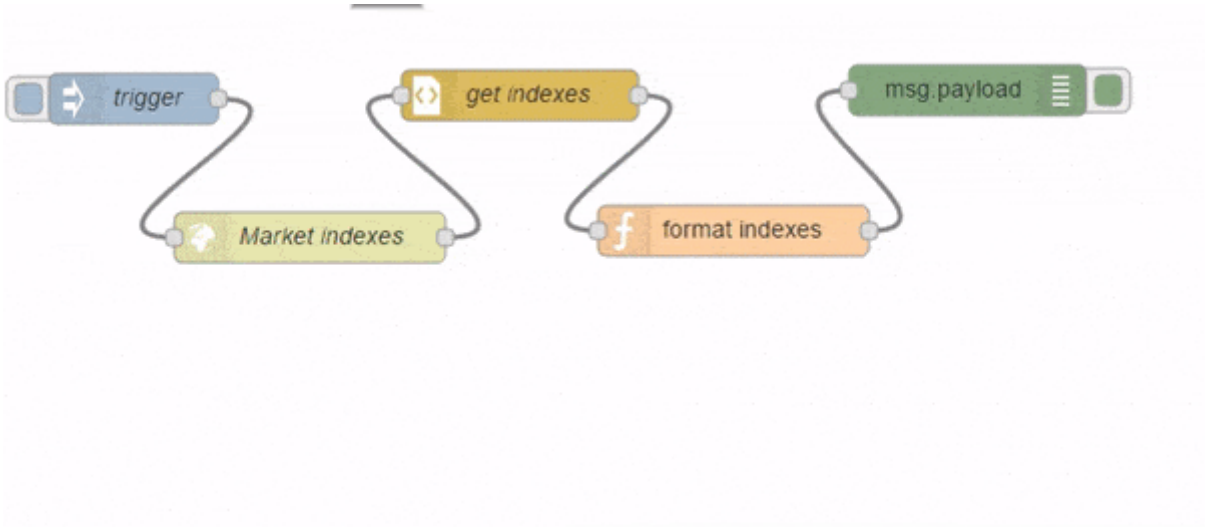
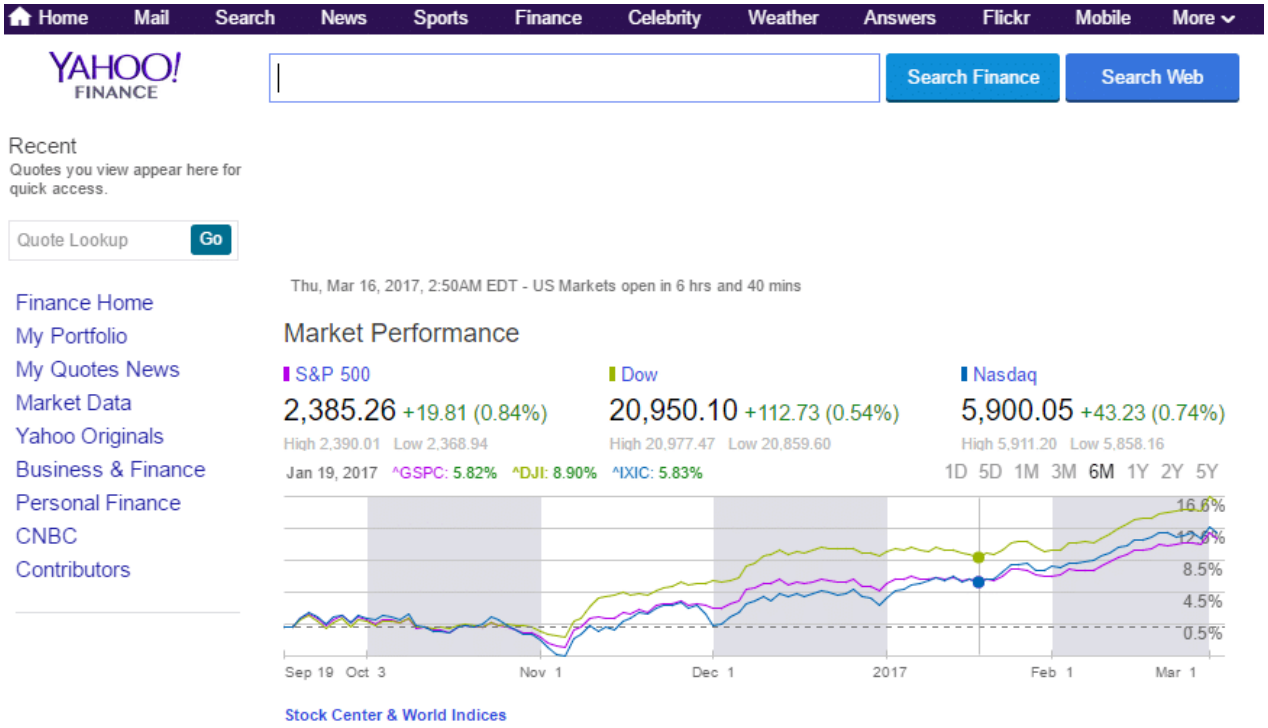
مشاريع Node-RED متوسطة التدفق

في هذا الدرس، سوف نبني على الأفكار التي تم عرضها في المحاضرة السابقة و التركيز على الأمثلة التي تكشف بعض المفاهيم الأساسية من الدرس السابق. و يشمل هذا الدرس مزيد من الأفكار حول السياق (Context) و الرسائل (Messages) و التدفقات الفرعية (sub-folws) . الأمثلة في هذا الدرس أكثر تعقيدا قليلا من الأمثلة السابقة– بمعنى أن عقد Function أكثر تعقيدا – ولكن لا تزال بسيطة قدر الإمكان.

