Intro to Deluge Workflows

in Zoho Creator

Table of Contents

Who should read this book?

This ebook is essential reading for anyone looking to build automation into their Creator applications. If you don't have any programming experience, this ebook will **teach you how to get started** automating your workflow. If you do have programming experience, this guide will help you **learn the fundamentals of Deluge syntax**.

People often say that learning to code is like learning a language. While this is a helpful analogy, and one we'll use while explaining some programming concepts, you shouldn't take it literally. Programming languages have structure and vocabulary like human languages. However, since programming is all about telling a computer what to do and when, you only need to learn the commands that will be helpful to your workflow.

What is Deluge?

Most Creator applications have three main parts: forms for collecting information, reports for analyzing information, and Deluge scripts for creating workflows. Deluge is Zoho's scripting language that lets you automate Creator applications and customize other software like Zoho CRM.

To understand what we mean by workflows, just think about what's happening while you're working. You are constantly receiving information, making judgments about it, and deciding how to act next. This process is your **personal workflow.**

Some of the steps in your workflow are probably complicated and require your complete attention. But usually there are a few steps that are easy to do; in fact these easy steps may even feel tedious when you do them over and over. For example, your workflow may involve repetitive calculations, or sending the same form messages out every day.

By learning to write Deluge scripts you'll be able to add automation to your Creator applications.Deluge can turn your workflow from an active process that constantly requires your attention, to a passive process where you only have to participate in a few key steps.

The Layers of Creator

Zoho Creator consists of two essential layers. The first layer is Creator's graphical interface, where you drag and drop fields in the form builder, or create a report. The second layer is the Deluge scripting layer, which is used to customize forms and create workflows beyond what the graphical interface offers.

These two layers allow you to build the forms, reports, and workflow rules that make up your custom application. The interface layer is laid on top of the Deluge scripting layer to make a more user-friendly experience.

Without these two layers, you would find custom application development to be much more time consuming and expensive. Not only would you need to know a programming langauge like Java, Python, or PHP, but you would also need to manage, secure, and update a server, database, network, and more. Zoho Creator's interface and Deluge scripting layers allow you to build custom applications within hours, rather than weeks or months.

In this ebook you will begin learning the basics of the Deluge scripting layer.

What you can do with Deluge

Deluge adds logic to your Creator application, making it much more powerful. Deluge is essential when it comes to automating, integrating, and customizing your application. Here are some practical examples of what you can do in each of these areas:

Automation – Think about the repetitive tasks you do. How much time could you save if you had an application to do them for you?

Notifications

-Email management when stock is running low.

-Text employees when office visitors arrive to meet with them.

Schedules

-Send a thank you email an hour after customers make a purchase.

-Get a weekly email with your schedule.

-Send quarterly reports.

Forms

-Give a discount based on region or the amount ordered.

-Display only the employees who will be available when scheduling shifts.

-Automatically fill in someone's email after they choose their name.

Formulas

-Generate quotes and estimates.

What you can do with Deluge

Approval process

-Set up a approval(eg: leave/travel request) process in the organization.

-Select one or multiple approvers.

-Specify the actions to execute when a request is approved or rejected.

Payment

-Collect invoice amount for the ordered items.

-Collect registration fee for a event like workshop or a certification programme

-Collect monthly fee for the rented vehicles.

What you can do with Deluge

Integration – Your data is in a few applications that aren't compatible. Pull info in and push updates out using API calls to get all your software working together.

Zoho applications

-Collect info in Zoho Creator and use it to add leads in Zoho CRM.

-Pull data from Zoho Invoice and analyze it with your other financial data in Zoho Creator.

Other cloud solutions

-Send invoices from sales in Zoho Creator to external programs.

On premise solutions

-MS Access, MySQL, SAP, etc.

Customization – Make dynamic forms that change depending on what information is being added to then. Add buttons to your application's records that can update information, send out emails, or do whatever else you need them to do.

Interactive Forms

-Show different questions to different people to conduct targeted surveys.

-Display extra instructions when someone checks a box asking for help.

Buttons

-Email someone by clicking a button next to their record.

-Quickly change a status by clicking a button.

Deluge 101

There are a few key concepts to understand before you actually hop in and start writing script in Deluge.

Right now, you may not know what the words *variable, boolean, string, conditionals,* or *integer* mean in a programming context. But don't let that intimidate you. These terms are a lot more straightforward than you might assume. Plus, since they're used in many programming languages, the things you learn here could be relevant if you continue learning about programming beyond Deluge scripting. Let's jump into defining these terms so we can get you writing some Deluge!

Syntax

When speaking English, we structure our sentences with syntax so people can understand what we're saying. For example, if we were eating dinner and asked someone to pass the potatoes, we wouldn't say: "The pass potatoes please." That would be incorrect syntax.

Programming languages use syntax in a similar way to English. The commands that you write in Deluge need to be properly structured for Creator to understand what you want it to do.

Its important to remember that while computers are extremely good at following directions, they can only follow directions that are perfectly written. Whereas most English speakers can make sense of the incorrect syntax, "The pass potatoes please," computers cannot. Therefore, the most important part of learning a programming language is the syntax.

Creator makes learning Deluge easier than learning other programming languages on your own. This ebook will teach you Deluge, but Creator will also assist you as you write your scripts.

Heads Up

Creator's drag-and-drop script builder provides the syntax for all the scripts you'll write.

Variables

Put simply, variables are used for data storage. Variables store and manipulate the data attributed to them. A metaphor can make this clearer. The glass cup below holds whatever you put into it. Similarly, a variable stores whatever data you assign to it.

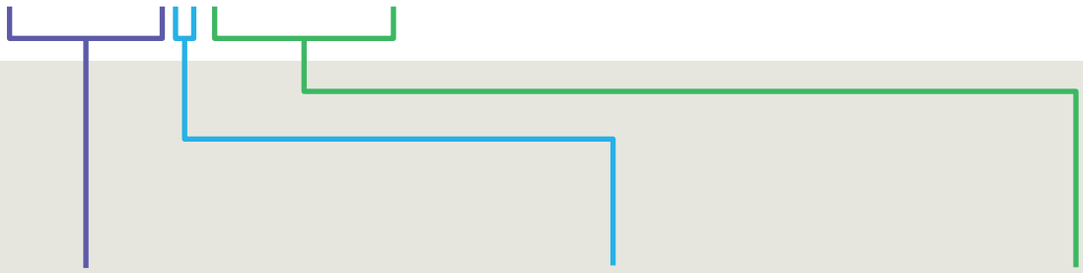
To store data in a variable, or **declare a variable**, you have to write with a specific **syntax**. "Declaring a variable" is the phrase used in most programming languages for creating a variable and storing something inside it. Let's see some examples.

```
variable_name = "value" ;

juice_glass = "lemonade" ;

soda_glass = "root beer" ;

milk_glass = "whole milk" ;
```



variable name: it's important to name your variable something that indicates what its purpose is. In this case, I've named them according to the type of juice that is stored in them. All of our variable names have underscores because if you add a space, Creator will get confused and think you're talking about two different variables.

; **and** =: these are part of Deluge's syntax. The = is used to tell Deluge that the data stored in the variable name is equal to the value given. The ; is used like a period at the end of a sentence. It signals the end of a statement.

value: this is what we've inserted into each variable. The data in a variable is called its value.

Now that we know how variables are declared, let's flesh out the concept a little bit more and get into understanding the types of data you can store in a variable.

Data Types

Continuing with the drink metaphor used previously, there are certain types of drinks that fit into the same category. For example, there are multiple types of soda, but they are all soda; there are multiple flavors of juice, but they are all juices. You can think of a data type in Deluge like a category of drink: a juice variable could contain many different drinks, so long as those drinks are different kinds of juice.

Data Types

Deluge has five main data types, and Creator treats each one differently. Deluge has additional, more advanced data types like lists and maps, but we'll cover those in other guides. Below are the five main data types.

1.**Text**	"laptop"	String
2. **Number**	10	Integer (or bigint)
3. **Decimal**	2.2	Decimal (or long or float)
4. **Decision**	true or false	Boolean
5. **Date/Date time**	'05-Mar-2017'	Date/Date Time

The first time you declare a variable, you're telling Creator what data it's storing and what type of data it should always store. A variable's **value can change** multiple times, but its **data type won't change** unless you use a special function.

If this seems confusing, remember the drinks metaphor that we used. You could declare a variable as a juice variable, and then change the value of the variable as many times as you want. The value could change from orange juice, to cranberry juice, and then to apple juice. However, you could not change the juice variable to a soda variable. To change a juice variable to a soda variable would require using a special function.

Now let's go through a few examples, where you can guess which data type is being displayed.

Guess the Data Type

Guess the data type of x below. The correct answer is in pink on the right.

1	x = "15 Years";	string
2	x = "15";	string
3	x = 15;	number
4	x = 15.0;	decimal
5	x = "true";	string
6	x = true;	boolean

These pink cream pages will include quizzes and extended examples.

Operators

Operators are difficult to define without the use of common programming jargon. Operators are characters that let you manipulate or compare two values. This is a lot easier to grasp with examples.

There are three main kinds of operators, but for now we'll focus on **arithmetic operators**, which are used for writing equations. We'll cover the other types of operators later in this guide.

Arithmetic Operators

`+` `-` `/` `*` `%`

Examples

`5 + 2 = 7`

`5 - 2 = 3`

`5 * 2 = 10`

`5 / 2 = 2.5`

`5 % 2 = 1`

Arithmetic operators should look familiar to you. `/` is for division and `*` is for multiplication. `%` isn't used as often. `%` gives you the remainder when you divide the number on the left by the number on the right. The `+` operator is special because it can be used with text as well as with numbers. The `+` operator lets you combine a few pieces of text together to create one longer piece.

Operators

When you start using variables in equations, you'll see that **a variable's value can change multiple times** in one script. Here's an example.

```
a = 3;
b = 2;
a = a + b;
```

In the beginning of this script, the variable **a equals 3**. After adding another variable to it, variable **a equals 5**. Creator reads your scripts one line at a time. This means that when you change a variable's contents, it will only affect that variable moving forward. Previous instances of it won't be altered.

Since variables contain different data at different times, it can be tricky to find mistakes in a long script. The **info** and **alert** commands let you see a variable's contents. By placing the info command at different places in your script, you can see what information is stored in a variable at different points in time. This process is called **debugging** because it helps you find mistakes, or bugs, in your scripts.

Operators

Let's look at a couple examples of debugging using the info command. We'll declare the value of the variable x, request that Creator return the value, then declare a new value for x.

```
x = 10;
info x;
x = 15;
```

Result: 10

Explanation: Creator evaluates scripts from beginning to end. Since the info command was used before the value of x changed to 15, it will return the last value that was stored in x.

```
x = 10;
x = 15;
info x;
```

Result: 15

Explanation: Since the info command was placed after the value of x was changed, Creator will return the most recent value stored in x.

Operators & Strings

Let's use some operators to combine strings. First, we'll declare a couple variables.

```
First_Name = "Bruce" ;

Last_Name = "Wayne" ;
```

Now, we'll declare another variable that adds together the two we just introduced.

```
First_Name = "Bruce" ;

Last_Name = "Wayne" ;

Full_Name = First_Name + Last_Name;
```

Let's see what happens when we add an **info** command to make Creator tell us what's stored in our Full_Name variable.

```
First_Name = "Bruce" ;

Last_Name = "Wayne" ;

Full_Name = First_Name + Last_Name;

info Full_Name;
```

Result: BruceWayne

Operators:
Operators & Strings

Creator gave us an accurate output, but it wasn't exactly what we intended: there's no space between the first and last name.

