University of Washington

IT Foundations 130

Module07-Functions

In this module you will learn about how to uses SQL Functions to retrieve information from a database.

Aggregate Functions (Review)

*"Aggregate functions **perform a calculation on a set of values and return a single value**. Except for COUNT, aggregate functions ignore null values. Aggregate functions are frequently used with the GROUP BY clause of the SELECT statement."* (https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql, 2017)

These functions are some of the **most useful** ones. They include Max, Min, Avg, Sum, and Count.

This example shows who placed orders on the **most recent recorded** day.

```
USE Northwind
SELECT OrderID, CustomerID
 FROM Orders
  WHERE OrderDate = (SELECT MAX(OrderDate) FROM Orders);
```

Results | Messages

	OrderID	CustomerID	OrderDate
1	11077	RATTC	1998-05-06 00:00:00.000
2	11076	BONAP	1998-05-06 00:00:00.000
3	11075	RICSU	1998-05-06 00:00:00.000
4	11074	SIMOB	1998-05-06 00:00:00.000

Figure: Result of previous SQL query

Most aggregate functions **exclude Null** values.

```
SELECT ShippedDate FROM dbo.Orders;
SELECT MAX(ShippedDate) FROM dbo.Orders;
SELECT MIN(ShippedDate) FROM dbo.Orders;
```

Results Messages

	ShippedDate
20	NULL
21	NULL
22	1996-07-10 00:00:00.000
23	1996-07-11 00:00:00.000

	(No column name)
1	1998-05-06 00:00:00.000

	(No column name)
1	1996-07-10 00:00:00.000

Figure: Result of previous SQL query

Count All (*) is the exception.

```
SELECT Count(*) as [All Orders] FROM dbo.Orders; -- INCLUDES nulls
SELECT Count(ShippedDate) as [ShippedOrders] FROM dbo.Orders; -- does NOT include nulls
```

Results Messages

	All Orders
1	830

	ShippedOrders
1	809

Figure: Result of previous SQL query

Determining the **average of the UnitPrice for all products** in the Products table can be done with the AVG function.

```
SELECT AVG(Price) as [avg price] FROM Pubs.dbo.Titles;
```

Results Messages

	avg price
1	14.7662

Figure: Result of previous SQL query

You can also use **multiple functions** in one Select statement.

```
SELECT
 [grand total] = SUM(ytd_sales),
 [average sales] = AVG(ytd_sales),
 [number of sales] = COUNT(ytd_sales),
 [number of entries] = COUNT(*)
 FROM Pubs.dbo.Titles;
```

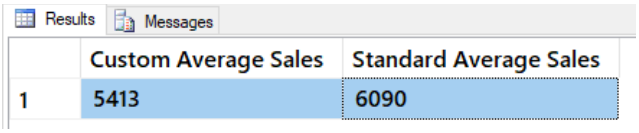
Results Messages

	grand total	average sales	number of sales	number of entries
1	97446	6090	16	18

Figure: Result of previous SQL query

You can create your own calculations as well by **combining them,** like this:

```
SELECT
 [Custom Average Sales] = SUM(ytd_sales) / COUNT(*),
 [Standard Average Sales] = AVG(ytd_sales)
 FROM pubs.dbo.titles;
```



	Custom Average Sales	Standard Average Sales
1	5413	6090

Figure: Result of previous SQL query

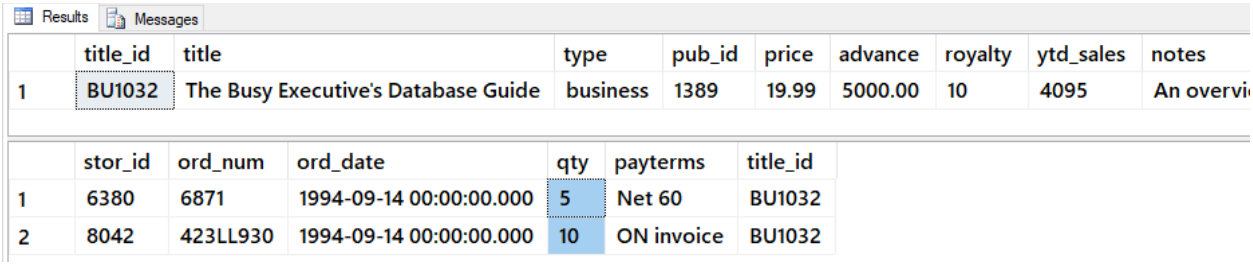
Grouping for Sub-Totals

*"The GROUP BY statement is **often used with aggregate functions** (COUNT, MAX, MIN, SUM, AVG) to **group the result-set by one or more columns**."* (https://www.w3schools.com/sql/sql_groupby.asp, 2017)

To understand group by, let's first look at some results without it:

```
SELECT * FROM Pubs.dbo.Titles WHERE title_id = 'BU1032';

SELECT * FROM Pubs.dbo.Sales WHERE title_id = 'BU1032';
```



	title_id	title	type	pub_id	price	advance	royalty	ytd_sales	notes
1	BU1032	The Busy Executive's Database Guide	business	1389	19.99	5000.00	10	4095	An overvi

	stor_id	ord_num	ord_date	qty	payterms	title_id
1	6380	6871	1994-09-14 00:00:00.000	5	Net 60	BU1032
2	8042	423LL930	1994-09-14 00:00:00.000	10	ON invoice	BU1032

Figure: Result of previous SQL queries

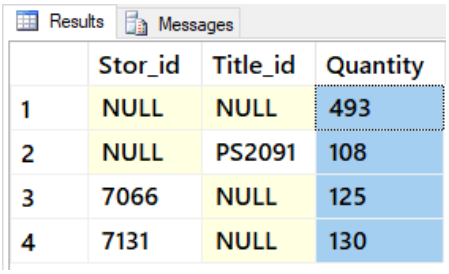
Now, let's add the Group By clause and note how it returns totals.

```
SELECT Title_id, SUM(qty) AS 'Quantity'
  FROM Pubs.dbo.Sales
  WHERE title_id = 'BU1032'
  GROUP BY Title_id;
```

Having

After data has been grouped, you can add a filter on the results using the Having option. This placement in the Select statement is different from Where clause it is applied after the grouped totals are created.

```
SELECT Stor_id, Title_id, SUM(qty) AS 'Quantity'
  FROM Pubs.dbo.Sales
  GROUP BY Stor_id,Title_id
    WITH Cube
  HAVING sum(Qty) > 100
  ORDER BY Stor_id, Title_id -- Order by is always last
```



	Stor_id	Title_id	Quantity
1	NULL	NULL	493
2	NULL	PS2091	108
3	7066	NULL	125
4	7131	NULL	130

Figure: Result of previous SQL query

Selecting with Common Functions

Functions are a named collection of SQL programming code. All RDMS include built-in functions, and some even let you create your own. We will see how to make your own custom MS SQL functions in a later module, but for now, let's look at some built-in MS SQL functions as examples.

Most SQL functions **return a single value**.

```
Select GetDate(), IsNull(null,0);
```

Results | Messages

	(No column name)	(No column name)
1	2017-09-29 12:01:47.853	0

Figure: Result of previous SQL query

But, you can use them in a Select-From statement to **apply the function to many rows**.

```
Select GetDate(), IsNull(Price, 0), Title
 From Pubs.dbo.Titles;
```

Results | Messages

	(No column name)	(No column name)	Title
1	2017-10-31 11:06:54.707	19.99	The Busy Executive's Database Guide
2	2017-10-31 11:06:54.707	11.95	Cooking with Computers: Surreptitious Balance Sheets
3	2017-10-31 11:06:54.707	2.99	You Can Combat Computer Stress!
4	2017-10-31 11:06:54.707	19.99	Straight Talk About Computers

Figure: Result of previous SQL query

You can **combine functions** to create better looking results:

```
Select
  Cast(GetDate() as Date) as [TodaysDate]
, IsNull(Price, 0) as [CurrentPrice]
, IsNull(Cast(Price as varchar(50)), 'Not For Sale!') as [CurrentPriceAsTEXT]
, Title
 From Pubs.dbo.Titles
Go
```

Results | Messages

	TodaysDate	CurrentPrice	CurrentPriceAsTEXT	Title
1	2017-10-31	19.99	19.99	The Busy Executive's Databa
2	2017-10-31	11.95	11.95	Cooking with Computers: Sur
3	2017-10-31	2.99	2.99	You Can Combat Computer S
4	2017-10-31	19.99	19.99	Straight Talk About Computer
5	2017-10-31	19.99	19.99	Silicon Valley Gastronomic Tr
6	2017-10-31	2.99	2.99	The Gourmet Microwave
7	2017-10-31	0.00	Not For Sale!	The Psychology of Computer
8	2017-10-31	22.95	22.95	But Is It User Friendly?