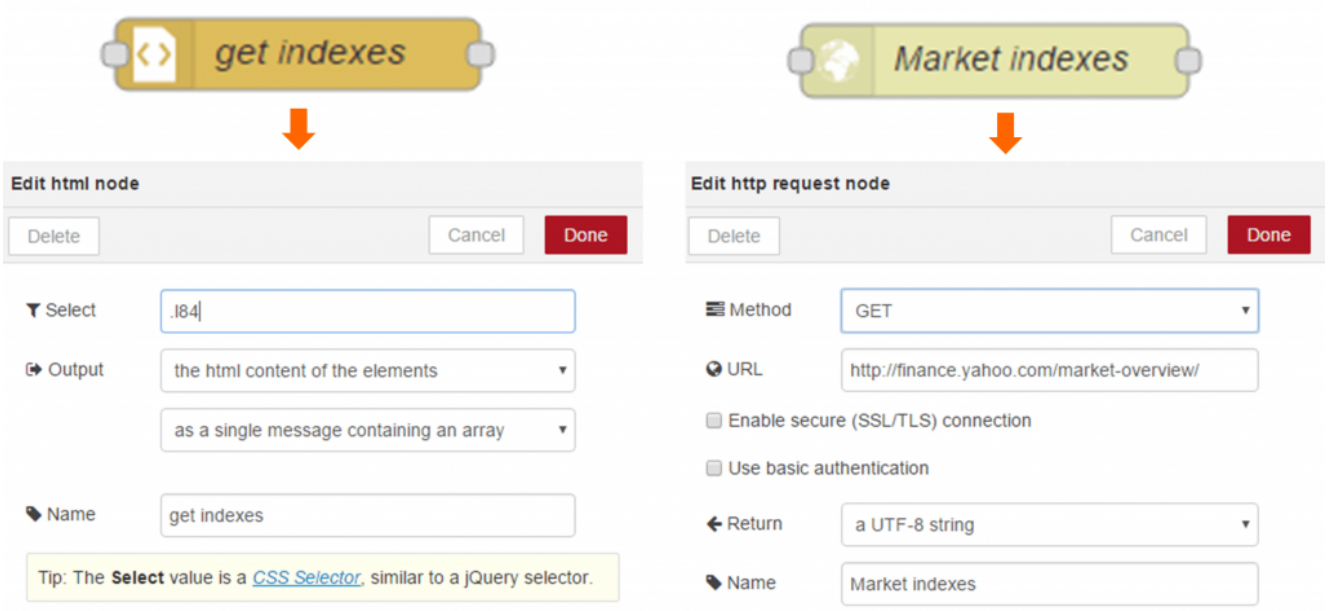


استرداد البيانات من صفحة ويب :

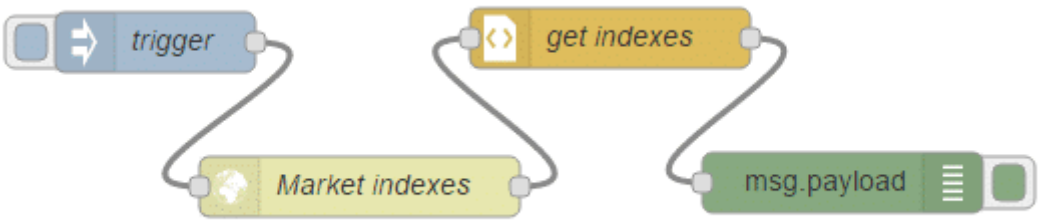
سنقوم بكتابة التدفق الذي يوفر معلومات حول مؤشرات البورصة من ياهو (Yahoo) في http://finance.yahoo.com/market-overview/ حيث يتم تنسيقه باستخدام عقدة function . إذا قمت بفحص الصفحة المعروضة في الشكل أدناه باستخدام "Inspect Element" في متصفح كروم ، فستجد أن المؤشرات الثلاثة لها نفس فئة CSS تسمى 'l84' (وهذا الحرف حرف "L" في المظهر الصغير) . يمكن استخدام هذه الفئة لاسترداد جميع المؤشرات الثلاثة من عقدة html . سوف نقوم بإعداد عقدة inject لتشغيل طلب http للحصول على الصفحة، ثم عقدة html للحصول على العناصر مع فئة 'l84' .



يتم تهيئة عقد HTML و http كما هو موضح بالصورة التالية :

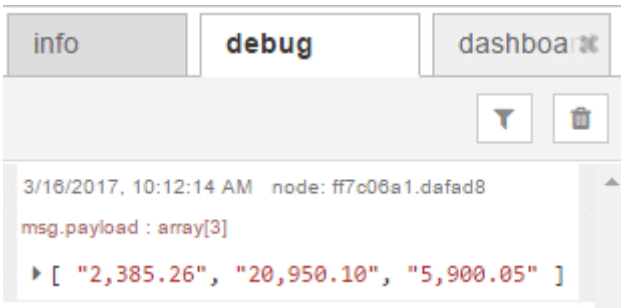


يتم تهيئة هذه العقد اعلاه لاسترداد مؤشرات البورصة . ثم قم بربط العقد معا كما هو موضح بالصورة أدناه ، مع عقدة debug لإظهار المخرج .



يجب أن يحتوي على أحدث قيم الفهرس . عند الضغط على التشغيل على عقدة inject . سيظهر لك ما يلي في جزء لوحة الاخراج عند

تبويب debug :



كان ذلك سهلا ، ولكن ما ترديه حقا هو مصفوفة من JSON وهي ازواج من الاسم:القيم التي تستخدم الاسم لوصف القيم و القيم كأرقام.
في الكود أدناه يوضح ناتج JSON المطلوب من مؤشرات الأسهم :

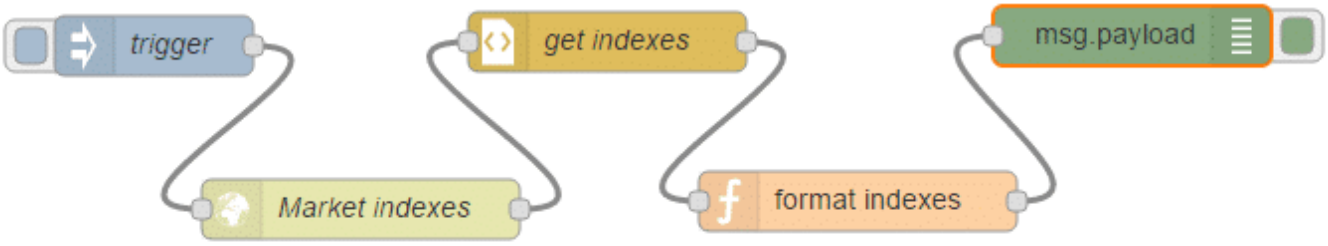
```
[{
    “index”:”S&P”,
    “value”:2096.92
 },{
    “index”:”Dow”,
    “value”:17511.34
 },{
    “index”:”Nasdaq”,
    “value”:5059.35
 }]
```

سنقوم بالكتابة على عقدة funcrion لتنسيق هذا بالطريقة التي تريدها . قم بالنظر إلى الكود أدناه فهو يقوم بإخذ اسعار الأسهم الواردة ويخزنهم في pices (كما هو موضح بالسطر الأول) ، ثم يتم إنشاء مصفوفتين جديدة، واحدة لإحتواء رسائل الإخراج و الثانية لإحتواء الاسم النصي للمؤشرات الثلاثة المختلفة.