To format the name correctly, we have to tell Creator to insert a space between the variables holding the first and last names. Remember, since we're working with text strings we have to put our space in quotes.

```
First_Name = "Bruce" ;

Last_Name = "Wayne" ;

Full_Name = First_Name + " " + Last_Name;

info Full_Name;
```

Result: Bruce Wayne

You may have noticed there are multiple ways to solve this formatting problem. Rather than adding a space between the first and last name variable, you can add a space to the value of your variables. For example, you could add a space after Bruce or before Wayne. This would look like “Bruce ” or “ Wayne”. There’s no ideal way to do this; it’s up to you.

Conditionals

Conditionals give your applications the power to react differently depending on how people use them. Conditionals, also called control statements, are like sets of instructions. It's easiest to think of them as **if, then statements**: if x happens, then do y. They're called conditionals because they tell Creator to check if a condition has been met before running the next step in your script.

Here's an example of a conditional written in plain English.

If it's a Sunday, **then** set my status to "Unavailable."

Here's how the same conditional might look written in Deluge:

```
if today == "Sunday" ;
{
        status  == "Unavailable" ;
}
```

There are three main conditionals:

1. If statements – These tell Creator how to act if a certain condition is met. Any script that includes conditionals must have at least one *if statement*.

2. Else if statements – When the condition in an *if statement* isn't met, you can tell Creator to check for other conditions using *else if statements*. This can be useful when you need to tell your application what to do in several different scenarios. If you write out what your Deluge script does in English, an else if would be like the word otherwise: If condition one is met, then do x. Otherwise, if condition two is met, then do y. When you put multiple *else if statements* in a row, Creator will only run the script from the first one whose conditions are met.

Conditionals

3. Else statements – *Else statements* tell Creator how to act if none of the conditions from your *if statements* or your *else if statements* have been met. *Else statements* don't include conditions.

When you write conditional statements, the condition you're looking for always goes inside parentheses. You don't need any semicolons in the part where you write your condition.The script that runs when the condition is met goes inside curly braces.

```
if (condition)
{
        Script that will run if the condition is met.
}
```

```
else if (condition)
{
        Script that will run only when the if condition isn't met, but the else if
        condition is met.
}
```

```
else
{
        Script that will run only if none of the other conditions above are met.
}
```

Writing Conditions

Before Creator runs the script in a conditional statement, it will check if the statement's condition has been met. In programming parlance, we say that a condition **is true** (or evaluates to true) if it has been met. If a condition isn't met, then it **is false** and the script inside your statement's curly braces won't run.

To tell Creator what conditions to look for, use **relational operators** and **logical operators**.

Relational Operators

Relational operators let you compare two values.

Operator	Meaning	Meaning for Date Variables
<	Less Than	Before
<=	Less Than or Equal to	Before or at the same time
>	Greater Than	After
>=	Greater Than or Equal to	After or at the same time
==	Equal to	At the same time
!=	Not Equal to	Not at the same time

Suppose we have a retail businesses and we want to limit the amount of items each customer can buy. Here's how you'd write that as an if/then:

If the quantity is greater than ten, **then** alert the user that they may only order up to ten at a time.

Now, here's how we'd write that in Deluge, using a relational operator:

```
if (quantity > 10)
{
        alert "You may only order up to 10 at a time.";
}
```

Quiz

Review this list of relational operators. Which conditions will be true, and which will be false? The answers are on the next page.

1 a = 1;
b = 2;
(b > a)

2 a = 6;
b = 6;
(a <= b)

3 a = "name";
b = "Name";
(a == b)

4 a = "name";
b = "Name";
(a != b)

5 a = "one";
b = "two";
(a < b)

6 start_date = '03-Dec-2017';
end_date = '01-Jan-2018';
(start_date < end_date)

Quiz Answers

1 True.

2 True.

3 False. Text data, or string variables, are case sensitive. Two strings have to be written exactly the same way for Creator to consider them equal.

4 True. Since these strings have different casing, Creator doesn't consider them equivalent.

5 Trick question! Since these variable values have quotes around them, they are string variables. You can't compare if a piece of text is greater or less than another piece of text. If you tried doing this, Creator would give you an error message.

6 True. The format and the single quote marks indicate that these variables are dates, not strings. When you're comparing dates, operators can tell you whether the date on the left is before, after, or at the same time as the one on the right.

Logical/ Conditional Operators

Logical operators let you combine multiple conditions to make more sophisticated conditionals.

For example, you can check that multiple conditions have been met before running a script.

`&&` both conditions are true
`||` at least one condition is true

Conditional Examples

Suppose you're building an app that calculates grades. You need to define percentages for each letter grade. After defining an A as 90 and above, you need to define a B.

Here's how you'd write that in plain English:

If the score is greater than or equal to 80 **and** less than 90, **then** the letter grade is a B.

Here's how a conditional statement using logical operators would look in Deluge:

```
if (score >= 80 && score < 90)
{
        grade = “B”;
}
```

Conditional Examples

Let's assume you use the same logic to define each of the letter grades. Say you also need to produce a status that says whether or not a student passed a class. For this you'll need to define what letter grades constitute a passing grade.

Here's how you'd write that in plain English:

If the grade is an A, **or** a B, **or** a C, **then** set the status to "Passed the class."

Here's how you'd write this conditional statement using logical operators in Deluge:

```
if (grade = “A” || grade = “B” || grade = “C”)
{
        status = “Passed the class.”;
}
```

Quiz

Take the following statements, and convert them into Deluge conditionals.
The answers will be on the next page.

1 If the item costs $50 or more, then set the shipping cost to free.

2 If today is Saturday or Sunday, then set my status to unavailable.
Otherwise, set my status to available.

3 If it's before noon, set the message to "Good morning." If it's any time from twelve to six in the afternoon, set the message to "Good afternoon."
Otherwise, set the message to "Good evening."

Quiz

Quiz Answers

1

```
if (item_cost >= 50.00)
{
        shipping_cost = “free”;
}
```

2

```
if (today == Saturday || today == Sunday)
{
        status = “unavailable”;
}
else
{
        status = “available”;
}
```

3

```
if (hour < 12)
{
        message = “Good morning”;
}
else if (hour >= 12 || hour <= 18)
{
        message = “Good afternoon”;
}
else
{
        message = “Good evening”;
}
```

Field Data

So far, we've been working with data stored in variables. But when you're writing scripts for your application, you'll probably want to use data that people are entering in your fields as well. To do this, you need to know a field's **link name**.

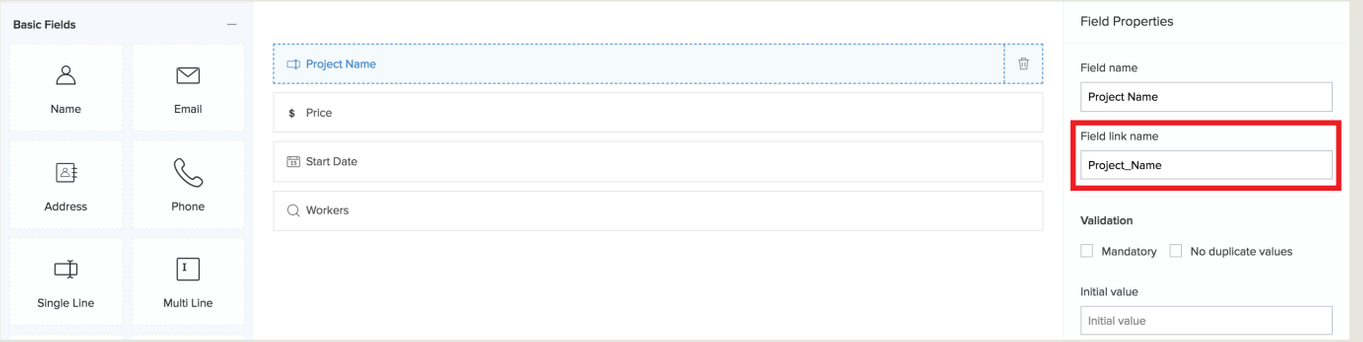
Each field has two names: the human-friendly one that you give it, and the computer-friendly **link name** that Creator gives it. Just like variable names, **Link Names** can't contain spaces or begin with numbers.

Field Data

Finding Field Link Names

There are two ways to find field link names.

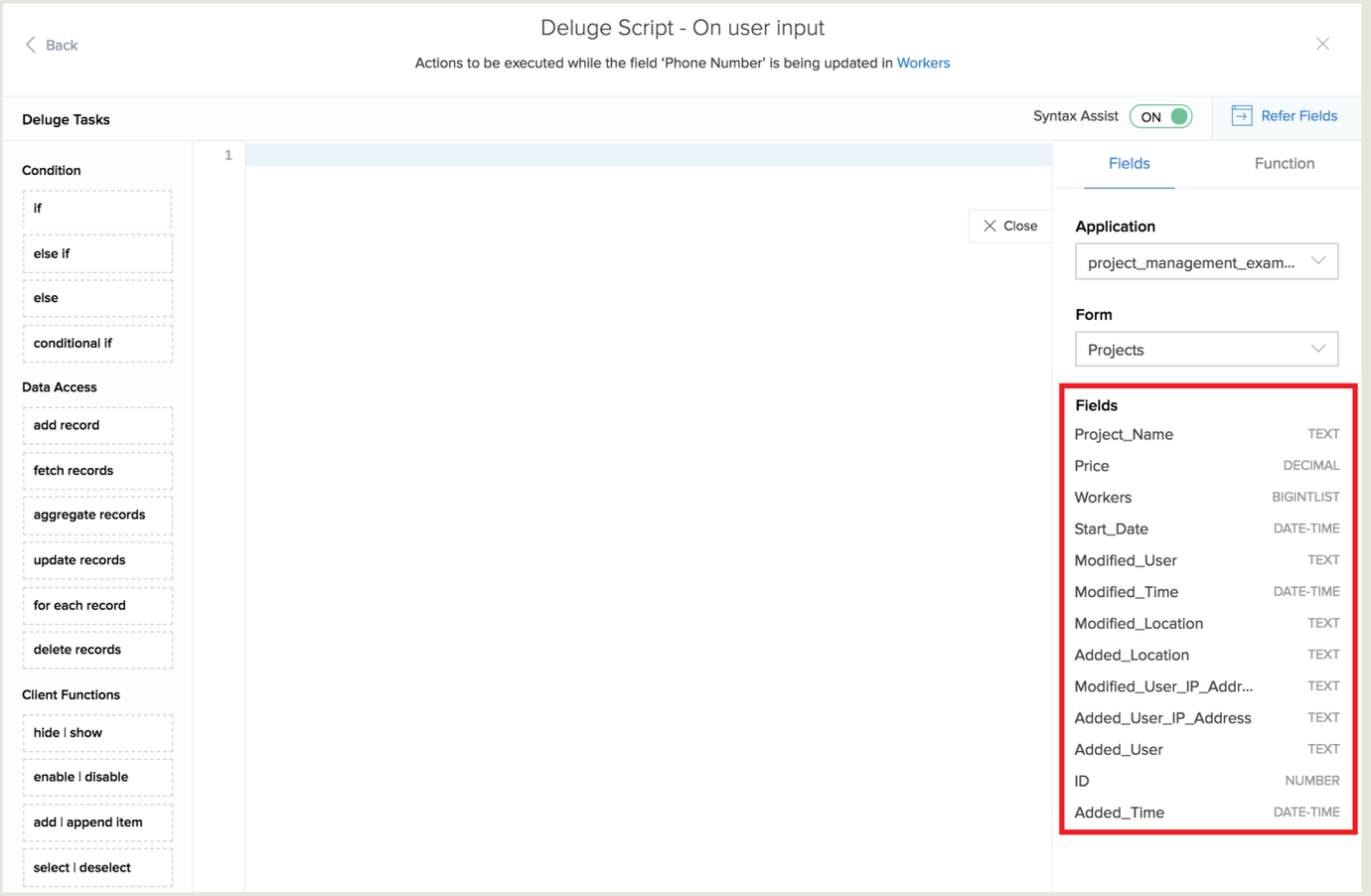
1 In the Form Builder.