Figure: Result of previous SQL query

Cast and Convert

Cast and Convert are both conversion functions.

```
Select Cast('1' as int), Cast('1' as decimal(3,2)), Cast(1 as nVarchar(50));
Select Convert(int,'1'), Convert(decimal(3,2),'1'), Convert(nVarchar(50), 1);
```

Results Messages

	(No column name)	(No column name)	(No column name)
1	1	1.00	1

	(No column name)	(No column name)	(No column name)
1	1	1.00	1

Figure: Result of previous SQL query

Convert has more features than Cast.

```
Select
 [Simple Cast] = Cast(GetDate() as Date)
,[Simple Convert] = Convert(Date, GetDate())
,[US with Slash] = Convert(varchar(50), GetDate(), 101)
,[US with Dash] = Convert(varchar(50), GetDate(), 110)
,[ANSI YearMonthDay] = Convert(varchar(50), GetDate(), 112)
;
Go
```

	Simple Cast	Simple Convert	US with Slash	US with Dash	ANSI YearMonthDay
1	2017-09-29	2017-09-29	09/29/2017	09-29-2017	20170929

Figure: Result of previous SQL query

Logical Functions allow you to look for a condition the **evaluates to true or false and then return an appropriate value**.

The String Function:

```
SELECT STR(3.147);
SELECT STR(3.147, 5, 2);
SELECT STR(3.147, 3, 3);
SELECT STR(123.456, 3, 0);
Go
```

Results Messages

	(No column name)
1	3

	(No column name)
1	3.15

	(No column name)
1	3.1

	(No column name)
1	123

Figure: Result of previous SQL query

Str() is **similar** to **CONVERT**(char(15), 123.456), but if you do not allow enough room Convert() will throw an error, while str() will not.

SELECT CONVERT(char(3), 123.456)

```
Msg 8115, Level 16, State 5, Line 18 Arithmetic overflow error converting numeric to data type varchar.
```

CONCAT

```
-- https://docs.microsoft.com/en-us/sql/t-sql/functions/concat-transact-sql
Select CONCAT(1, 'a', Cast('1/1/2020' as date), 5.6); -- Different data types
go
Select CONCAT(rtrim(ltrim(str((3 * 100 / 9),5,2) )), '%') union
Select CONCAT(rtrim(ltrim(str((3 * 100 / 3.21),5,2) )), '%');
go
```

Results Messages

	(No column name)
1	1a2020-01-015.6

	(No column name)
1	33.00%
2	93.46%

Figure: Result of previous SQL query

The Format Function

```
Select Format(GetDate(), 'd', 'en-US' ) AS 'US Result'
      ,Format(GetDate(), 'd', 'en-gb' ) AS 'Great Britain Result'
      ,Format(GetDate(), 'd', 'de-de' ) AS 'Germany Result'
      ,Format(123.456, 'C', 'en-US') AS 'US Format'
      ,Format(123.456, 'C', 'en-gb') AS 'Great Britain Format'
      ,Format(123.456, 'C', 'de-de') AS 'Germany Format'
;
go
```

Results Messages

	US Result	Great Britain Result	Germany Result	US Format	Great Britain Format	Germany Format
1	10/31/2017	31/10/2017	31.10.2017	$123.46	£123.46	123,46 €

Figure: Result of previous SQL query

The Immediate IF function

```
Select IIF(5 = 5, 'T', 'F');
Select
  [ProductName] = IIF(ProductID = 3,  ProductName + ' (Not For Sale!)', ProductName)
 From Northwind.dbo.Products;
go
```

Results Messages

	(No column name)
1	T

	ProductName
1	Alice Mutton
2	Aniseed Syrup (Not For Sale!)
3	Boston Crab Meat

Figure: Result of previous SQL query

The Case Function (directive)

```
Select Case (5 + 5)
       When 10 Then 'Ten'
       When 9 Then 'Nine'
       End;
Select
 ProductName
,[Category] = Case CategoryID
               When 1 Then 'A'
               When 2 Then 'B'
               When 3 Then 'C'
              End
 From Northwind.dbo.Products;
go
```

Results | Messages

	(No column name)
1	Ten

	ProductName	Category
1	Chai	A
2	Chang	A
3	Aniseed Syrup	B
4	Chef Anton's Cajun Seasoning	B
5	Chef Anton's Gumbo Mix	B
6	Grandma's Boysenberry Spread	B
7	Uncle Bob's Organic Dried Pears	NULL
8	Northwoods Cranberry Sauce	B
9	Mishi Kobe Niku	NULL

Figure: Result of previous SQL query

A better use of Case would be when you need to do a comparison between two columns and generate a third one. For example, say we have the following shipping data and want to see if orders shipped before their due date.

Results | Messages

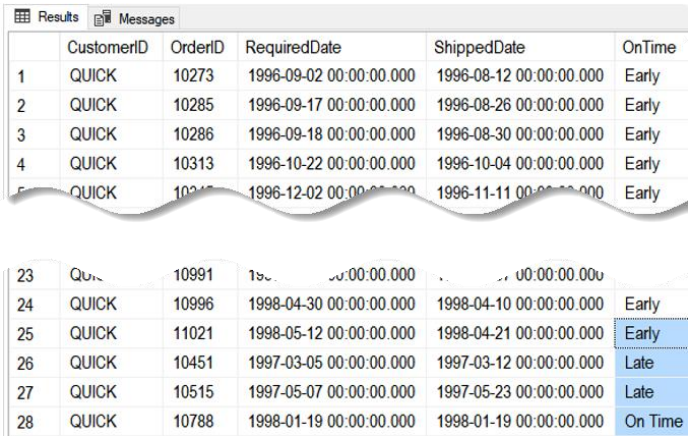
	CustomerID	OrderID	RequiredDate	ShippedDate	Late or Not Late?
1	QUICK	10273	1996-09-02 00:00:00.000	1996-08-12 00:00:00.000	
2	QUICK	10285	1996-09-17 00:00:00.000	1996-08-26 00:00:00.000	
3	QUICK	10286	1996-09-18 00:00:00.000	1996-08-30 00:00:00.000	
4	QUICK	10313	1996-10-22 00:00:00.000	1996-10-04 00:00:00.000	
5	QUICK	10345	1996-12-02 00:00:00.000	1996-11-11 00:00:00.000	
6	QUICK	10361	1996-12-20 00:00:00.000	1996-12-03 00:00:00.000	

Figure: Result of previous SQL query

Using a "Select – Case" statement will make this easy!

```
Select
  CustomerID
 ,OrderID
 ,RequiredDate
 ,ShippedDate
 ,[OnTime] = Case
              When RequiredDate > ShippedDate Then 'Early'
              When RequiredDate = ShippedDate Then 'On Time'
              When RequiredDate < ShippedDate Then 'Late'
              Else 'No Info Yet'
```

```
            End
 From Northwind.dbo.Orders
 Where CustomerID = 'QUICK'
 Order by [OnTime];
Go
```

