-ELIF-ELSE construct
- Specific form of IF-ELIF-ELSE construct

GENERAL FORM OF IF-ELIF-ELSE CONSTRUCT

- The IF-ELSE structure allows you to perform one set of statements if a condition is true, and another if it is false

```
if Condition_1:
    Statement
    Statement
    Statement
elif Condition_2:
    Statement
```

Statements to be executed if condition is true (eg. pay someone more)

Statements to be executed if first condition is not true and second condition is true

SPECIFIC FORM OF IF-ELIF-ELSE DECISION CONSTRUCT

```
engagement_score = 0.8
if engagement_score < 0.5
    print('Need improvement')
elif engagement_score <= 0.7:
    print('Good!')
else:
    print('Excellent!')
```

'Need improvement' will only be printed if score is below 0.5

'Good!' will only be printed if score between 0.5 and 0.7

'Excellent!' will only be printed if score above 0.7

IF-ELSE COMMON LOGIC ERROR

- One of the most common form of If-Else logic error takes the below form. Can you tell what is the error?

```
engagement_score = 0.8
if engagement_score >= 0.5
    print('Good')
elif engagement_score >= 0.7:
    print('Excellent!')
else:
    print('Need improvement')
```

IN-CLASS PRACTICE: CALCULATING INFLUENCER PAY*

- The YouTube influencer management team wants to automate the payment to the influencers based on monthly views on their channel. The influencer monthly payment rules as follow:
    - The influencer must have at least 50,000 views in the month to qualify for a payment.
    - The influencer will be paid $0.01 per view if he/she has between 50,000 to 100,000 views.
    - If the influencer has more than 100,000 he/she will be paid $0.05 per views
- Write a Python program to calculate the pay for the below influencers:
    - Xia Xue has 126,311 views
    - Roy has 819 views
    - Browie Tv has 51,101 views

NESTED DECISIONS & OTHERS

- Nested Decisions
- Try-Except
- “Not” operator

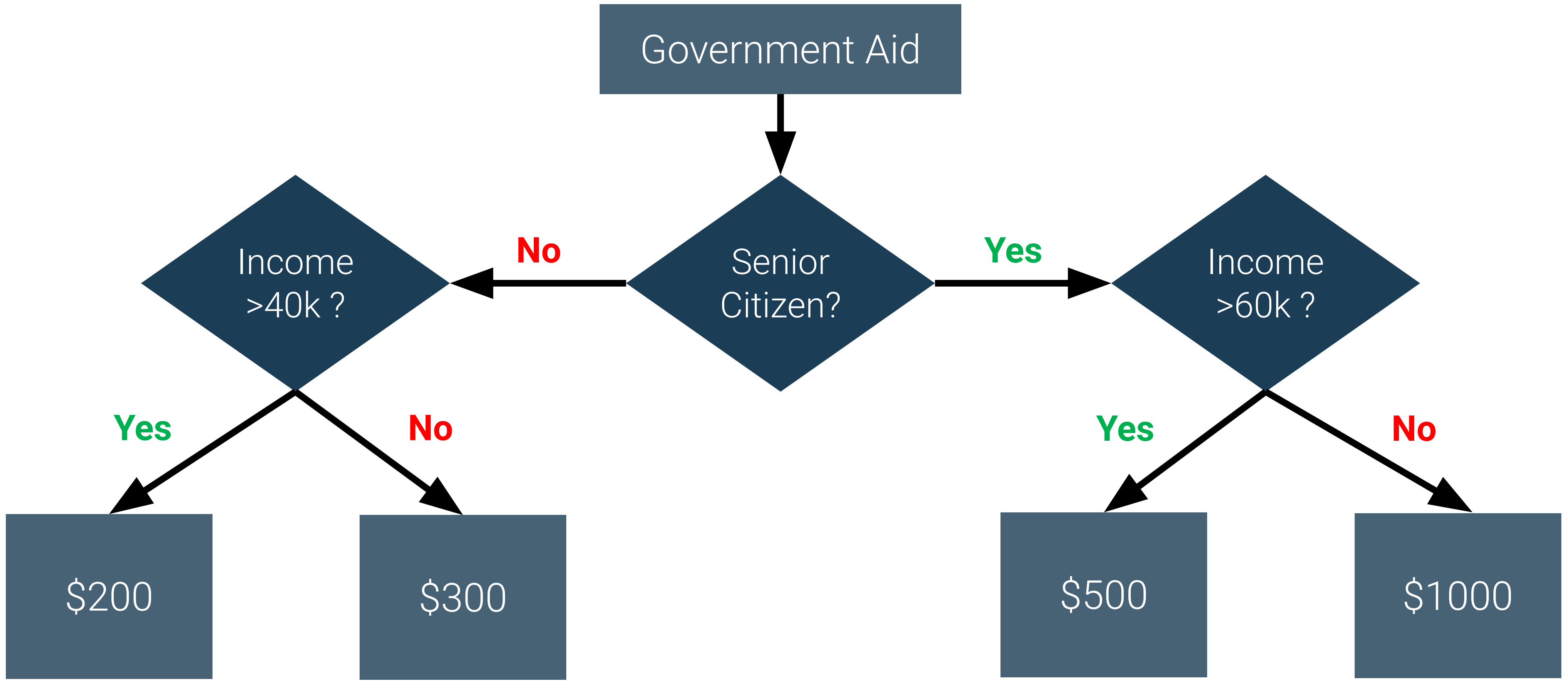
NESTED DECISIONS: WHAT AND WHY?