When you click on a field in the **Form Builder**, its properties will display on the right of the screen. At the top of the field properties, you'll see the field name and field link name listed. When you change a field's name, your script won't look any different. When you change a field's link name, any scripts that reference that field will be automatically updated with the new link name.

Field Data

2 In the Refer Fields Section.



The field link name can be referred in the script editor when you write the script. You can view each field's link name from the **Refer fields** section on the right. It will display the field link names of all the fields in your form. To refer field link name from another application/form, click the **Application** dropdown. It will display all the applications on your account. Select the required application and the **Form** dropdown will display all the forms in the selected application. Choose a form and it will display the **field link names** of all the fields in the form.

Accessing Field Data with Deluge

Once you know a field's link name, you can use the syntax below to access its data in Deluge:

```
input.<fieldlinkname>
```

Say you have a contact form on your website where people type their name. phone number, and message. Your boss would like to personalize this experience by using the submitter's name while thanking them for submitting the form.

Here's how you'd write that in plain English:

If someone adds data to the first name field, **then** use their name in the thank you message.

Here's how you'd write this message while accessing field data in Deluge:

```
"Thanks " + input.First_Name + ", we'll be contacting you soon.")
```

Remember to mind the spacing in your text strings. We inserted one after "thanks" above.

Putting it All Together

Now that we've gone over the foundation of Deluge, let's try writing a script that puts it all to use. Let's quickly review what we've learned:

- What variables are
- How to declare variables and determine their data type
- How to manipulate data with operators
- How to add logic with conditionals
- How to access data entered in fields

We'll be using all of these tools and techniques to write a script that discounts the price of pizza orders. Discounts will be based on the price of the order and from which location the order was submitted.

Let's list the goals for this script as clearly as possible:

- Calculate total price
- Provide discount for specific location
- Provide discount if the total order amount is greater than $100

Putting it all Together

Calculate total price

To get the price total of a pizza order, we'll need to multiply the price of a particular pizza by the number ordered. We'll start writing our script by declaring our total price variable.

```
Total_Price = Price * Quantity;
```

The line of code above simply defines what a total price is. The new variable **Total_Price** holds the product of the **Price** and **Quantity** parameters.

Provide discount for specific location

To provide a discount for a specific location, we'll need to start by declaring a variable. Since a discount percentage is always a number, we'll need to declare the discount variable as an *integer*, or *bigint*, data type, and then set the condition for the discount.

```
Discount_Percent = 0;
if (Location == “San Francisco”)
{
        Discount_Percent = 10;
}
```

The variable **Discount_Price** is set to the value 0 by default. Then the conditional states that if the user's **Location** is in "San Francisco" then the variable **Discount_Percent** is set to a value of 10.

Putting it all Together

Provide discount if the total order amount is greater than $100

This script will be similar to the one on the previous page. We'll need to make sure we declare our variable, and then create a conditional.

```
Discount_Percent = 0;
if (Total_Price > 100)
{
        Discount_Percent = 10;
}
```

All Together

Now, let's put all the code we've just looked at together.

```
Total_Price = Price * Quantity;
Discount_Percent = 0;
if (Location == “San Francisco” || Total_Price > 100)
{
        Discount_Percent = 10;
}
Dis_Price = Total_Price * Discount_Percent / 100;
Total_Price = Total_Price - Dis_Price;
```

The above lines of code establish how to calculate the total price and set the default discount to 10. You then have the *if statement* that states that if it is true that the location is "San Francisco" or it is true that the **Total_Price** is greater than 100, then a discount is applied. **Dis_Price** is declared as being the **Total_Price** multiplied by the **Discount_Percent** (which is either 0 or 10, depending on the results of the *if statement*), all divided by 100. The **Total_Price** variable is then re-declared as being the former total, minus the newly declared **Dis_Price** variable.

And there you have it! With that, you can now edit and manipulate your own custom applications using Deluge.

Where Does my Script Go?

Now that you've learned the basics of writing Deluge, you'll probably want to start applying it to your applications. The first thing to decide when you begin scripting is when your script should run. Depending on how people use your application, you'll want to run different scripts at different times.

For example, if you had an application to collect patient information at a doctor's office, you could run a script every time a patient checks a box saying that they have allergies. Your script could show these patients a series of follow up questions. You could write a separate script that runs every time a patient submits the check-in form that sends an SMS message to the patient's doctor.

To write a Deluge script, click **Workflows** on the header while you are editing the application. You'll have to decide whether your script should be triggered **while using the form, on a specific date and time**, during **payment process**, during **approval process**, or work as a **Function**.

Form Workflows

If the action/script that you have defined should be executed when people are filling out forms or altering records in a report, then it belongs to Form workflows. You can run the workflow in any of the following record events.

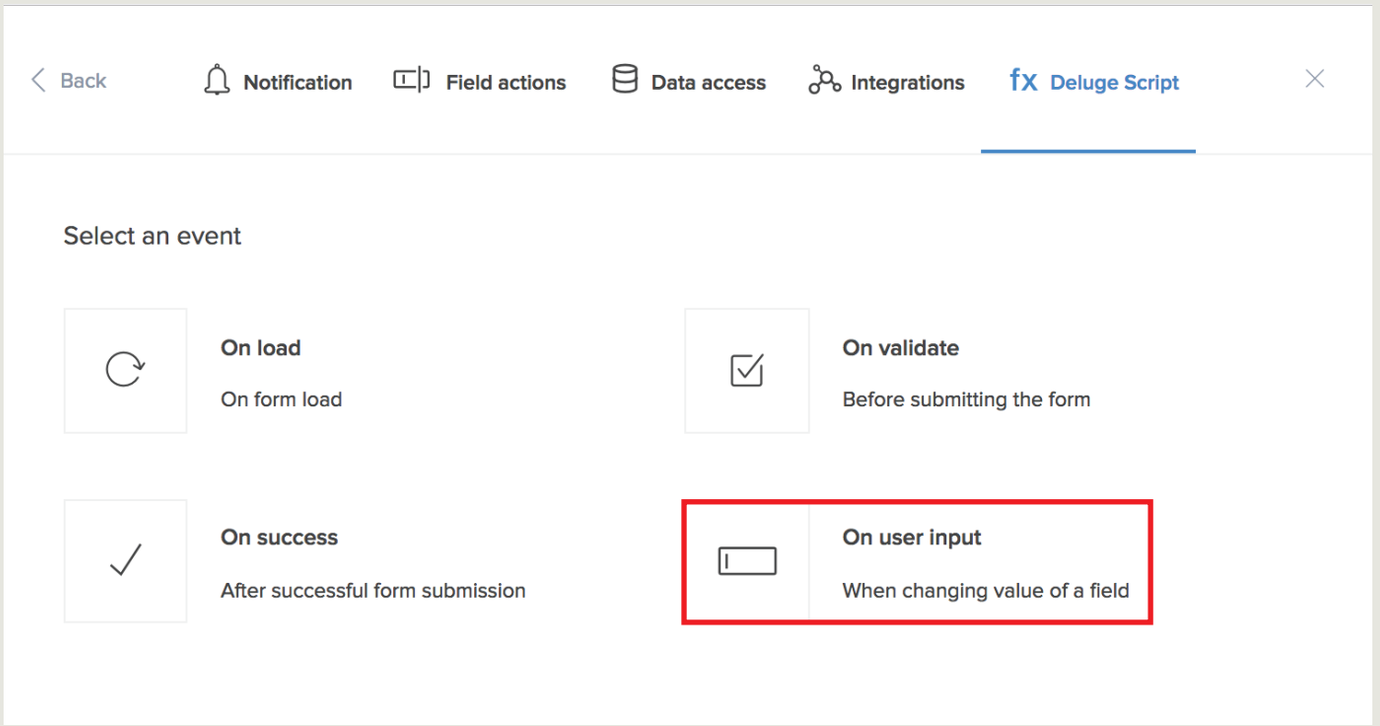
- **Created** – This record activity will trigger the actions/scripts when a new record is submitted.
- **Created or Edited** – This record activity will trigger the actions/scripts when a new record is submitted or an existing record is modified.
- **Edited** – This record activity will trigger the actions/scripts when an existing record is modified.
- **Deleted** – This record activity will trigger the actions/scripts when a record is deleted.

After you've chosen the record event to run the workflow, you must choose the form activity at which it should run.

- **On Load** – Write scripts here to alter how your form is set up before the user even sees it. On Load scripts are often used to hide fields that will only be relevant to a few users, or to pre-populate information in the fields when the form loads.
- **On Validate** – Write scripts here to verify that users have entered the right information or prevent them from deleting important information. These scripts will run when a user tries submitting, updating, or deleting a record, but before any permanent changes are made.
- **On Success** – Write scripts here to run after a record event is completed. For example, you could place scripts in this section that alerts people once a record has been created, updated, or deleted

Field Actions

You can also execute workflows to run after people fill out a particular field in a form, or while updating a record.



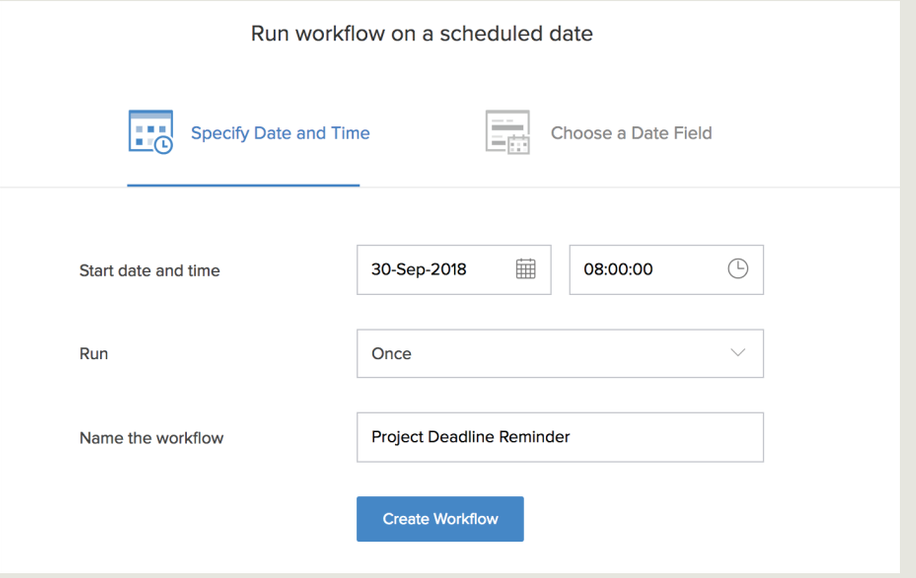
• **On User Input** – Write scripts here to run after user fills out a field in a form.

> If you run an On User Input script on a field users type in, like a single line or a currency field, then it won't run until their cursor leaves the box. This means your script won't run until they click somewhere outside the field they just typed in.