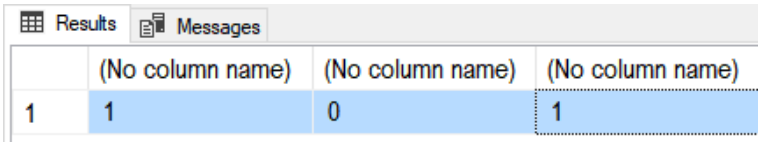


	CustomerID	OrderID	RequiredDate	ShippedDate	OnTime
1	QUICK	10273	1996-09-02 00:00:00.000	1996-08-12 00:00:00.000	Early
2	QUICK	10285	1996-09-17 00:00:00.000	1996-08-26 00:00:00.000	Early
3	QUICK	10286	1996-09-18 00:00:00.000	1996-08-30 00:00:00.000	Early
4	QUICK	10313	1996-10-22 00:00:00.000	1996-10-04 00:00:00.000	Early
	QUICK		1996-12-02 00:00:00.000	1996-11-11 00:00:00.000	Early
23	QUICK	10991	1998-03-30 00:00:00.000	1998-03-10 00:00:00.000	
24	QUICK	10996	1998-04-30 00:00:00.000	1998-04-10 00:00:00.000	Early
25	QUICK	11021	1998-05-12 00:00:00.000	1998-04-21 00:00:00.000	Early
26	QUICK	10451	1997-03-05 00:00:00.000	1997-03-12 00:00:00.000	Late
27	QUICK	10515	1997-05-07 00:00:00.000	1997-05-23 00:00:00.000	Late
28	QUICK	10788	1998-01-19 00:00:00.000	1998-01-19 00:00:00.000	On Time

Figure: Result of previous SQL query

The IsNumeric Funtion:

```
Select IsNumeric('1'), IsNumeric('a1'), IsNumeric('1.23');
```

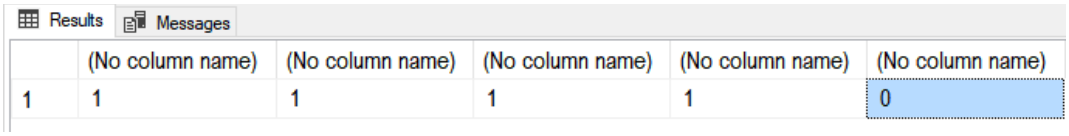


	(No column name)	(No column name)	(No column name)
1	1	0	1

Figure: Result of previous SQL query

The IsDate Function:

```
Select
 IsDate('1/1/2001')
,IsDate('01-01-2001')
,IsDate('20010101')
,IsDate('Jan,01,2001')
,IsDate('1st of Jan,2001');
```

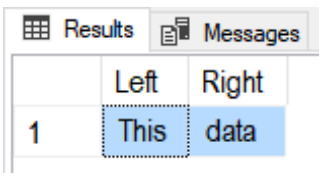


	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)
1	1	1	1	1	0

Figure: Result of previous SQL query

The Left and Right Functions

```
DECLARE @string varchar(100) = 'This is some data'
SELECT [Left] = Left(@string,4),[Right] = Right(@string,4)
;
Go
```



	Left	Right
1	This	data

Figure: Result of previous SQL query

The LTrim and RTrim Functions

```
DECLARE @string_to_trim varchar(100) = '    This is some data    '
SELECT
 [Without spaces] = '|' + LTrim(RTrim(@string_to_trim)) + '|'
,[With spaces:] = '|' + @string_to_trim + '|'
;
go
```

Results | Messages

	Without spaces	With spaces:
1	|This is some data|	| This is some data |

Figure: Result of previous SQL query

The REPLACE Function

(string_expression , string_pattern , string_replacement)

```
Select Replace('Bob Smith','Bob','Robert');
Select Replace('Bob Jim-Bob Smith','Bob','Robert');
go
```

Results | Messages

	(No column name)
1	Robert Smith

	(No column name)
1	Robert Jim-Robert Smith

Figure: Result of previous SQL query

The PATINDEX Function

('%pattern%' , expression)

```
Declare @Email varchar(50) = 'BSmith@MyCo.com';
SELECT
 [Name Ends] = PatIndex('%@%', @Email)
,[Domain Starts] = PatIndex('%.%', @Email)
;
go
```

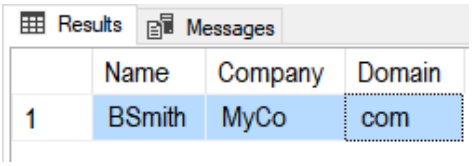
Results | Messages

	Name Ends	Domain Starts
1	7	12

Figure: Result of previous SQL query

The SUBSTRING Function

```
( expression ,start , length )
Declare @Email varchar(50) = 'BSmith@MyCo.com';
SELECT
 [Name] = SubString(@Email,0,PatIndex('%@%',@Email))
,[Company] = SubString(@Email,PatIndex('%@%',@Email) + 1, patindex('%.%',@Email) -
patindex('%@%',@Email) - 1)
,[Domain] = SubString(@Email,PatIndex('%.%',@Email) + 1,20)
go
```

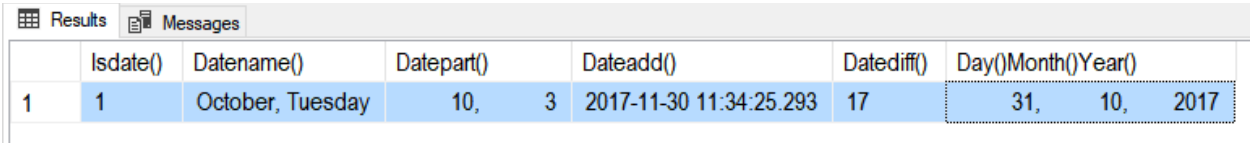


	Name	Company	Domain
1	BSmith	MyCo	com

Figure: Result of previous SQL query

Date/Time Functions

```
Declare @Date as DateTime = GetDate();
Select
 [Isdate()] = Isdate(@Date)
,[Datename()] = DateName(mm,@Date) + ', ' + DateName(Weekday,@Date)
,[Datepart()] = str(DatePart(mm, @Date)) + ', ' + str(DatePart(Weekday,@Date))
,[Dateadd()] = DateAdd(mm, 1, @Date)
,[Datediff()] = DateDiff(yy, '20000101', @Date)
,[Day()Month()Year()] = str(Day(@Date)) + ', ' + str(Month(@Date)) + ', ' + str(Year(@Date));
```



	Isdate()	Datename()	Datepart()	Dateadd()	Datediff()	Day()Month()Year()
1	1	October, Tuesday	10, 3	2017-11-30 11:34:25.293	17	31, 10, 2017

Figure: Result of previous SQL query

Soundex()

Often you need to clean-up incorrect data. Soundex() is an one function that can help you do a clean-up task that would normally be difficult for humans to do quickly.

https://docs.microsoft.com/en-us/sql/t-sql/functions/soundex-transact-sql

```
Select SOUNDEX ('Patient'), SOUNDEX ('Pateint'), SOUNDEX ('Peteint'), SOUNDEX ('Patint'), SOUNDEX
('Patit');
go
If Exists(Select Name from Sys.tables where Name = 'LookupBySound')
  Drop Table LookupBySound
go
Create Table LookupBySound
(ID int Primary Key Identity,[SoundExValue] nvarchar(100), Word nvarchar(100))
go
Insert Into LookupBySound Values ('P353', 'Patient');
go
If Exists(Select Name from Sys.tables where Name = 'DirtyData') Drop Table DirtyData
Create Table DirtyData
(ID int Primary Key Identity, DirtyDataValue nvarchar(100))
go
Insert Into DirtyData
Values
('Patient'), ('Pateint'), ('Peteint'), ('Patint'), ('Patit'), ('test');
go
Select * from LookupBySound;;
Select * from DirtyData;
go
Select DirtyData.DirtyDataValue, LookupBySound.Word
From DirtyData
Left Join LookupBySound
 ON Soundex(DirtyData.DirtyDataValue) = LookupBySound.SoundExValue
```

Partitioned or Windowed Functions

https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql

Partitioned or Windowed Functions allow you to group data differently than the standard Group By clause. Here is an example of a simple select statement with a Group By clause.

```
Select
stor_id
,[store min was] = min(qty)
,[store max was] = max(qty)
From Pubs.dbo.sales
Group By stor_id
Order By stor_id;
```

Results Messages

	stor_id	store min was	store max was
1	6380	3	5
2	7066	50	75
3	7067	10	40
4	7131	15	25
5	7896	10	35
6	8042	10	30