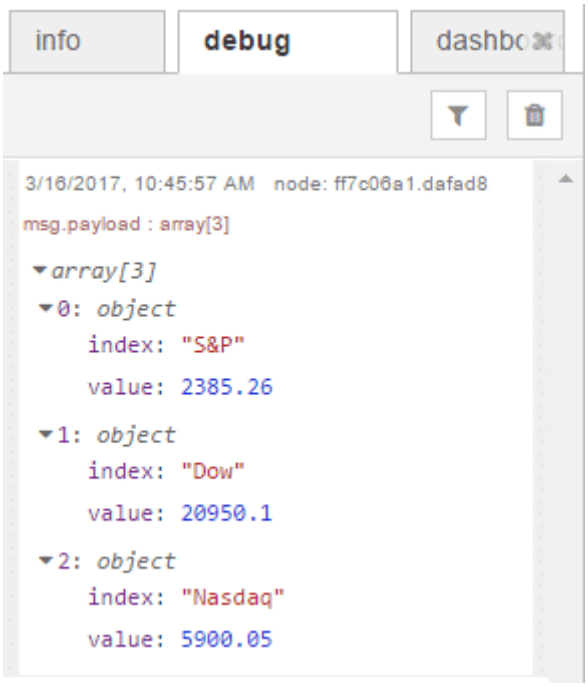
في السطور من 5–8 عبارة عن حلقة تكرارمن خلال مؤشرات الأسهم الواردة وكل واحد يدفع زوج من (الاسم:القيمة) إلى مصفوفة رسالة الإخراج. في السطر 8، يستخدم وظيفة replace بالجافا سكريب لإزالة كل الفواصل من قيم المؤشر قبل تحليلها كأرقام.

```
var prices = msg.payload;
var newPayload = [];
var priceIndex = ['S&P','Dow','Nasdaq'];

for (var i=0; i<prices.length; i++) {
  newPayload.push({
    index:priceIndex[i],
    value:Number(prices[i].replace(/,/g,''))
  });
}

msg.payload = newPayload;
return msg;
```

الآن قم بإضافة عقدة function ثم قم بكتابة الكود أعلاه :



بعد نشر و إختبار التدفق، يجب أن تقوم وحدة التحكم في debug بإخراج شيء مشابة للشكل أدناه . البيانات الآن تبدو جيدة ويمكنك استخدام هذا الشكل لعقد في التدفق الخاص بك .



عد الكلمات في السلسلة :

بعد ذلك، دعونا نكتب عقدة Function أكثر تعقيدا والتي تتلقى بعض النصوص في حمولة الرسالة (message payload) ، ثم إخراج رسائل متعددة تحتوي على كل الكلمات الفردية وعدد مرات استخدام كل كلمة .

الكود الموضح أدناه لعدد الكلمات :

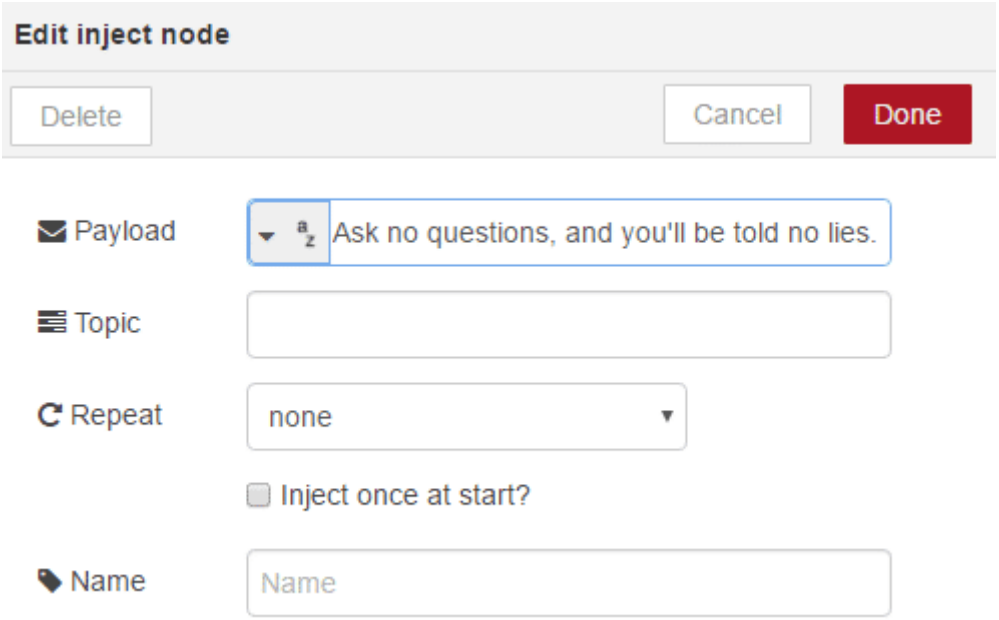
```
var outputMsgs = [];
var wordMap = {};

var sentence = msg.payload.replace(/[.,-\/#!$%\^&\*;:{}=\-_`~()]/g,"");
sentence = sentence.replace(/\s{2,}/g," ");

var words = sentence.split(" ");
for (var i = 0; i < words.length; i++) {
  var lowerCaseWord = words[i].toLowerCase();
  if (!wordMap[lowerCaseWord]) {
    wordMap[lowerCaseWord] = 1;
  } else {
    wordMap[lowerCaseWord] = wordMap[lowerCaseWord] + 1;
```

```
  }
}
for (var prop in wordMap) {
  if( wordMap.hasOwnProperty( prop ) ) {
    outputMsgs.push({payload:{word:prop,count:wordMap[prop]}});
  }
}
return [outputMsgs];
```