• **On Update** – Write scripts here to run after user fills out a field while updating records in a report.

Schedules

Zoho Creator lets you schedule when your actions/scripts should run. For example, if you have a form where users choose an appointment time, you can schedule a script that sends them a reminder email one hour before their appointment begins. Another idea for using schedules is to write a script that erases outdated records once a month.



To run a script on a schedule, click **Workflow** on the header while you're editing the application. Click **New workflow** on the workflow dashboard, **choose on a scheduled date.**

- **Specify date and time** – You can schedule actions/scripts to run on a specific date and time and set the frequency of the schedule to daily, weekly, monthly or yearly.

- **Choose a date field** – You can schedule actions/scripts to run using the time saved in a date field, the time a record is added to your application, or the time a record is modified. After you choose which time to base your schedule on, you may choose how long before or after that time your script should run.

Functions

If your script should run in several applications, several parts of one application, or when users click a button in a report, you should write the script as a **function**. Functions differ from Form and Field Actions because they're not constrained to one part of your application. Once you've written a function, you can apply it to any Form or Field Action by writing just one line of Deluge. This is called “invoking” or “calling” a function.

Since functions can be created without affecting your forms and fields, they're useful for adding features to applications that your organization is already using. Functions make it easier to find bugs in your scripts because you don't have to switch to the live mode of your application to test them. Just click **execute** when you finish writing a function script, and a pop-up with the results of your script will appear.

When you start creating a function, you have to fill out a pop-up box with some technical information. Keep in mind, function names can’t include spaces. Here's a quick overview of what all the stuff in that pop-up box means.

Functions

• **Function Name** – As with variables, you can choose any name but it helps to name your function something that will remind you of what it does. When you're done writing the function and you want to use it in your application, you'll be using this name to call it and run the script inside.

• **Namespace** – You can leave this field blank. A namespace is like a folder. You won't need to use it unless you have lots of functions that you want to organize by category.

• **Return type** – This is where you choose the data type that your function will output or return. Click here to jump to the chapter on basic data types for descriptions of each one.

• **Arguments** – Argument is just programming jargon for input. Since functions aren't tied to particular parts of your application, getting data from fields into your function's script requires arguments. Without arguments, Creator wouldn't know which data your function should work with.

Here's an example of an argument. You need a function that can take any price entered in a currency field and calculate the sales tax. Before you can begin making calculations, you'd need to add an argument that passes the price from the currency field into your function. Otherwise, Creator wouldn't know which number to use while calculating the sales tax.

In programming, this process is often called “passing an argument” because you're passing information from a field into a function. Just as each field in Creator has a Deluge name and a data type, each argument has a name and a data type that you'll have to fill out when you create a function.

Creating a Function

As you can see, there are a lot of places where you can run Deluge scripts. Let's look at an example of how you'd write a script that you could reuse several times.

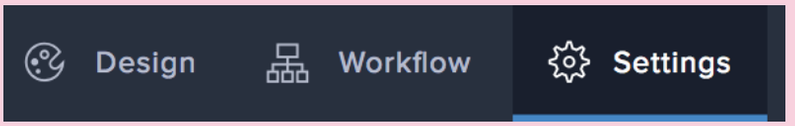
The Requirements
Jack has an application for managing his schedule and keeping track of which hours his employees work. His application has one form where employees can sign up for appointments with him and another form where employees can request time off.

Since Jack needs time to reschedule shifts when employees take days off, he wants a script that will make sure requests are submitted at least two business days in advance. He wants to use the same script to ensure that employees schedule appointments in advance so he can clear room in his schedule to meet with them.

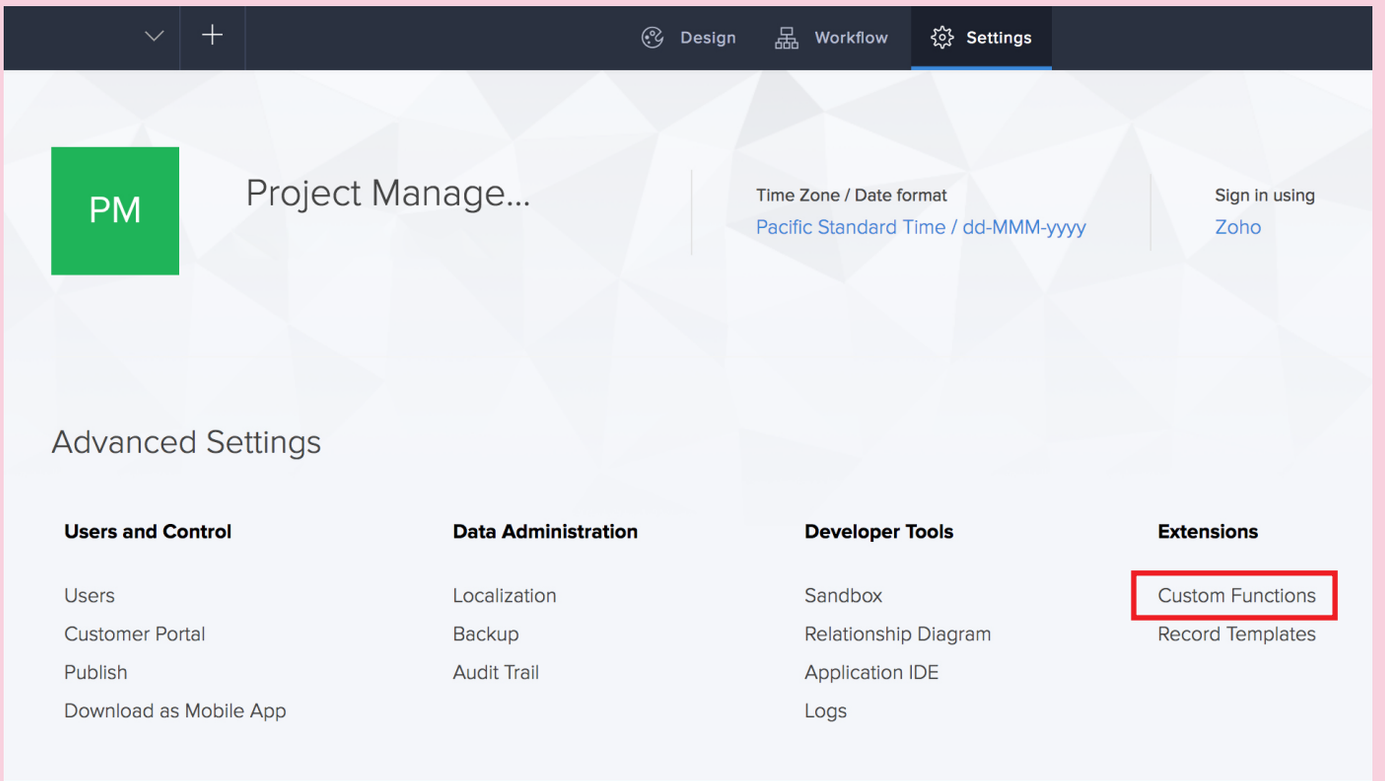
Creating a Function

Creating the Function

This script works best when it's written as a function since it needs to be used in more than one place. To create this function, click **Settings** on the header tab.

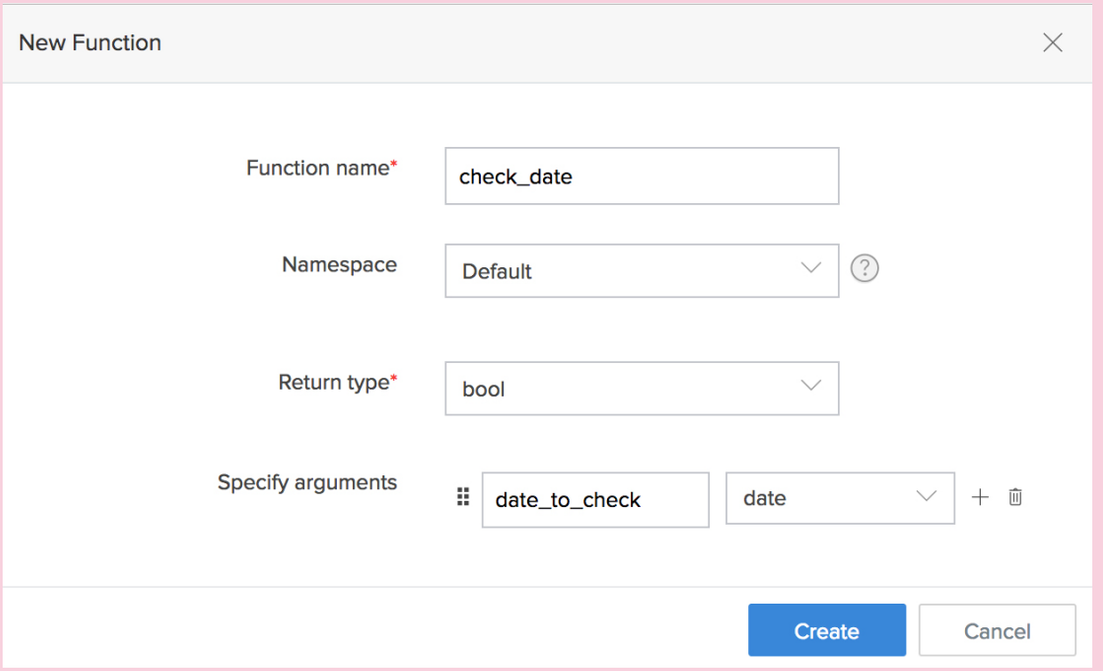


Click **Custom Functions** under **Extensions**.



Before you can begin writing a function, you have to fill out a pop-up with some information about it. A **Function Name** is like a field link name because it must be unique and can't contain any spaces. The **Namespace** field is only important when you have lots of functions that need organizing.