Figure: Result of previous SQL query

Note how just adding Qty doesn't work correctly, because each line becomes part of its own group

```
Select
 stor_id
,qty -- does not work correctly
,[store min was] = min(qty)
,[store max was] = max(qty)
From Pubs.dbo.sales
Group By stor_id, qty -- qty is used for grouping
Order By stor_id;
```

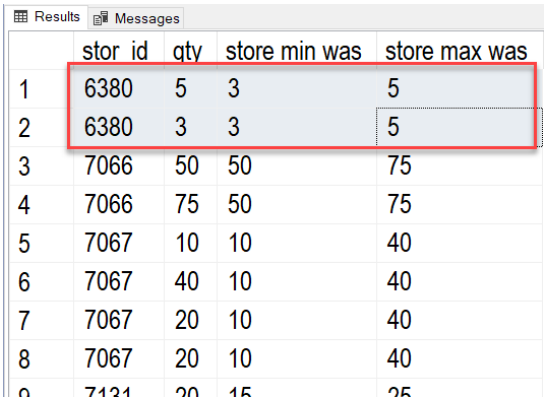
Results Messages

	stor_id	qty	store min was	store max was
1	6380	3	3	3
2	6380	5	5	5
3	7066	50	50	50
4	7066	75	75	75
5	7067	10	10	10

Figure: Result of previous SQL query

Using the Over-Partition function lets you group by only some of the columns.

```
Select
 stor_id
,qty
,[store min was] = min(qty) over(partition by stor_id)
,[store max was] = max(qty) over(partition by stor_id)
From Pubs.dbo.sales
Order By stor_id;
```



	stor_id	qty	store min was	store max was
1	6380	5	3	5
2	6380	3	3	5
3	7066	50	50	75
4	7066	75	50	75
5	7067	10	10	40
6	7067	40	10	40
7	7067	20	10	40
8	7067	20	10	40

Figure: Result of previous SQL query

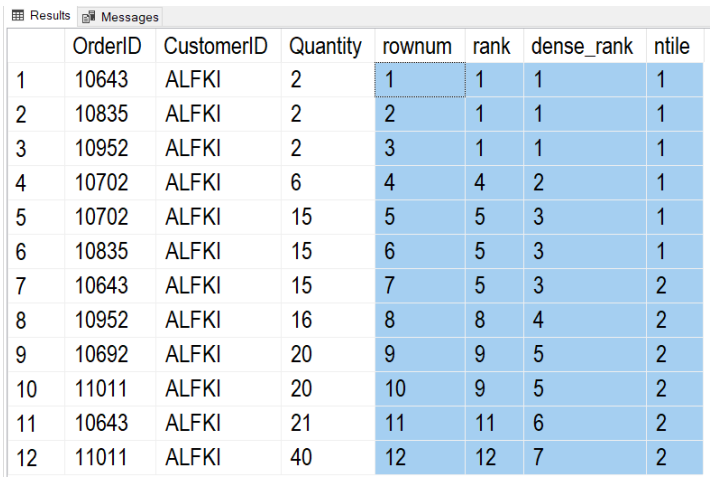
If I **add** both stor_id and **qty** to the partition the results are the **same as Group By.**

```
Select
 stor_id
,qty
,[store min was] = min(qty) over(partition by stor_id, qty)
,[store max was] = max(qty) over(partition by stor_id, qty)
From Pubs.dbo.sales
Order By stor_id;
```

Ranking Functions

"Ranking functions return a ranking value for each row in a partition. Depending on the function that is used, some rows might receive the same value as other rows." -- https://docs.microsoft.com/en-us/sql/t-sql/functions/ranking-functions-transact-sql

```
SELECT o.OrderID, o.CustomerID, od.Quantity,
  -- Ignore duplicate Values in the (Quantity) column
  ROW_NUMBER() OVER(ORDER BY od.Quantity) AS rownum,
  -- Group duplicate Values in the (Quantity) column and tells how many rows have come before it
  RANK() OVER(ORDER BY od.Quantity) AS rank,
 -- Groups duplicate Values in the (Quantity) column and tells what the last row NUMBER came before it
  DENSE_RANK() OVER(ORDER BY od.Quantity) AS dense_rank,
-- Divide rows into groups based in the number of groups you ask for. Ntile(2) would be 2 groups.
  NTILE(2) OVER(ORDER BY od.Quantity) AS ntile
FROM Northwind.dbo.Orders as o Join Northwind.dbo.[Order Details] as od
 On o.OrderID = od.OrderID
Where CustomerID = 'ALFKI'
ORDER BY od.Quantity;
```



	OrderID	CustomerID	Quantity	rownum	rank	dense_rank	ntile
1	10643	ALFKI	2	1	1	1	1
2	10835	ALFKI	2	2	1	1	1
3	10952	ALFKI	2	3	1	1	1
4	10702	ALFKI	6	4	4	2	1
5	10702	ALFKI	15	5	5	3	1
6	10835	ALFKI	15	6	5	3	1
7	10643	ALFKI	15	7	5	3	2
8	10952	ALFKI	16	8	8	4	2
9	10692	ALFKI	20	9	9	5	2
10	11011	ALFKI	20	10	9	5	2
11	10643	ALFKI	21	11	11	6	2
12	11011	ALFKI	40	12	12	7	2

Figure: Result of previous SQL query

Extracting a Percentage

A common task is the calculate the percentage of one value compared to another. Here is how it can be done with SQL code.

Tip: Creating an artificial column just for sorting my data is useful in reporting!

```
Select [sortcolumn] = 'a', [result] = (3 / 9) Union
               Select 'b', (3 * 100 / 9) Union
               Select 'c', Cast((3 * 100 / 9) as float) Union
               Select 'd', (1. * 3 * 100 / 9) Union -- No need to cast to float now!
               Select 'e', (1. * 12 * 100 / 706) – Ex. 12 products of 706 total products
Order By [sortcolumn]
Go
```

Results | Messages

	sortcolumn	result
1	a	0
2	b	33
3	c	33
4	d	33.333333
5	e	1.699716

Figure: Result of previous SQL query

Here is an example of how that code could be used.

```
Select
 [stor_id]
,[title_id]
,[QtyByTitle] = Sum(Qty) Over(Partition By [stor_id])
,[QtyByStoreAndTitle] = Sum(Qty) Over(Partition By [title_id], [stor_id]) -- The Order Matters here!
,[Title Percent Per Store] = Cast( (1. * (Sum(Qty) Over(Partition By [title_id], [stor_id])) * 100) /
                                   Sum(Qty) Over(Partition By [stor_id]) as decimal(10,2))
From pubs.dbo.sales
Order By 1;
```

Results | Messages

	stor_id	title_id	QtyByTitle	QtyByStoreAndTitle	Title Percent Per Store
1	6380	BU1032	8	5	62.50
2	6380	PS2091	8	3	37.50
3	7066	PC8888	125	50	40.00
4	7066	PS2091	125	75	60.00
5	7067	PS2091	90	10	11.11
6	7067	TC3218	90	40	44.44
7	7067	TC4203	90	20	22.22
8	7067	TC7777	90	20	22.22

Figure: Result of previous SQL query

Lag and Lead

These functions are very useful in reporting! It shows the previous or following values based on a given group of values.

Here is an example that shows the following year's values.

```
Select
  [OrderYear] = Year(OrderDate)
 ,[YearlyTotalQty] = Sum(Quantity)
 ,[PreviousYearlyTotalQty] = lead(Sum(Quantity)) Over(Order By Year(OrderDate))
 From Northwind.dbo.Orders as O
  Join Northwind.dbo.[Order Details] as OD
   On o.OrderID = o.OrderID
 Group By Year(OrderDate);
Go
```

Results Messages

	OrderYear	YearlyTotalQty	PreviousYearlyTotalQty
1	1996	7800184	20937336
2	1997	20937336	13855590
3	1998	13855590	NULL

Figure: Result of previous SQL query

Here is another example that shows the shows the previous year's values.

```
Select
  [OrderYear] = Year(OrderDate)
 ,[YearlyTotalQty] = Sum(Quantity)
 ,[PreviousYearlyTotalQty] = Lag(Sum(Quantity)) Over(Order By Year(OrderDate))
 From Northwind.dbo.Orders as O
  Join Northwind.dbo.[Order Details] as OD
   On o.OrderID = o.OrderID
 Group By Year(OrderDate);
go
```

Results Messages

	OrderYear	YearlyTotalQty	PreviousYearlyTotalQty
1	1996	7800184	NULL
2	1997	20937336	7800184
3	1998	13855590	20937336

Figure: Result of previous SQL query

Using Functions for Reporting

To create reporting queries, you start off with a simple Select statement and then build on it by adding more and more detailed code to finally get what you want as a result.