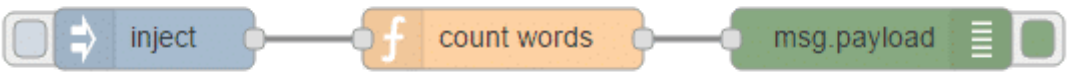
في الكود أعلاه، يتم تعريف قائمة رسائل الاخراج و كائن للاحتفاظ بعدد الكلمات (كما هو موضح بالسطر 1 و 2) . في السطر 4 و 5 يتم إزالة علامات الترقيم و المسافات الإضافية من الحمولة (payload) بإستخدام التعابير القياسية (regular expressions) . التعابير القياسية أو المنتظمة (regular expressions) هي أداه مفيدة جدا لمعالجة النصوص ، يمكنك معرفة المزيد عن هذه التعبيرات من خلال الإطلاع على صفحة مطوري برامج Mozilla للجافا سكريبت هنا . في السطر 7، يتم تقسيم النص إلى كلمات متعددة، ثم يكرر هذا من خلال الكلمات، حيث يتم إنشاء مصفوفة من الكلمات ذو الأحرف الصغيرة إلى عدد الكلمات الذي يدعى wordMap كما هو موضح بالسطور من 8 ـ15 . في السطور من 16 إلى 18 يتم تقسيم wordMap إلى رسائل متعددة في مصفوفة outputMsgs . وأخيرا، في السطر 21 ، يتم إرجاع المصفوفة التي تحتوي مجموعة الرسائل، و إرسالها جميعا إلى منفذ الإخراج الأول في وقت واحد.

دعونا ننشئ التدفق لنرى كيف يمكن أن تعمل مع بعض النص . أولا، قم بإنشاء عقدة function ، و نسخ التعليمات البرمجية أعلاه. ثم قم بإضافة عقدة inject :

قم بتهيئة عقدة inject و إضافة النص التالي كما هو موضح بالصورة :



قم بإضافة عقدة debug وربطها مع التدفق كما هو موضح بالشكل التالي :



عند النقر فوق عقدة inject ، يجب أن تشاهد قائمة و عدد الكلمات على لوحة الإخراج عند تبويب debug ، انظر الشكل أدناه:

استخدام السياق (Context) لتوليد متوسطات التداخل :

وحدة خاصة تسمى السياق (Context) ، وتستخدم لتخزين البيانات بين استدعاءات الـ function، وهو متاح في عقد الـ Function . يمكن أن يكون هذا مفيدا عندما تحتاج الدالة (function) للاحتفاظ بالحالة للقيام بمعالجتها. على سبيل المثال، قد يكون من الضروري حساب قيمة متوسط قراءات بيانات جهاز الاستشعار على مدى فترة من الزمن .

في الكود الموضح أدناه يتم احتساب متوسط التداخل للقيم المتلقاة خلال الخمس ثواني الأخيرة، يتم اضافة حقل 'average' إلى الحمولة (payload) ، عند انقضاء أكثر من 5 ثوان بين الرسائل المستلمة.

```
var currentTime = new Date().getTime();

if (!context.lastTime) {
    context.lastTime = currentTime;
    context.sum = msg.payload.value;
    context.count = 1;
}
```

```
if (currentTime-context.lastTime > 5000) {
    // calculate average for previous messages
    msg.payload.average = context.sum/context.count;
    // start tracking average again
    context.sum = msg.payload.value;
    context.count = 1;
    context.lastTime = currentTime;
} else {
    context.sum += msg.payload.value;
    context.count +=1;
}
return msg;
```

عند النظر إلى الكود، سوف تحصل أولا على الوقت الحالي (السطر 1) . إذا لم يكن هناك آخر وقت (lastTime) مخزن في السياق (Context) ، ستقوم بحفظ الوقت الحالي (currentTime) ، و إعادة تعيين مجموع (sum) و عدد (count) متغيرات السياق (كما هو مبين بالأسطر من 3 إلى 7).

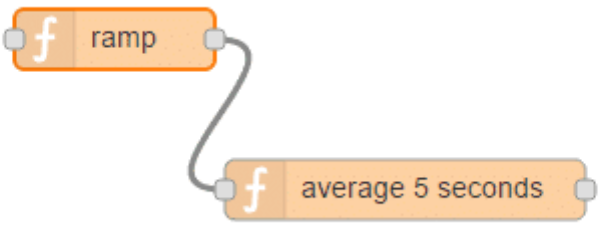
إذا كان الوقت الحالي 5(currentTime) ثواني (5000 ميلي ثانية) أكبر من آخر وقت (السطر 9)، فإنك تحسب متوسط القيم الأخيرة المستلمة و تدرج ذلك مع الحمولة للرسالة (في الاسطر من 11 إلى 13) . ثم يمكنك إعادة تعيين مجموع sum و عدد count و تعيين آخر وقت lastTime و الوقت الحالي currentTime لبدء العد مرة أخرى.

إذا لم يتم انقضاء 5 ثواني منذ استلام الرسالة الأخيرة، ، يجب تحديث counts و sums (كما هو مبين بالأسطر من 16–19) . ثم تقوم بإخراج الرسالة التي تتضمن أحدث قيمة و متوسط اذا تم حسابها للفترة الزمنية (السطر 20).

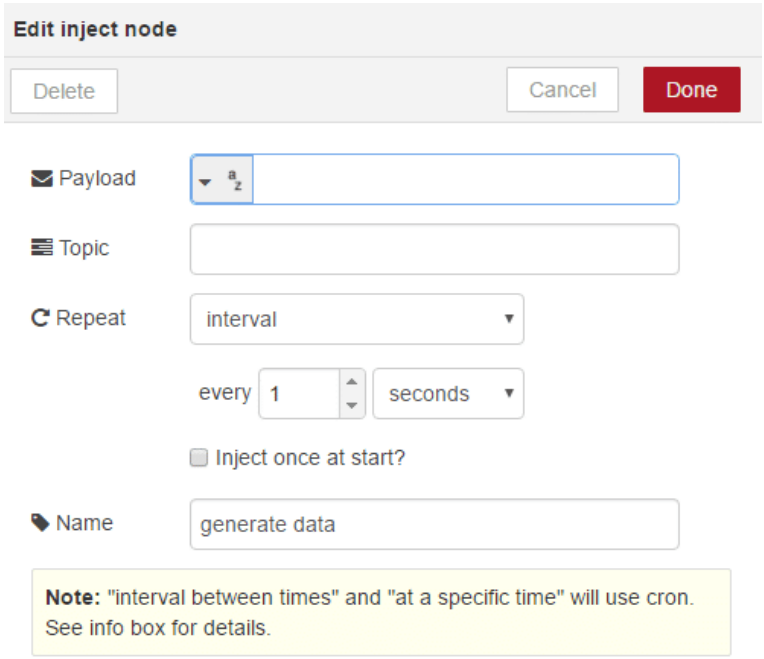
قم بإضافة عقدة Function وكتابة الكود أعلاه .