Creating a Function



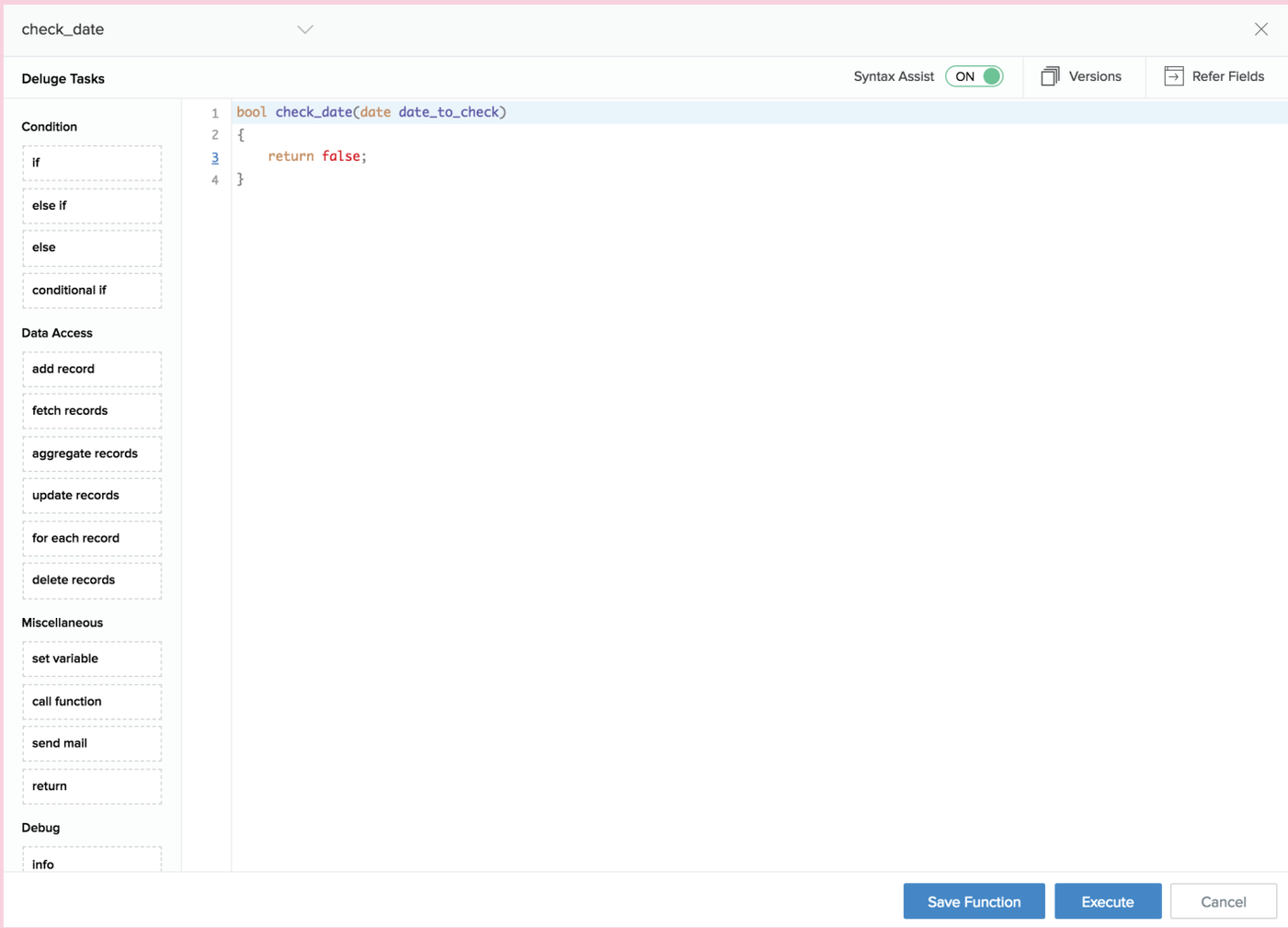
The **Return Type** is the kind of data that you want your function to give you when it's done running. This function should return Boolean, or **bool** data, because there are only two possible outcomes. Boolean variables can only have two values (true and false), which correspond to only two outcomes. If the function determines that a date is too soon, it returns the Boolean **false**. If the function determines that a date is acceptable, it returns the Boolean **true**.

A return type is an output, but an **argument** is an input. Arguments tell Creator what kind of data to bring into your function. Functions aren't tied to particular parts of your application, so getting data from a field into a function requires an **argument**. Each argument has a name and a data type.

This function works with date data, so it needs an argument with the date data type.

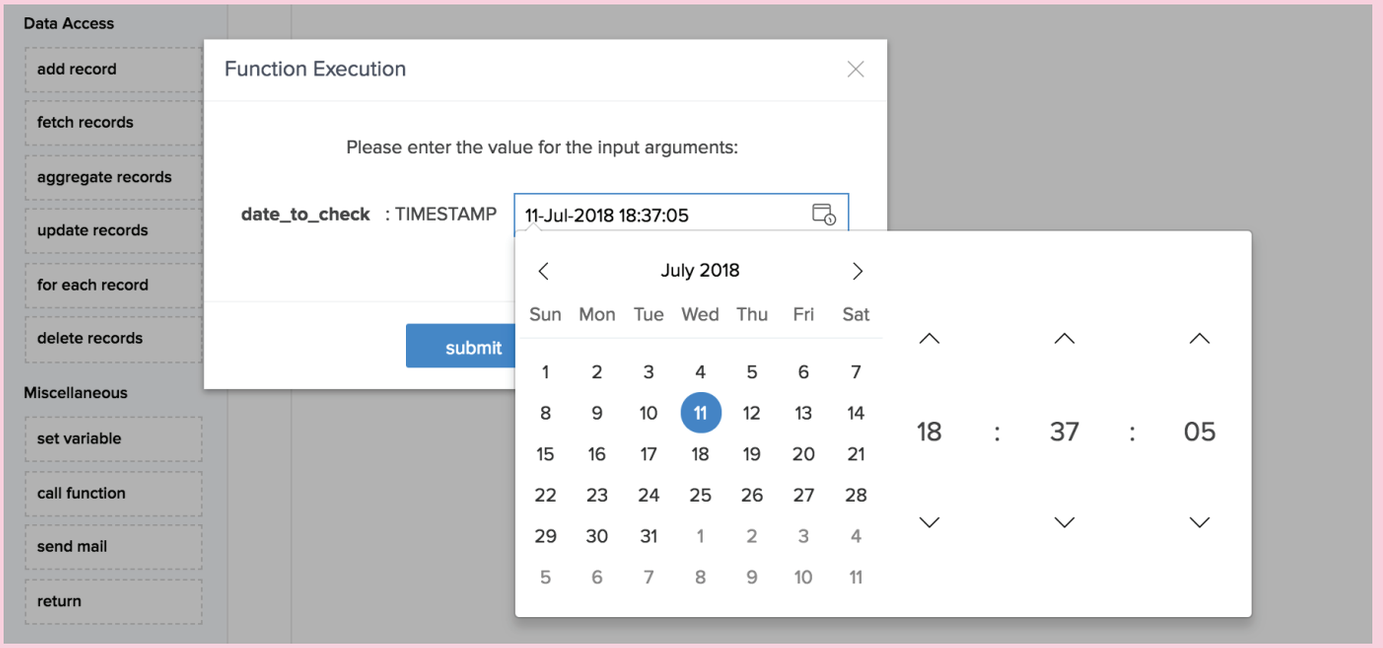
Creating a Function

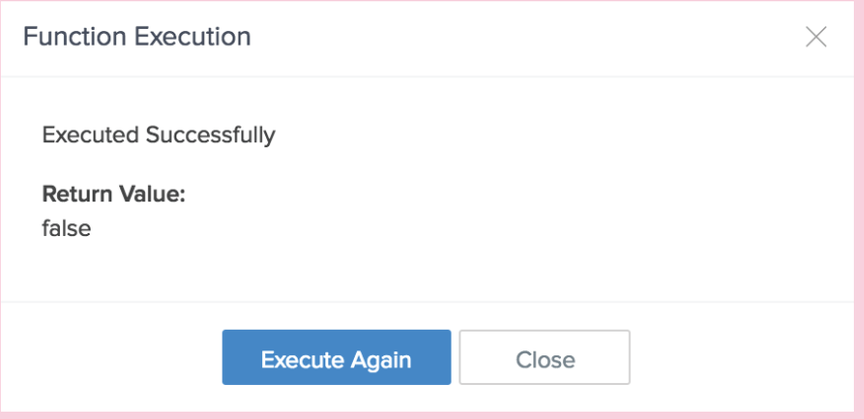
After clicking **Create Function**, you'll see a screen which already has some Deluge script entered in it. The script on the first line outside the parentheses says what kind of data the function returns and what its name is. The script inside the parentheses lists all the arguments and their data types. Every function except for *void functions* also has a line saying what the function returns.



Creating a Function

Try saving your script and clicking **Execute**. Creator will give you a pop-up box to enter a date for your argument. Right now, when you click **Submit** Creator will say the return value is **false** every time because your script tells it to always return false. When we're done, the return value will only be false if you choose a time that's not acceptable.

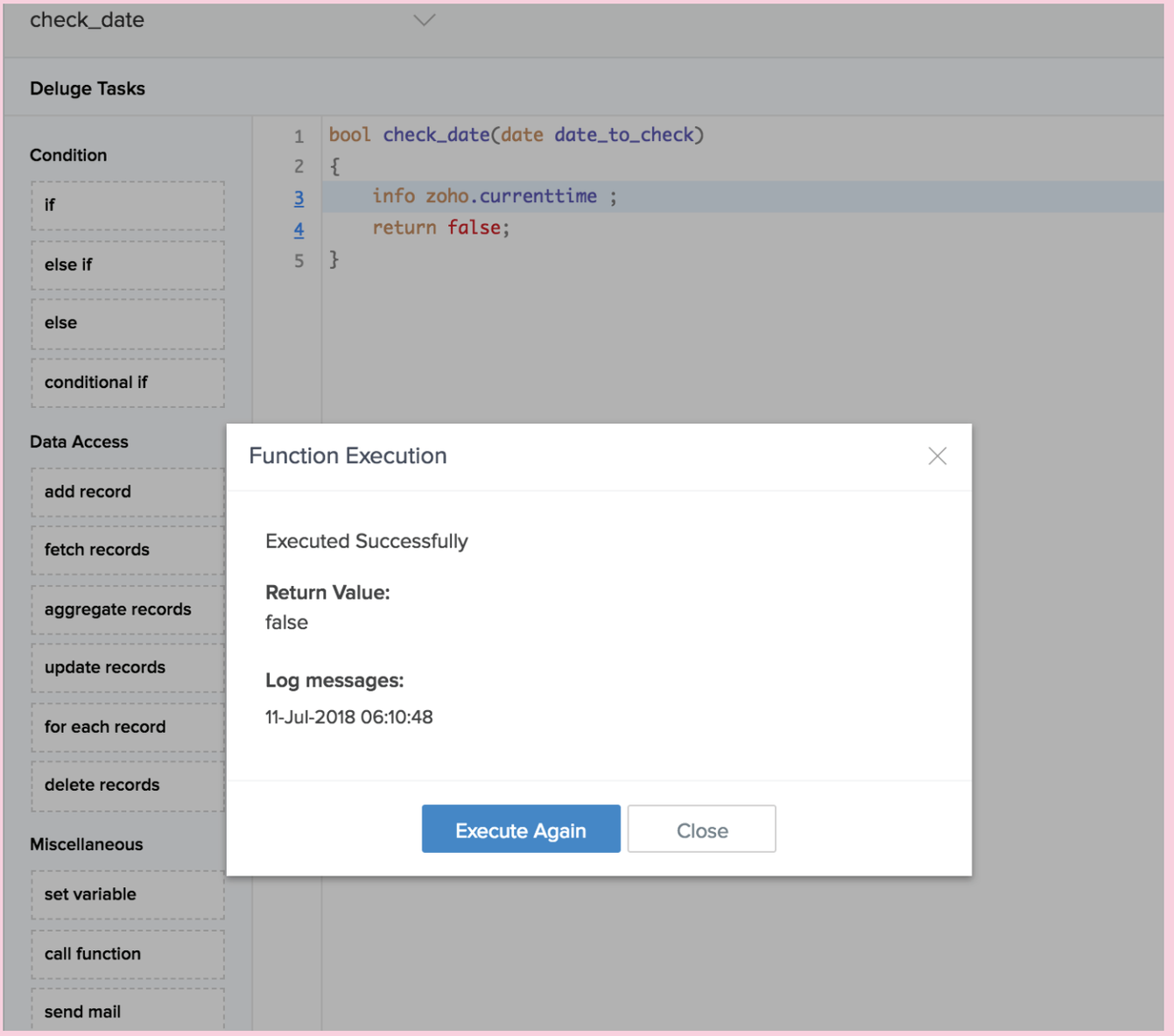




Creating a Function

The first step to knowing if the date entered is acceptable is knowing the current date. You can get the current date by using the system variable **zoho.currenttime**. System variables are special because they contain values that you didn't assign them, like the current time.