To create reporting queries, you start off with a simple Select statement like this one.

```
Select Distinct
  OrderDate
 From Northwind.dbo.Orders;
```

Results Messages

	OrderDate
1	1996-07-04 00:00:00.000
2	1996-07-05 00:00:00.000
3	1996-07-08 00:00:00.000

Figure: Result of previous SQL query

Next, add simple functions and test the results!

```
Select Distinct
  [OrderYear] = Year(OrderDate)
 From Northwind.dbo.Orders;
```

Results Messages

	OrderYear
1	1998
2	1996
3	1997

Figure: Result of previous SQL query

Then, add more columns, functions, or tables as needed.

```
Select Distinct
  [OrderYear] = Year(OrderDate)
 ,[YearlyTotalQty] = Sum(Quantity)
 From Northwind.dbo.Orders as O
  Join Northwind.dbo.[Order Details] as OD
   On o.OrderID = o.OrderID
 Group By Year(OrderDate);
```

Results Messages

	OrderYear	YearlyTotalQty
1	1998	13855590
2	1996	7800184
3	1997	20937336

Figure: Result of previous SQL query

Keep adding more complex function as needed (like using the Lag Function).

```
Select
  [OrderYear] = Year(OrderDate)
 ,[YearlyTotalQty] = Sum(Quantity)
 ,[PreviousYearlyTotalQty] = Lag(Sum(Quantity)) Over(Order By Year(OrderDate))
 From Northwind.dbo.Orders as O
  Join Northwind.dbo.[Order Details] as OD
   On o.OrderID = o.OrderID
 Group By Year(OrderDate);
go
```

Results Messages

	OrderYear	YearlyTotalQty	PreviousYearlyTotalQty
1	1996	7800184	NULL
2	1997	20937336	7800184
3	1998	13855590	20937336

Figure: Result of previous SQL query

Continue adding more features until you get what you are looking for.

```
Select
  ProductName
 ,[OrderYear] = Year(OrderDate)
 ,[YearlyTotalQty] = Sum(Quantity)
 ,[PreviousYearlyTotalQty] =
        IIF(Year(OrderDate) = 1996, 0, Lag(Sum(Quantity)) Over (Order By ProductName,Year(OrderDate)))
 ,[Bad-PreviousYearlyTotalQty] = IsNull(Lag(Sum(Quantity)) Over (Order By ProductName,Year(OrderDate)),
0) -- Don't use Is Null!
 From Northwind.dbo.Orders as O
  Join Northwind.dbo.[Order Details] as OD
   On o.OrderID = o.OrderID
  Join Northwind.dbo.Products as P
   On OD.ProductID = P.ProductID
 Group By ProductName, Year(OrderDate);
go
```

Results Messages

	ProductName	OrderYear	YearlyTotalQty	PreviousYearlyTotalQty	Bad-PreviousYearlyTotalQty
1	Alice Mutton	1996	148656	0	0
2	Alice Mutton	1997	399024	148656	148656
3	Alice Mutton	1998	264060	399024	399024
4	Aniseed Syrup	1996	49856	0	264060
5	Aniseed Syrup	1997	133824	49856	49856
6	Aniseed Syrup	1998	88560	133824	133824
7	Boston Crab Meat	1996	167656	0	88560

Figure: Result of previous SQL query

Once you feel getting too complex or if you think you might reuse the results, create a reporting View like this one.

```
Create -- Drop
View vProductOrderQtyByYear
AS
Select
  ProductName
 ,[OrderYear] = Year(OrderDate)
 ,[YearlyTotalQty] = Sum(Quantity)
 ,[PreviousYearlyTotalQty] = IIF(Year(OrderDate) = 1996, 0, Lag(Sum(Quantity)) Over (Order By
ProductName,Year(OrderDate)))
 From Northwind.dbo.Orders as O
  Join Northwind.dbo.[Order Details] as OD
   On o.OrderID = o.OrderID
  Join Northwind.dbo.Products as P
   On OD.ProductID = P.ProductID
 Group By ProductName, Year(OrderDate);
Go
```

When using the View, you can always add on more functions -- as needed. For example, our current view makes it easy to create a Key Performance Indicators (KPIs) report

```
Select
  ProductName
 ,[OrderYear]
 ,YearlyTotalQty
 ,PreviousYearlyTotalQty
 ,[QtyChangeKPI] = Case
   When YearlyTotalQty > PreviousYearlyTotalQty Then 1
   When YearlyTotalQty = PreviousYearlyTotalQty Then 0
   When YearlyTotalQty < PreviousYearlyTotalQty Then -1
   End
 From vProductOrderQtyByYear
Go
```

Results Messages

	ProductName	OrderYear	YearlyTotalQty	PreviousYearlyTotalQty	QtyChangeKPI
1	Alice Mutton	1996	148656	0	1
2	Alice Mutton	1997	399024	148656	1
3	Alice Mutton	1998	264060	399024	-1
4	Aniseed Syrup	1996	49856	0	1
5	Aniseed Syrup	1997	133824	49856	1
6	Aniseed Syrup	1998	88560	133824	-1
7	Boston Crab Meat	1996	167656	0	1

Figure: Result of previous SQL query

User Defined Functions

In addition to SQL Server's built-in functions, you can create **custom functions**. These are **often called User Defined Functions** or just **UDFs**. There are **two basic types of functions**; functions that **return a table of values** and functions that **return a single value**.

Scalars Functions

You can create **UDFs to return a single (scalar) value as an expression**. (Note: In MS SQL, you **must use include the schema name in scaler UDFs**, in this case, dbo).

Unlike parameters in table functions, parameters in scalar functions are very useful!

```
Create Function dbo.MultiplyValues(@Value1 Float, @Value2 Float)
 Returns Float
 As
  Begin
   Return(Select @Value1 * @Value2);
  End
go
-- Calling the function
Select Tempdb.dbo.MultiplyValues(4, 5);
go
```

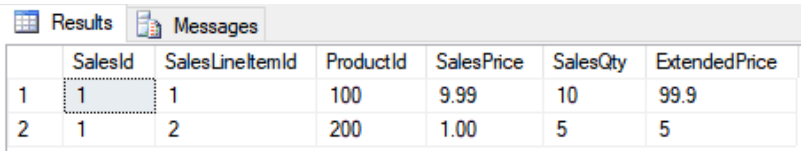
If you want **to apply the function to each row** of a result set, you use the new function like this:

```
Create table dbo.SalesDetails
( SalesId int, SalesLineItemId int
, ProductId int
, SalesPrice money
, SalesQty int,
 Primary key(SalesId, SalesLineItemID)
);
go
Insert Into dbo.SalesDetails
(SalesId,SalesLineItemId,ProductId,SalesPrice,SalesQty)
Values
 (1,1,100,$9.99,10)
,(1,2,200,$1.00,5)
Go

Select
 SalesId
,SalesLineItemId
,ProductId
,SalesPrice
,SalesQty
,dbo.MultiplyValues(SalesPrice,SalesQty) as ExtendedPrice
```

```
From dbo.SalesDetails
```

Here are the results:



Results | Messages

	SalesId	SalesLineItemId	ProductId	SalesPrice	SalesQty	ExtendedPrice
1	1	1	100	9.99	10	99.9
2	1	2	200	1.00	5	5