لاختبار هذا، سنقوم بكتابة عقدة Function أخرى، التي تسمى ramp ويستخدم أيضا السياق (context) لتوليد القيم من 0 إلى 9 في التسلسل ، كما هو موضح بالكود ادناه ، دالة (function) لتوليد تسلسل القيم بإستخدام السياق .

```
if (!context.value) {
  context.value = 0;
}
msg.payload = {
  value:context.value
}
context.value +=1;
if (context.value > 9) {
  context.value = 0;
}
return msg;
```

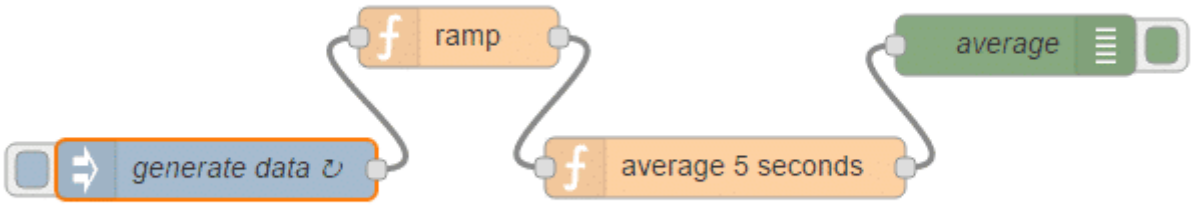


سنقوم إضافة و ربط عقدة inject إلى التدفق ، ثم قم بإعدادها لإرسال البيانات في كل ثانية، كما هو مبين بالشكل التالي :

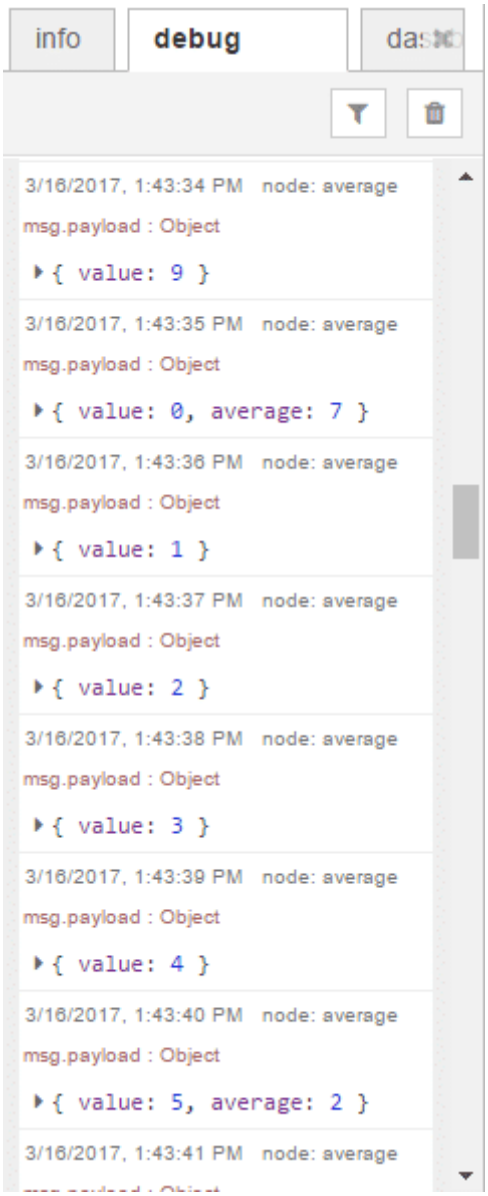


عقدة inject لضخ حمولة فارغة كل ثانية .

ثم قم بربط التدفق كما هو مبين بالشكل التالي :



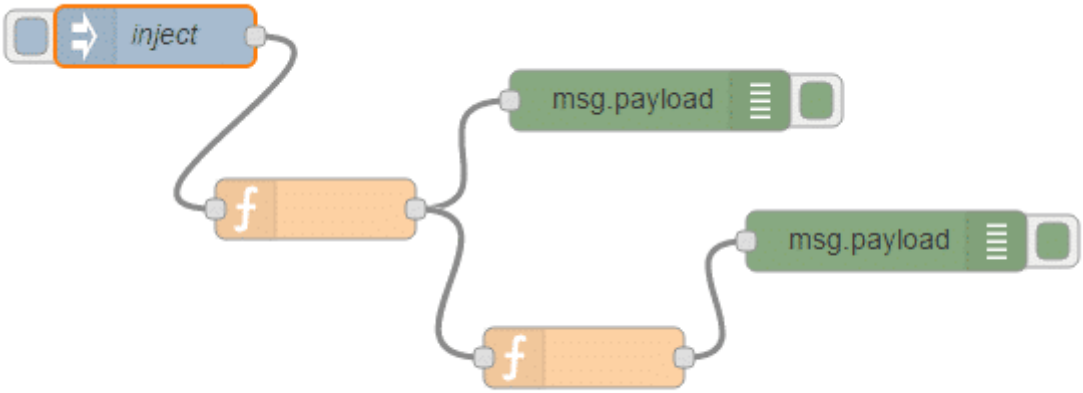
يجب أن يبدو الناتج كما هو موضح بالشكل التالي :



استخدام عنصر السياق لمشاركة الدالة وجعلها في متناول جميع الدوال (Functions) في اللوحة :