Write **info zoho.currenttime;** in your function. Save your script, and then click **Execute**. You should see something like this.



Creating a Function

In addition to the return value, you should now see a section called "Log messages" that has the current time in it. Log messages appear when you use an info statement in your script. They're useful whenever you want to check the value of a variable to see if your function is working correctly.

Now that we know the current time, we can figure out what time it will be two business days from now. We'll use one of Deluge's **built-in functions**.

Each type of data has its own built-in functions that allow you to manipulate it. For example, string (text) data has a function that lets you find every instance of a word and replace it with another word. Since we're manipulating information from a Date-Time field, we'll look at the list of **Date and Time functions** in the Creator User Guide. The addBusinessDay function lets you add business days onto a date. The Creator User Guide displays the syntax for this function as:

```
<start_date>.addBusinessDay(<days>)
```

When you look at syntax in the Creator User Guide, the parts inside the carrot symbols need to be replaced with your own variables or values. Usually, the Creator User Guide will tell you which data types are compatible with each part of the function. The page about the addBusinessDay function says that <start_date> is supposed to be a date variable, so you should write zoho.currenttime (which is a date variable) in this part of your script.

<days> indicates how many business days you want to add onto your start date. <days> is supposed to be an integer (or big int) variable. We could declare a variable and use that here, like this:

```
daystoadd = 2;
info zoho.currenttime.addBusinessDay(daystoadd);
```

Creating a Function

However, that's adding an unneccessary step. Using a variable is useful when you have a value that can change. In our case, we only want this function to add two business days; the value never changes. So we can just write 2 where it says <days>.

```
info zoho.currenttime.addBusinessDay(2);
```

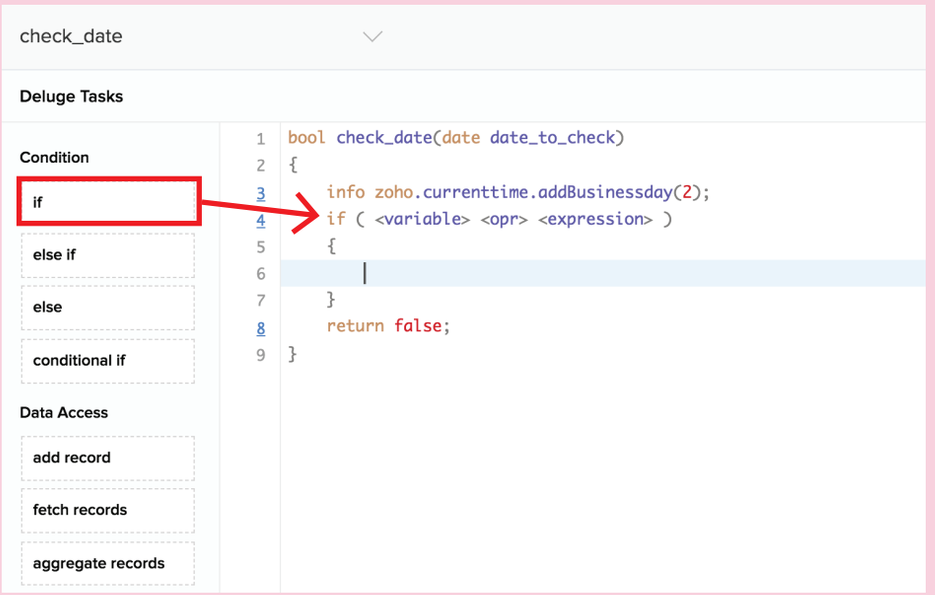
Try testing out the addBusinessDay function and use *info* to see the results. The final step in the function is to compare the date scheduled for an appointment or time off with the date we've calculated to be two business days from now. To do this, we'll need to use a couple of conditionals, or *if statements*.

Before we write this script in Deluge, let's think about how we'd phrase these instructions in English. Remember that our function is set up to **return a boolean**, a true or false. When the scheduled date is acceptable, our function should return true, indicating that the application should accept it. When the scheduled date is too soon, our function should return **false**, indicating that the application should **not** accept it. So if we wrote our *if statements* in English, they'd look like this:

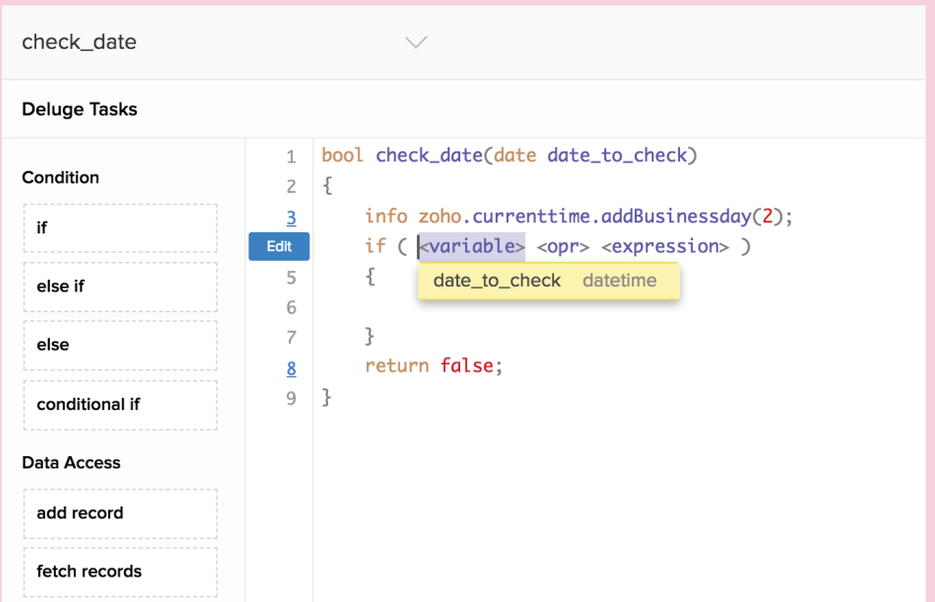
If the scheduled date is less than two business days from the present, **then** return false.
If the scheduled date is at least two business days from the present, **then** return true.

Creating a Function

To start writing your *if statements*, look in the Condition Deluge Tasks and drag an **If** block onto the script builder before the line that says **return false;**

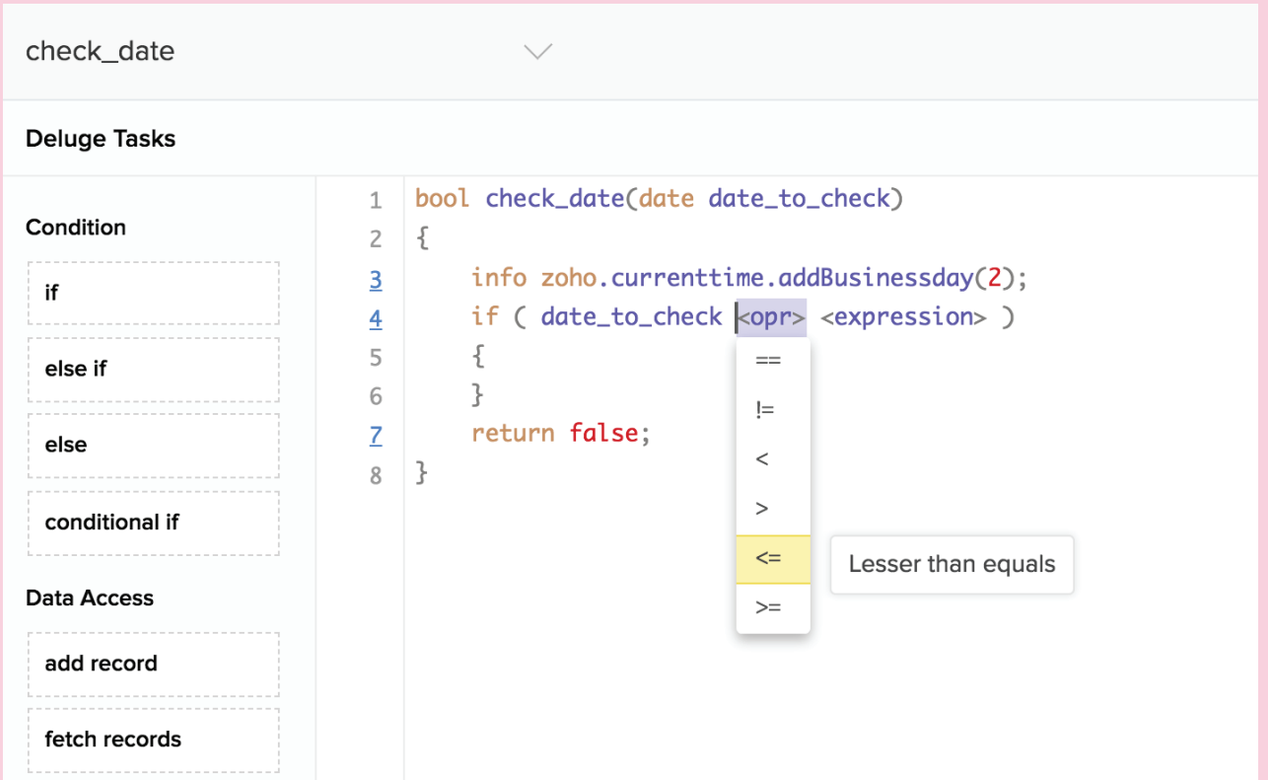


When you click on the word **<variable>** you'll see a dropdown with all the possible variables you could write there. You should see the name of your argument in this dropdown since that's the only variable that's been declared for this function. Click on the argument to add it into your script.



Creating a Function

Next, click on **<opr>**. You'll see a dropdown with all the operators you can use in your condition. Remember that operators have different meanings for date variables than they do for other variables. In the context of numbers, **<=** means **less than or equal to**. Since we're dealing with dates, we'll choose this operator because it also means **before or at the same time**.



Where it says <expression> you should paste the script you wrote earlier that adds two business days to the present. Inside the parentheses, you should see this script:

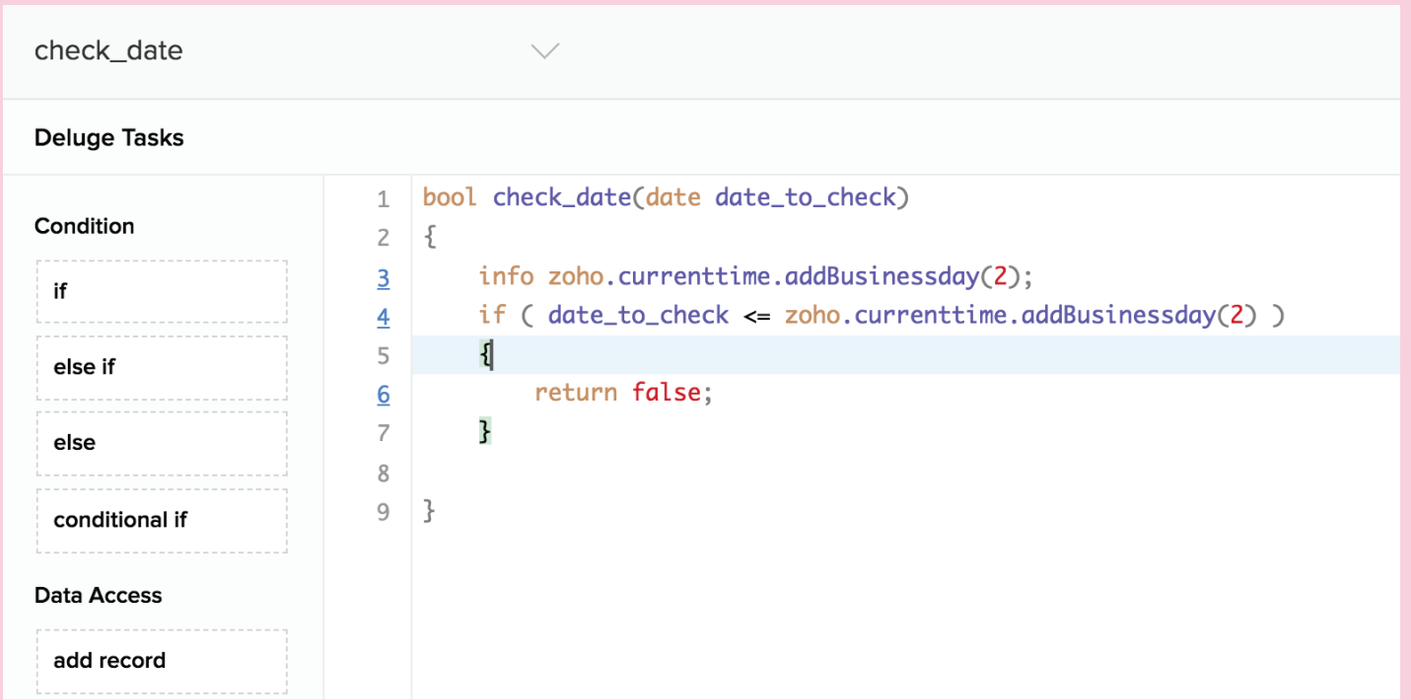
```
date_to_check <= zoho.currenttime.addBusinessDay(2)
```

Since this is a condition and not a statement, you don't need to keep the semicolon at the end of your expression.