Figure: Results of using the Multiply-Values function in a query

Using UDFs for Check constraints

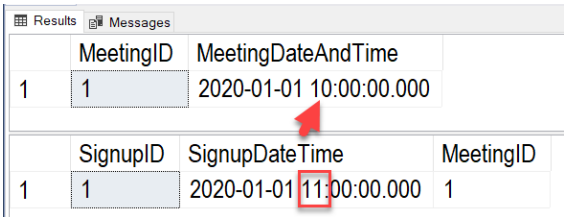
Custom Scalar functions are sometimes used for Check constraints because you cannot otherwise reference a column in another table. Here is an example

```
Set NoCount ON; -- Turns of the (1 row affected) messages
Go
Use TempDB;
Go
If Exists (Select Name From Sys.Tables where Name = 'SignupForMeetings')
  DROP TABLE SignupForMeetings;
Go
If Exists (Select Name From Sys.Tables where Name = 'Meetings')
  DROP TABLE Meetings;
Go

-- Make dependent tables.
CREATE TABLE Meetings (MeetingID int Primary Key, MeetingDateAndTime datetime);
Go
INSERT INTO Meetings (MeetingID, MeetingDateAndTime)
 VALUES (1,'1/1/2020 10:00:00');
Go

CREATE TABLE SignupForMeetings
( SignupID int PRIMARY KEY
, SignupDateTime datetime
, MeetingID int Foreign Key References Meetings(MeetingID)
);
Go
INSERT INTO SignupForMeetings (SignupID, SignupDateTime, MeetingID)
VALUES (1, '1/1/2020 11:00:00', 1) -- Opps! This is One hour AFTER the meeting
Go

SELECT MeetingID, MeetingDateAndTime From Meetings;
SELECT SignupID, SignupDateTime, MeetingID From SignupForMeetings;
Go
```



Results | Messages

	MeetingID	MeetingDateAndTime
1	1	2020-01-01 10:00:00.000

	SignupID	SignupDateTime	MeetingID
1	1	2020-01-01 11:00:00.000	1

Figure: Incorrect meeting signup data is allowed

```
-- Remove that row and work on a way to Fix this issue!
DELETE FROM SignupForMeetings WHERE SignupID = 1;
Go
```

```
-- Make a function that will get the meeting date and time based on a meeting ID
CREATE or ALTER FUNCTION dbo.fGetMeetingDateTime (@MeetingId int)
RETURNS DATETIME
AS
 BEGIN
    RETURN (SELECT MeetingDateAndTime
            FROM Meetings
            WHERE Meetings.MeetingID = @MeetingID);
 END
Go

-- Test the function
SELECT dbo.fGetMeetingDateTime(1);
SELECT IIF(CAST('1/1/2020 07:00:00' as datetime) < dbo.fGetMeetingDateTime(1), 'TRUE', 'FALSE'),'Before
Start';
SELECT IIF(CAST('1/1/2020 11:00:00' as datetime) < dbo.fGetMeetingDateTime(1), 'TRUE', 'FALSE'), 'After
Start';
Go
```

Results Messages

	(No column name)
1	2020-01-01 10:00:00.000

	(No column name)	(No column name)
1	TRUE	Before Start

	(No column name)	(No column name)
1	FALSE	After Start

Figure: Results of the function test

```
-- Now, create a constraint that checks that a signup is before the meeting time
ALTER TABLE SignupForMeetings
  ADD CONSTRAINT ckSignupVsMeetingDateTime
    CHECK(SignupDateTime < dbo.fGetMeetingDateTime(MeetingID));
Go

-- Test the check constraint
INSERT INTO SignupForMeetings
(SignupID, SignupDateTime, MeetingID)
VALUES
(1, '1/1/2020 9:00:00', 1) -- One hour BEFORE the meeting
Go

INSERT INTO SignupForMeetings
(SignupID, SignupDateTime, MeetingID)
VALUES
(1, '1/1/2020 11:00:00', 1) -- One hour AFTER the meeting
Go
Msg 547, Level 16, State 0, Line 69
The INSERT statement conflicted with the CHECK constraint "ckSignupVsMeetingDateTime". The conflict
occurred in database "tempdb", table "dbo.SignupForMeetings".
The statement has been terminated.

SELECT MeetingID, MeetingDateAndTime From Meetings;
SELECT SignupID, SignupDateTime, MeetingID From SignupForMeetings;
Go
```

Results Messages

	MeetingID	MeetingDateAndTime
1	1	2020-01-01 10:00:00.000

	SignupID	SignupDateTime	MeetingID
1	1	2020-01-01 09:00:00.000	1

Figure: Results of the previous statements.

Creating Advanced GitHub Pages

In this module's assignment, you create a GitHub webpage. This web page should be **like the Word documents you created in previous assignments**. To create your page, you need to know some basic commands in a language called "markdown."

"GitHub combines a syntax for formatting text called GitHub Flavored Markdown with a few unique writing features.

Markdown is an easy-to-read, easy-to-write syntax for formatting plain text.

We've added some custom functionality to create GitHub Flavored Markdown, used to format prose and code across our site." (https://help.github.com/en/github/writing-on-github/about-writing-and-formatting-on-github, 2019)

Creating a Markdown File

To demonstrate an example, I create a **new GitHub repository called "ITFnd100-Mod07"** as shown in this figure:

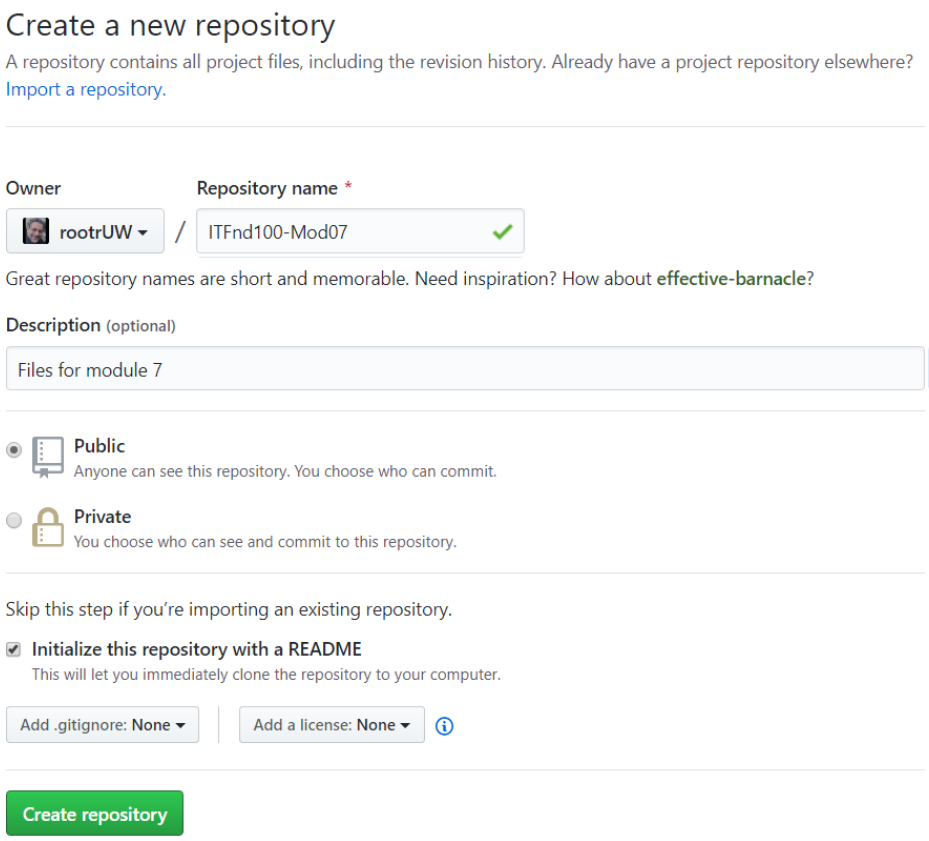


Figure. Creating a new GitHub repository

Next, I **create a new "docs" folder with a file called "index.md"** inside of it. When I do, I need to **type or paste in some text** for the new file before it is created in the folder. In Figure 17, I have typed in some **simple markdown commands to format my document**. Once I have at least some text in the file I can create the folder and file on my repository.