- There are instances where you need to ask “follow-up” questions after clearing the first condition

- Python allows you to evaluate “follow-up” conditions through the nesting of one decision construct in another

NESTED DECISIONS: WHAT AND WHY?

- Consider the following structure which a government adopts to dish out its budgetary aids:



NESTED DECISIONS: HOW?

- The decision tree can be represented by the following nested structure:

```python
senior_citizen = True
income = 52000
if senior_citizen == False:
    if income > 40000:
        print('Give $200')
    else:
        print('Give $300')
else:
    if income > 60000:
        print('Give $500')
    else:
        print('Give $1000')
```

NESTED DECISIONS: HOW?

- Notice that the nesting is actualized with **indentation** – Python will use **indentation** level of a structure to determine whether it's a child or standalone statement

```python
senior_citizen = True
income = 52000
if senior_citizen == False:
    if income > 40000:
        print('Give $200')
    else:
        print('Give $300')
else:
    if income > 60000:
        print('Give $500')
    else:
        print('Give $1000')
```

NESTED DECISIONS: BONUS QUESTION*

- Can you rewrite the below code without nested decision?

```
senior_citizen = True
income = 52000
if senior_citizen == False:
    if income > 40000:
        print('Give $200')
    else:
        print('Give $300')
else:
    if income > 60000:
        print('Give $500')
    else:
        print('Give $1000')
```

NESTED DECISIONS: BONUS QUESTION*

- Can you rewrite the below code without nested decision?

```
senior_citizen = True
income = 52000
if senior_citizen == False:
    if income > 40000:
        print('Give $200')
    else:
        print('Give $300')
else:
    if income > 60000:
        print('Give $500')
    else:
        print('Give $1000')
```

```
senior_citizen = True
income = 52000
if senior_citizen == False and income > 40000:
    print('Give $200')
elif senior_citizen == False and income <= 40000:
    print('Give $300')
elif senior_citizen == False and income > 60000:
    print('Give $500')
else:
    print('Give $1000')
```

IN-CLASS PRACTICE: YOUTUBE INFLUENCER BONUS

- As part of a campaign to support healthy living, the YouTube influencer management team wants to give a special bonus to influencer who posted videos which support healthy living. The bonus are as follow:
  - Channels with 50,000 views or less get $0.01/view
  - Channels with more than 50,000 views get $0.01/view in the first 50,000 views and $0.10/view for subsequent views.
- Roy is a YouTube influencer and has the following stats:
  - has_healthy_living_video = True
  - channel_views = 85000
- Use a nested decision to calculate how much bonus Roy should get

“NOT” OPERATOR: WHAT IS IT? (SELF-STUDY)

- The “not” operator is an operator which reverses the logical value of the condition
- Basically it will change a “True” to a “False” value and vice versa

“NOT” OPERATOR: EXAMPLE (SELF-STUDY)

- The following example illustrates how a “not” operator can be used

```
senior_citizen = False
if not (senior_citizen == True):
   print('You are not a senior citizen')
else:
   print('Paying our respects to the senior citizen!')
```

TRY-EXCEPT CONSTRUCT (SELF-STUDY)

- This is a useful construct that allows you to prevent your program from crashing
- To leverage on this, you would surround a dangerous section of code with **`try`** and **`except`**
- If the code in try works, the except part would not be executed
- If the code in the try fails, it would not continue executing the code in the try portion and just jump right to the except portion

IN-CLASS EXERCISE: TRY EXCEPT (SELF-STUDY)

- One of the common thing we do in data analytics/science is to convert data types during “data cleaning”. The following codes will give an error, write a try-except code to catch this error:

```
views = 'One hundred and ten'
views = int(views)
```

SUMMARY

- Sequence structure
- Boolean operators (conditional & relational)
- IF, ELIF, ELSE
- Nested decisions
- “Not” operator
- Try-Except