Creating a Function

Now that you've finished the condition, you need to tell your function what to do when that condition is met. Your function will execute any script that's written inside the conditional's braces { } if its condition is met.

Cut **return false;** and paste it in between the braces.

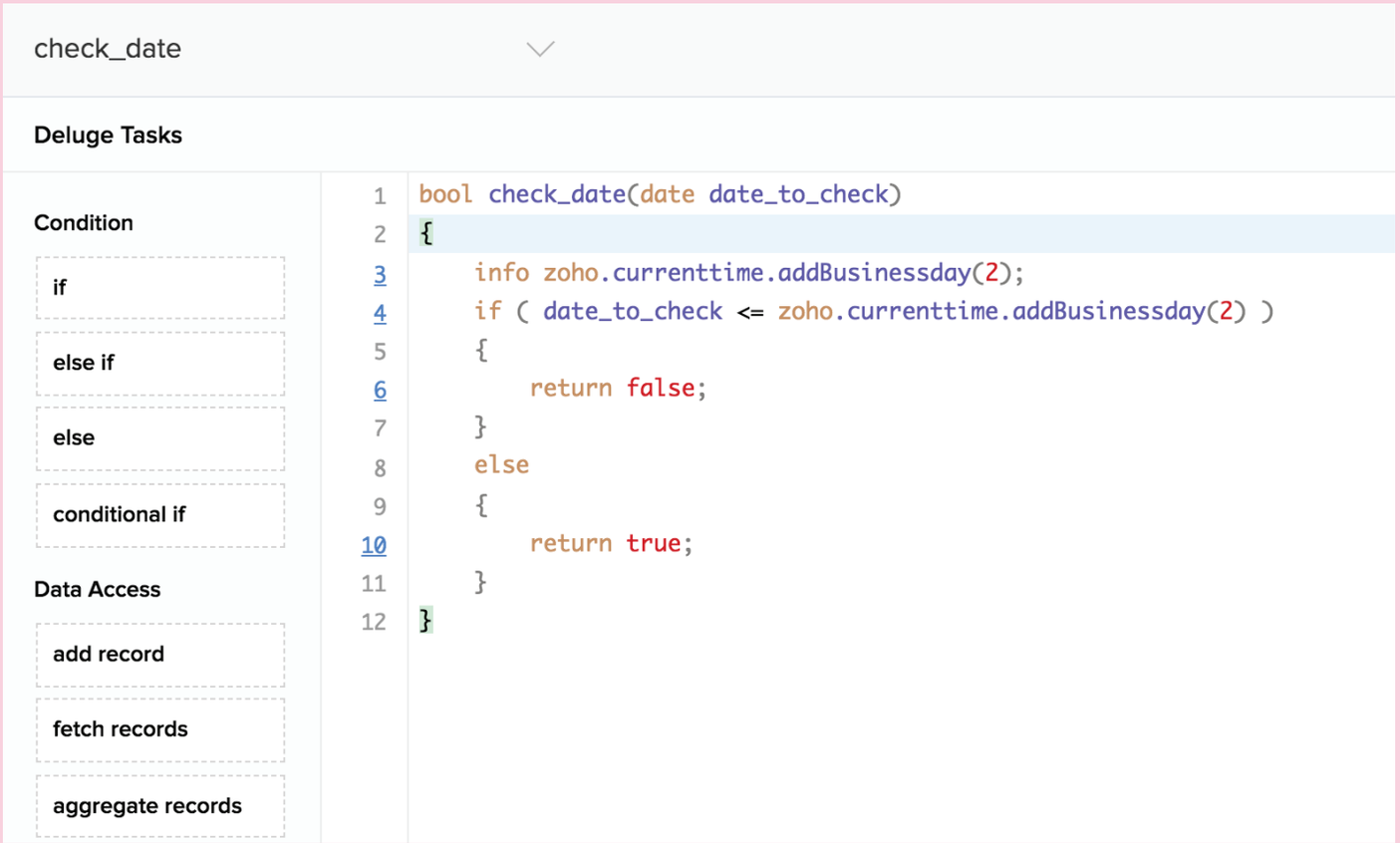


If you tried saving your function right now, Creator would give you an error message because you've told it what to return inside a conditional. If the condition isn't met, then your function won't return any value, which could cause problems.

To solve this problem, you need to put another return statement inside an **else** conditional. By adding a return statement inside an else condition, you ensure that your function will know which value to return.

Creating a Function

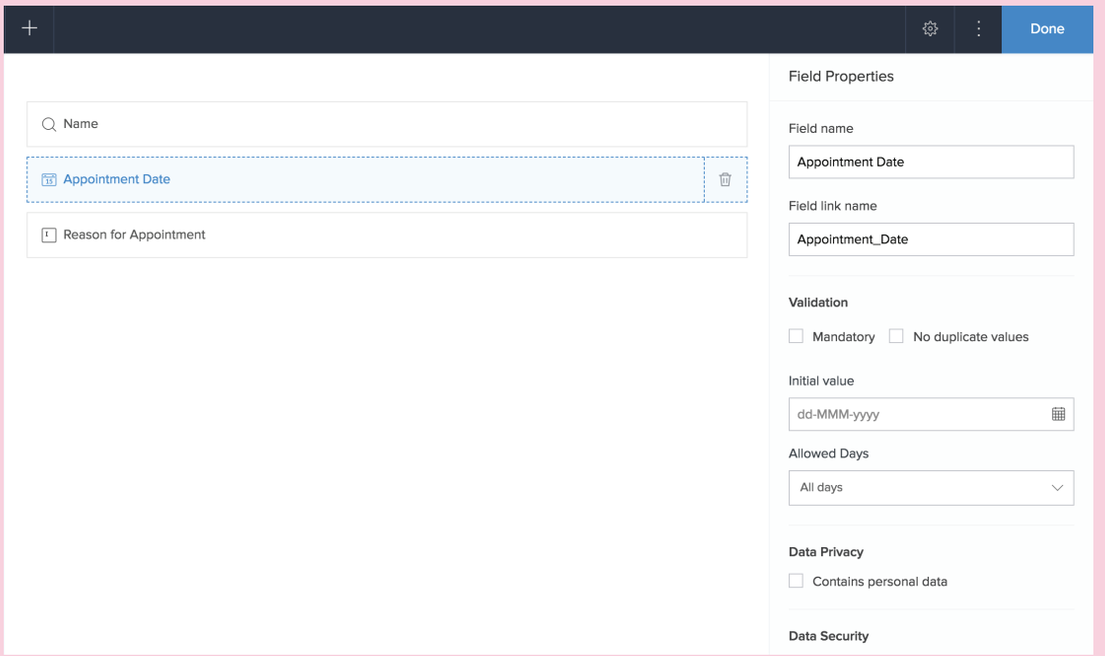
In the Condition section of the Deluge Tasks, drag on an **else** block. Write **return true;** inside the braces, then save your script. Now your function will return **true** only when a date doesn't match your if condition.



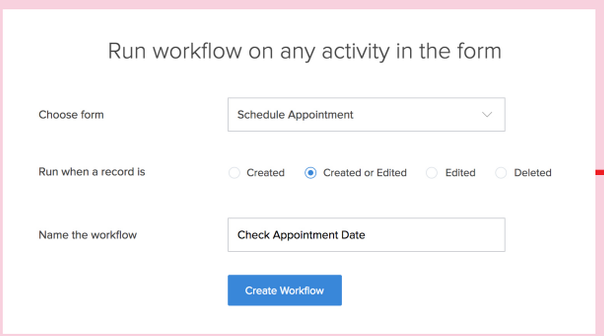
To use this function, you have to write a script that tells your application when it should run and how to use the information that it returns. In our example, Jack wants this function to run any time someone fills out a form requesting time off or any time someone fills out a form requesting an appointment. For the sake of brevity, we'll show you how this function would be implemented in one of those forms. The steps for implementing it elsewhere would be almost identical.

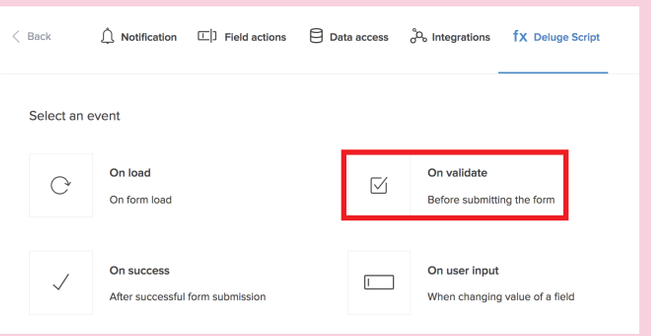
Creating a Function

Below is the form that users see when they try scheduling an appointment. Notice that the field link name for the appointment date is **Appointment_Date**.



We want to give users an error message if they pick an appointment that's too soon. To do this, we'll need to run our script in the **On Validate** section of the Deluge script when a record is **Created** or **Edited**. We put the script there because **On Load** would be too soon; a user wouldn't have chosen an appointment time yet. **On Success** would be too late because their appointments would already be scheduled in the application.