Figure. Creating a new Index.md file in the "docs" folder

Formatting the Page

I **use the "Preview" tab to see what my new markdown file looks** like each time I modify the text (Figure 18). The look changes based on what markdown commands I use. While there are lots of commands available, let's **focus on the ones you need for the assignment.**

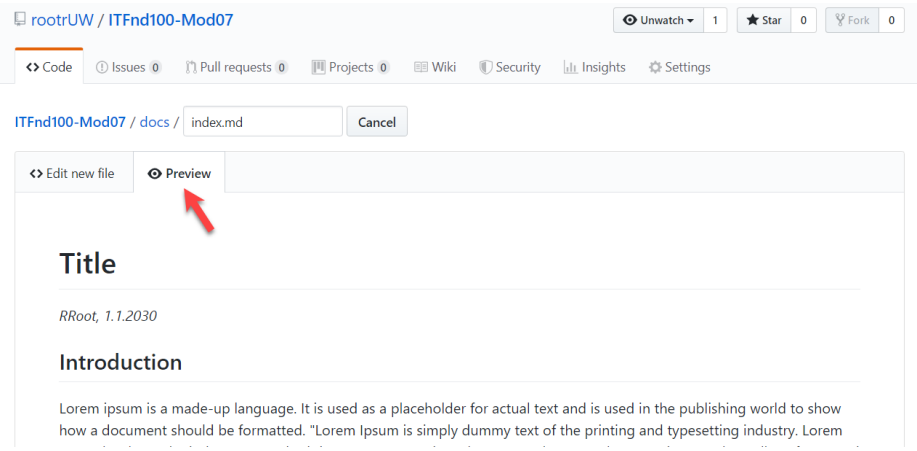


Figure. Previewing a GitHub page

Creating Text Headers

The hash, **"#" symbol, indicates a header**. You use one or more hash symbols to define the level of the header. Oddly, the more hash marks you use, the smaller the header size.

```
# Title
## Introduction
## Topic 1
### Subtopic
## Summary
```

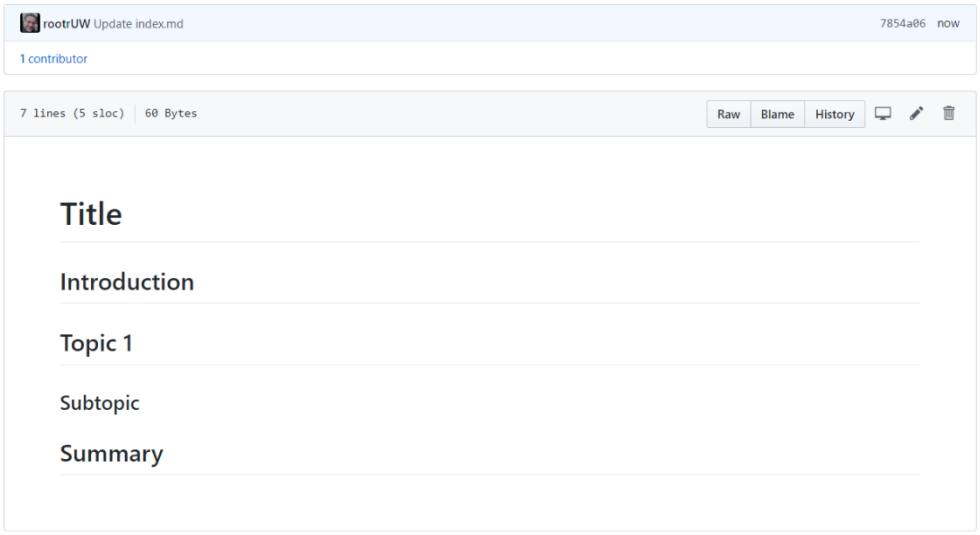


Figure. The results use header markdown commands

Formatting Text

You can make text **bold using two Astrid "**"** symbols or **italic using a single Astrid**.

```
**Dev:** *RRoot*
**Date:** *1.1.2030*
```

Listing 17

Note: *There are* ***two invisible space characters after the text "*RRoot* that force a newline into the page****. Newlines can be tricky in this language, and you may need to experiment to style the page to your liking.*

Title

Dev: *RRoot*
Date: *1.1.2030*

Figure. Creating told text

Adding Code Samples

To add some **sample code** to the page, you **use the backtick (`) symbol,** as shown in Listing 18.

````
# Title
**Dev:** *RRoot*
**Date:** *1.1.2030*

## Structured Error Handling (Try-Except)
When you are programming, you fix your bugs immediately and make sure the code runs smoothly.
However, it often happens that other people introduce new bugs when they use your program.

### Raising Custom Errors
Python automatically generates errors based on conditions defined by the Python Runtime. However,
you can also "raise" errors based on custom conditions (Listing 13).