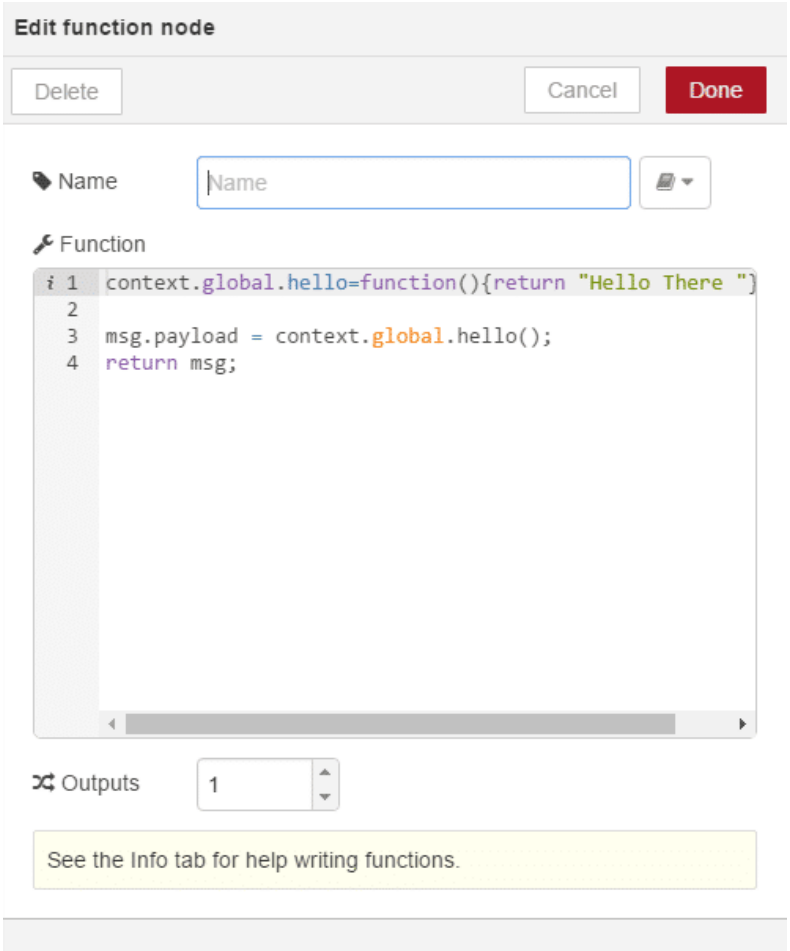
في هذا المثال، نوضح كيفية استخدام عنصر السياق (Context) و عنصره العالمي (global) لمشاركة البايانات عبر عقد الـ function . استخدم المثال السابق لتخزين قيمة عددية. ومع ذلك، واحدة من الأشياء العظيمة للجافا سكريبت هو أنه يمكنك تعيين (functions) للكائنات. هذا يسمح لك لتبادل الأساليب عبر اللوحات الخاصة بك دون الحاجة إلى إعادة تعريفها في كل عقدة function تستخدم على التوالي .

أولا، قم بتوصيل عقدة inject ، وعقدتين Function و عقدتين debug كما هو مبين أدناه :



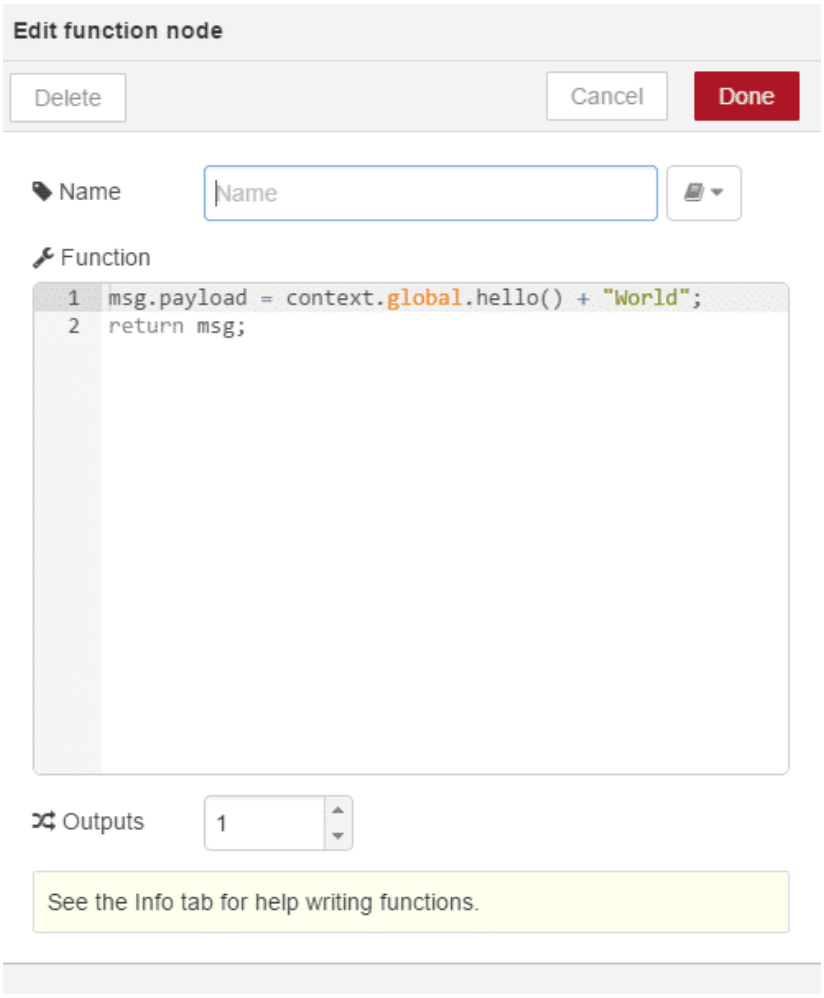
سنقوم بتحرير عقدة function الأولى و إضافة هذا الكود. قم بتعيين عنصر العام (global) من عنصر السياق ليكون عنصر جديد، "hello" ، وهي دالة التي تقوم بترجيع السلسلة "Hello There" . ثم قم بتعيين الحمولة (payload) للرسالة لهذه الدالة لإرجاع قيم لهذه الدالة العامة (global function) . هذا سيطبع "Hello There".