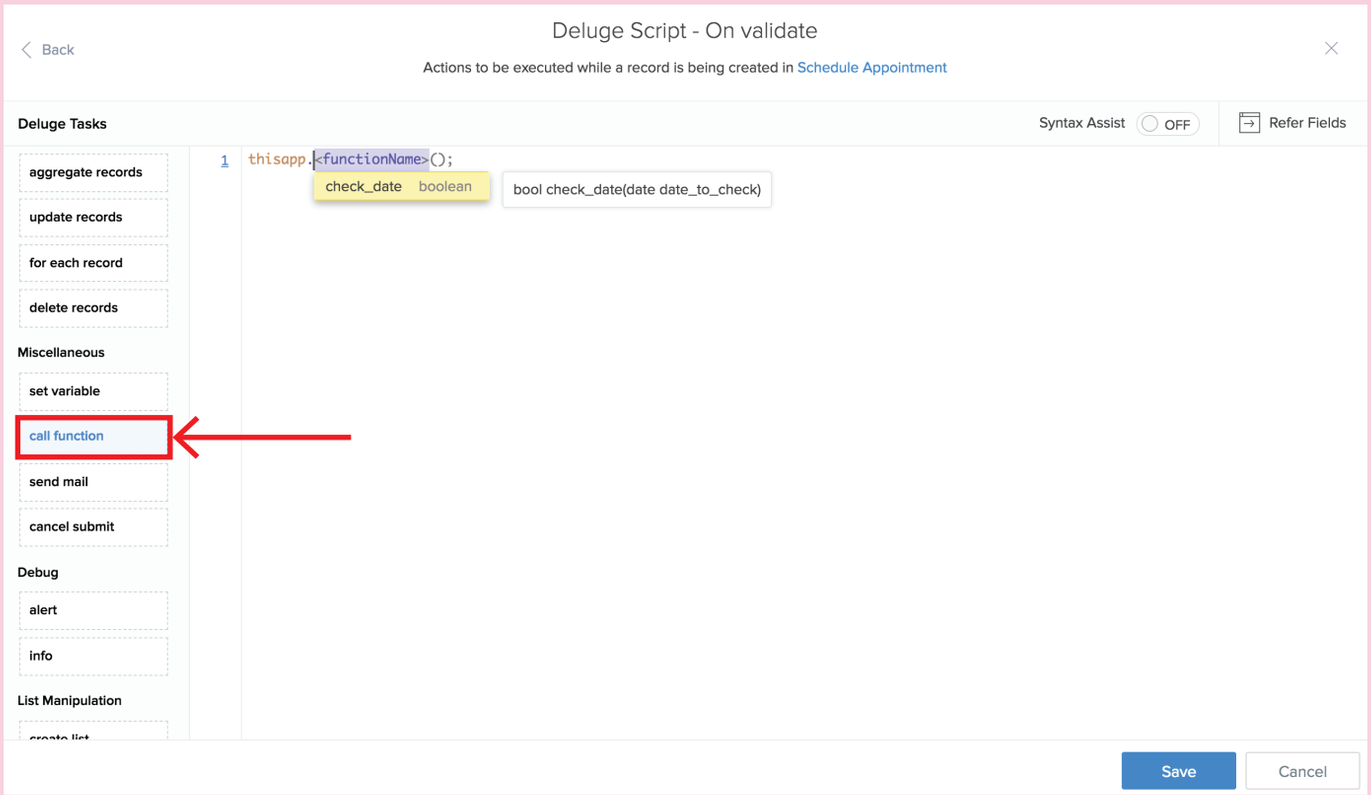




Creating a Function

When you want to run a function you've created, look in the Miscellaneous Deluge Tasks and drag the **Call Function** block onto the script builder. (Running a function is often referred to as calling or invoking because functions are stored offsite and only called up when they're needed.)

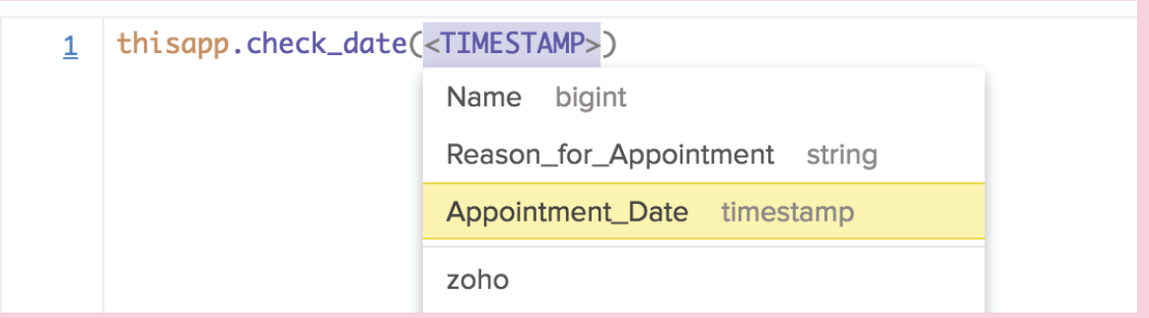
When you click on **<functionName>** you'll see a dropdown with all the functions you've made in this application. To the right of the function's name, you'll see the data type that it returns. Choose the function you just wrote.



Creating a Function

After you choose your function's name, you'll see some text appear inside the parentheses. This is where you choose which data you want to send to your function. When we set up our function, we created an **argument** so that it could accept the value chosen in a date field.

When you click on this text, you'll see a dropdown with all of the Field Deluge Names from your form. We'll choose **Appointment_Date** from the list of options because that's the name of the field where users are choosing a time for their appointment.



Now we have the script that calls our function, but we still need a way to use the information that it returns. If the function returns **false**, then we need to give our users an error message and prevent the form from being submitted. If it returns **true**, then we don't have to do anything and we can let our application save the new appointment to our schedule. We can accomplish this using a conditional.

Drag an **if** block onto the script builder and replace **<variable>** with the script for calling your function. Since this script can return two different values depending on the circumstances, you can think of it like a variable. Click **<opr>** and chose **==** and then click **<expression>** and replace it with the word **false**.

Creating a Function

```
1  if ( thisapp.check_date(Appointment_Date) == false )
2  {
3      
4  }
```

When your function returns false, it will run the script you put inside the braces. To give users an error message, look in the Debug Deluge Tasks and drag the **alert** block onto your script in between the braces. Click on **<expression>** and write your message inside quotes to indicate that it's a string, and shouldn't be treated as a variable. (Without the quotes, Creator would think each word you're writing is the name of a variable!) You'll notice that text inside quotes is displayed in red; this helps you differentiate between variable names and strings.

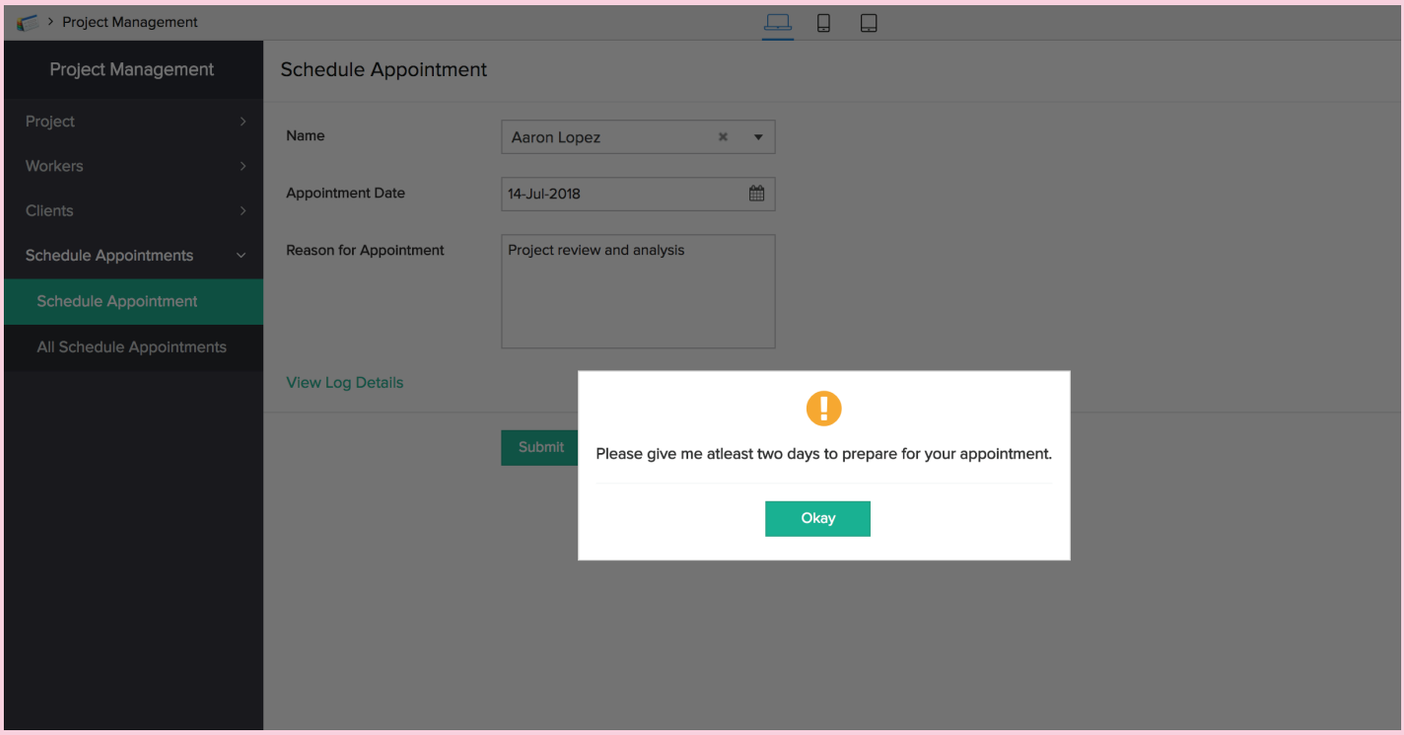
Next, look in the Miscellaneous Deluge Tasks and drag the **cancel submit** block onto your script, below the alert message. This script prevents the form from saving its data, and forces the user to pick a new date before they can submit their appointment.

Syntax Assist OFF

```
1  if ( thisapp.check_date(Appointment_Date) == false )
2  {
3      alert "Please give me atleast two days to prepare for your appointment." ;
4      cancel submit;
5  }
```

Save your script, and access the live mode of your application. Try submitting your form with a date that's only one business day away to see your error message in action.

Creating a Function



Congratulations, you're done!

Review

We covered a lot of material in this ebook. Let's look at some of the things you've learned:

- **Variables** are like containers for data. The information they store can change over time.

- **Data Types** are categories of variables. Depending on the data type, Creator will treat the information inside differently. Also each data type has its own set of built-in functions.

- **Operators** let you work with variables. They can do things like add numbers or combine bits text together. **Relational operators** let you compare variables to see things like which one is bigger, whether or not they're equivalent, or (in the case of date variables) which date comes first.

- **Conditionals** are like if-then statements. The part in parentheses checks for a condition, and the part in braces { } tells Creator what to do when that condition is met.

- **info** and **alert** are what programmers call "debug" statements. Sprinkle them into your scripts in different places to see how a variable's value is changing line by line.

Review

- The place where you put your script matters a lot. You can run scripts when records are created, edited, or deleted. You can run scripts before the page loads, after someone submits a form but before their record has been saved, or as soon as a record gets stored in your application. You can also run scripts when someone fills out or updates a particular field.

- **Functions** are scripts that you can reuse in several places. They're also easier to work with because they let you isolate code into smaller chunks that are easier to see, and they make it easy to debug your scripts.

- **Arguments** are the information you pass into your functions. Return types are the data types that your functions spit back out when they're done processing.

With these building blocks at your disposal you'll find you can start accomplishing most tasks you set your mind to. Once you've got the hang of the basics in this guide, we strongly encourage you to check out some of the built in functions in the help guide. With your knowledge of variables and data types, you should be able to get them working.

Thanks for reading this ebook. If you loved our work or felt like something was missing, please let us know your thoughts by emailing us at feedback@zohocreator.com

Contact Creator support:

USA : +1 (888) 900 9646
UK : +44 (20) 35647890
Australia : +61-2-80662898
India : +91-44-67447000

Email us at support@zohocreator.com