```
# ---------------------------------------------- #
# Title: Listing 13
# Description: A try-catch with manually raised errors
# ChangeLog: (Who, When, What)
````
````

```
# RRoot,1.1.2030,Created Script
# ------------------------------------------------ #

try:
    new_file_name = input("Enter the name of the file you want to make: ")
    if new_file_name.isnumeric():
        raise Exception('Do not use numbers for the file\'s name')
except Exception as e:
    print("There was a non-specific error!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```
Listing 13

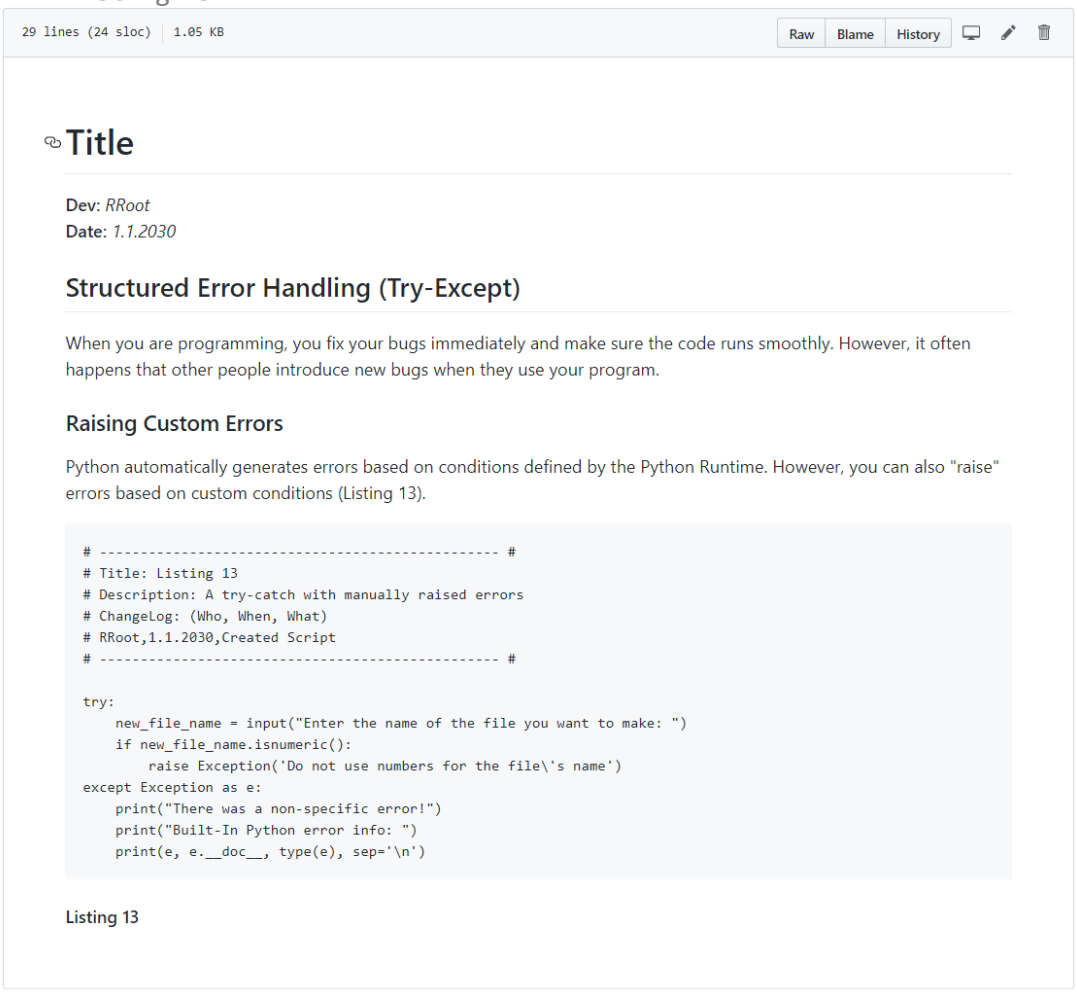


Figure. The results of adding code to the page

***Tip:** The keyboard key for the **backtick symbol (`)** is shared with the tilde symbol (~) **above the Tab key on both a Windows and Mac keyboard**.*

Adding Images

To **add an image** to a page, perform the following steps.

1. **Save image to your computer's hard drive**. In MS Word, you do this by right-clicking the image and using the "Save as Picture..." option of the context menu.

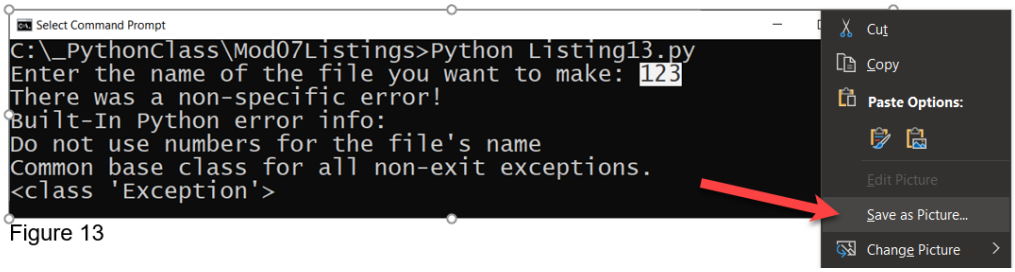


Figure 13

Figure. Saving an image from Word to a drive

2. **Upload the image to your GitHub** repository's "docs" folder (or a subfolder if you wish to be more organized.

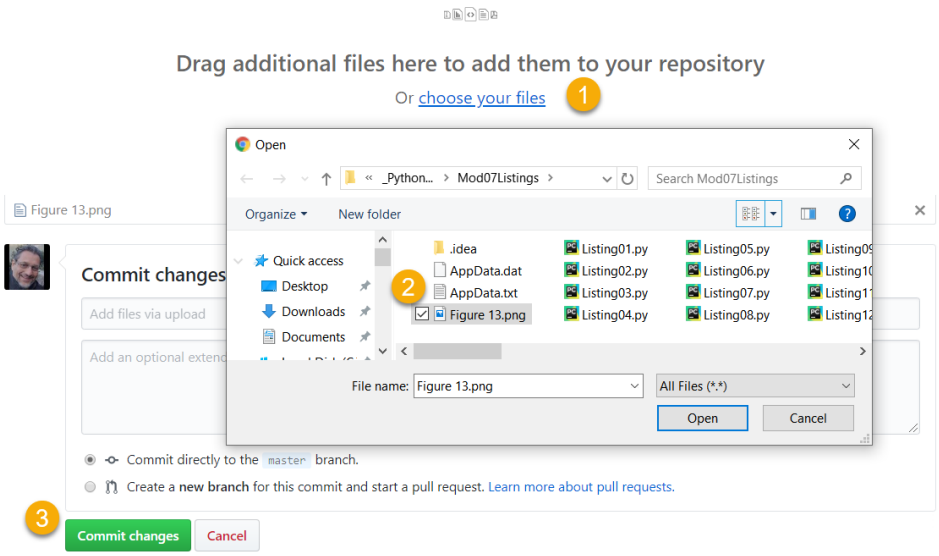


Figure. Uploading the saved image file to GitHub

3. **Copy the web address for the file**, by locating your file on the GitHub page, then right-clicking it to access the context menu. From there, use the "Copy link address" option, or an equivalent one if you are using a browser other than Chrome.

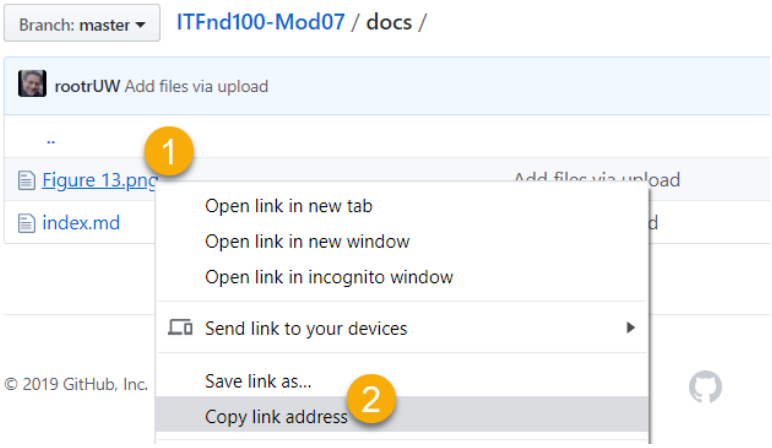


Figure. Copying the file's URL on GitHub with a Chrome browser

4. **Link your image to your page using an image link code.** The syntax for the image link is:

```
![alt text](web address "tooltip text")
```

Here is an example:

```
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```

Listing 13

```
![Results of Listing 13](Figure13.png "Results of Listing 13")#### Figure 13. The results of Listing 13
```

Note: If you cannot get the images to work, try using HTML tags instead. https://www.w3schools.com/tags/tag_img.asp

The **first part of command** indicates the **alternate text** use by screen readers. The **second** part of the command indicates the **URL to the file**. The **third** part of the command indicates the text you want a **tool tip** to display. Figure 24 shows the results of the command.

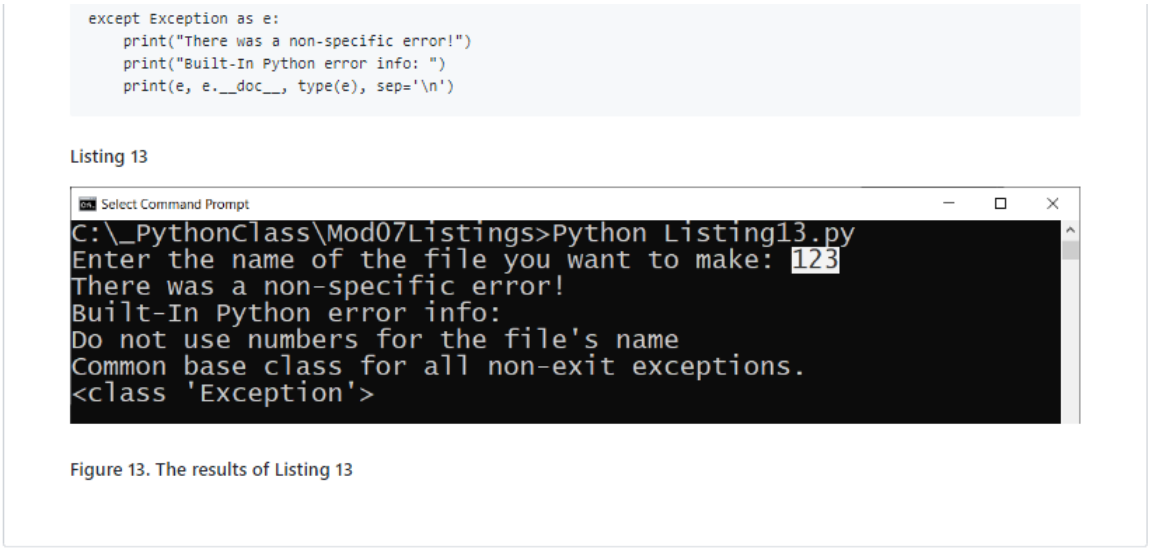


Figure. The results of using a Markdown image link on a GitHub page

Important: *It is best to use a relative path for files that are in your folder, and only use the "hard coded" physical path for files outside of your website.*

Learning More

There is much information on the Internet about the Markdown language, but you **should find all you need for this course on this one webpage**:

https://help.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax

Important*: Learning to use Markdown and Jekyll could well be the topic of a complete course, but in this course,* ***you do NOT need to know much about Markdown*** *programming.* ***Please use only the basics shown in this module*** *instead of more advanced features* ***and do not worry about getting the format perfect!***