في الشكل أدناه يوضح كيفية إعداد عقدة Function الأولى :

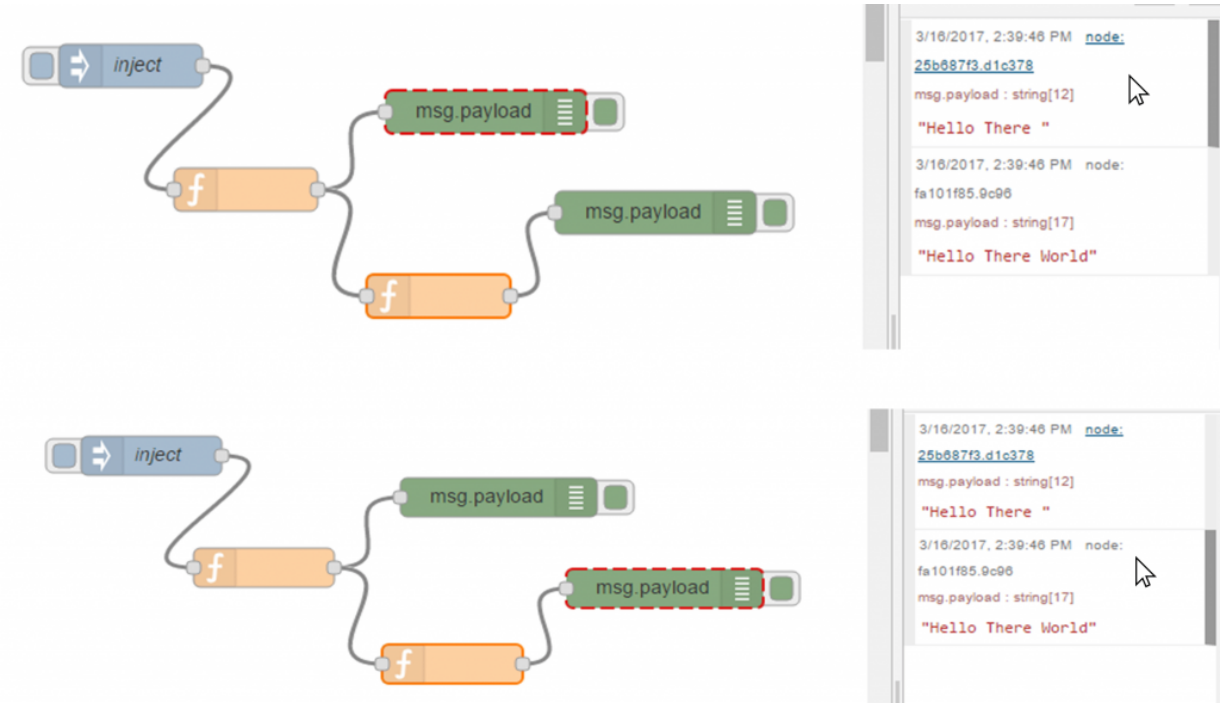


الآن قم بتحرير عقدة الدالة الثانية من خلال إعداد حمولة الرسالة من الدالة الثانية لتوصيل الإخراج من عنصر 'hello' في العنصر العام من كائن السياق مع كلمة "World".

في الشكل أدناه يوضح كيفية إعداد عقدة Function الأخرى :



قم بنشر التدفق، ثم الضغط على زر الموجود على عقدة inject سيظهر لك إخراج كل من عقد الـ Function كما يلي :



على الرغم من كونه بسيطا جدا، فإن هذا المثال يوضح مدى سهولة استخدام عنصر السياق لتعيين المتغيرات التي تحتوي على البيانات فقط ، ولكن أيضا الدوال (Functions) التي يمكن مشاركتها و الوصول إليها بواسطة عدة دوال(Functions) في اللوحة الخاصة بك .

هناك جانبان مهمان يجب مراعاتهما هما : أولا، يجب أن تأتي الـ Function التي تحدد العنصر العام (global) في عنصر السياق قبل

اي دالة (function) أخرى ترغب في استخدام هذا المتغير أو الدالة في التدفق. ثانيا، في الاصدار (1.11.0) للـ Node-red ، يبقى كائن السياق في الذاكرة بعد إعادة النشر و حتى يتم إعادة تشغيل Node-RED .لذلك حتى بعد حذف عقد الـ Function التي تقوم بتعيين عناصر في العنصر العام (global) ، ستظل هذه العناصر في المتناول حتى يتم تعيينها إلى قيمة خالية "null" .

تعريف و استخدام التدفق الفرعي للتكرار :

كما ذكرنا سابقا، يمكن استخدام التدفقات الفرعية لحزم الدوال (functions) في لوحة العقد الخاصة بك. في هذا المثال، سنقوم بإنشاء تدفق فرعي يعالج مصفوفة من القيم في حمولة الرسالة و ينتج مصفوفة جديدة تمت معالجتها. سنقوم عقدة upstream بتزويد الرسالة مع الحمولة تحتوي على مصفوفة من القيم التي سيتم معالجتها. المخرج الأول لعقدة التكرار (iterate) سوف تنتج كل رسالة من رسائل الحمولة مصفوفة رسال إدخال ، الواردة في حمولة الرسالة بالترتيب . المخرج الثاني سوف ينتج رسالة تحتوي على حمولة مع مصفوفة جديدة تحتوي على قيم تم معالجتها.

ابدأ بإنشاء علامة التويب لتدفق فرعي (sub-flow) جديد . قم بسحب عقدة function ، و اضافة الكود الموضح أدناه :

```
 var currentMsg = null, outMessage = null;
 var iState = msg.iState;

  if (!iState) {
     // we received an initial message
     // if the message is not an array, make it one
     if( Object.prototype.toString.call(msg.payload) !== '[object Array]' ) {
           msg.payload = [msg.payload];
     }
     iState = {};
     iState.index = 0;
     iState.inArray = msg.payload;
     iState.outArray = [];
     msg.iState = iState
   } else {
    // save results from the last iteration
     iState.outArray.push(msg.payload)
 }
  //If there are still objects left to iterate goto the next one in the original
array
  if (iState.index < iState.inArray.length) {
    currentMsg = msg;
    msg.payload = iState.inArray[iState.index];
  } else {
    currentMsg = null;
    outMessage = msg;
    msg.payload = iState.outArray;
    delete msg.iState;
}

iState.index ++;

return [currentMsg, outMessage];
```

شرح الكود :

في السطرين 1 و 2 ، يتم تعريف عدة متغيرات. currentMessage، لإحتواء رسالة الإدخال الحالية ، outMessage لإحتواء رسالة الإخراج النهائية التي تحتوي على مصفوفة الحمولة التي تم معالجتها، ويستعمل iState لتسهيل الوصول إلى الحالة الحالية من التكرار من الخاصية msg.iState . في حين يمكن استخدام السياق لإدارة حالة الدالة (function) ، فانه من السهل احتواء الحالة في الرسالة، حيث قد تتلقى مصفوفات جديدة كإدخال قبل إكمال التكرار في مصفوفة أخرى.

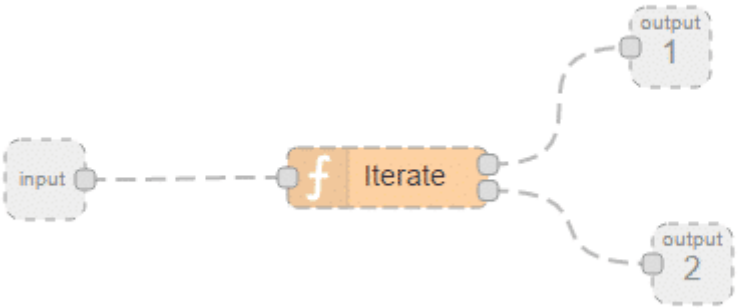
في السطر4، يمكنك التحقق لمعرفة ما إذا كانت iState موجودة . إذا لم تكن كذلك، افترض أن الرسالة هي رسالة إدخال جديدة. تحقق من أن الحمولة هي مصفوفة، وإذا لم تكن كذلك، سيتم جعلها واحدة (كما هو مبين بالأسطر من 7 إلى 9). ثم يتم إنشاء كائن حالة التكرار الذي يتضمن الفهرس الحالي، و مصفوفة الإدخال ، ومصفوفة الإخراج، و إضافته إلى msg (كما هو مبين بالاسطر من 10–14) .

إذا كانت iState موجودة (الاسطر 16–18) ، وانت على علم أن رسالة التكرار تم إنشاؤها بواسطى عقدة function و تم معالجتها في حلقات متكررة . سنقوم بدفعها إلى مصفوفة الإخراج لإرسالها عند الانتهاء.

بعد ذلك سيتم مراجعة ما إذا كنت قد انتهيت أم لا. إذا كنت بحاجة إلى تكرار مرة أخرى (إذا كان الفهرس هو أقل من طول مصفوفة الإدخال) ، نقوم بتعيين currentMessage إلى الرسالة و تعيين الحمولة payload إلى الفهر الحالي (كما هو مبين بالاسطر من 20 إلى 22) . عند الإنتهاء من التكرار ، يتم تعيين الرسالة الحالية (currentMessage) إلى null ، و حولة الرسالة إلى مصفوفة الإخراج (outputArray) وحذف خاصية iState التي لم تعد هناك الحاجة لها.

ثم قم بزيادة المؤشر و إخراج الرسائل إلى نقاط الإخراج. أثناء التكرار، يتم فحص الرسالة الحالية (currentMessage) للتاكد من أنها تحمل قيمة، وإلا (عندما تكون فارغة -null) تنتهي حلقة التكرار . و في النهاية ، الرسالة المخرجة (outMessage) لن تكون فارغة و سيتم إرسالها إلى المخرج الثاني.

الآن دعونا نقوم بربط المدخلات و المخرجات إلى التدفق الفرعي (sub-flow) كما هو مبين بالصورة التالية ، سنقوم بتسمية هذا التدفق بإسم 'Iterate' .



لاختبار التدفق، قم بإنشاء مصفوفة من 1 إلى 5 كما هو موضح بالكود أدناه، ومضاعفة كل عنصر بهذه المصفوفة بمقدار 5 .

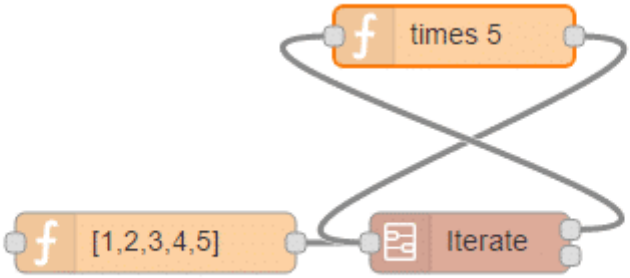
```
msg.payload = [1,2,3,4,5];
return msg;
```

للقيام بذلك، قم بإنشاء عقدة Function جديدة تسمى [1,2,3,4,5] ثم قم بنسخ الكود أعلاه، حيث يتم بربطها بالتدفق كما هو موضح بالصورة التالي :

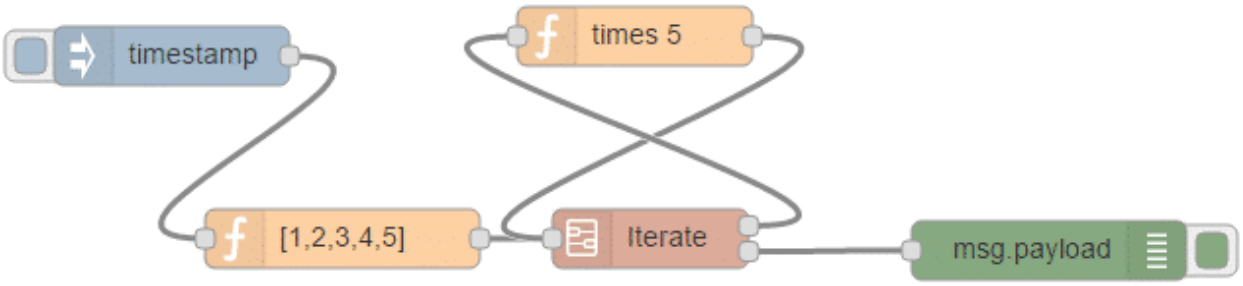
قم بكتابة الدالة (function) الثانية والتي يتم استخدامها في كل تكرار، كما هو موضح أدناه، حيث تقوم العقدة بمضاعفة الحمولة بمقدار 5 :

```
msg.payload = msg.payload * 5;
return msg;
```

قم بإسقاط عقدة function تم تسميتها بـ times 5 ، قم قم بربط مدخل العقدة إلى المخرج 1 لعقدة التدفق الفرعي و مخرج العقدة إلى مدخل عقد التدفق الفرعي، كما هو موضح بالشكل التالي :



وأخيرا لتشغيل التدفق، استخدم عقدة inject، و إضافة عقدة debug . كما هو موضح بالصورة التالية :



قم بالنقر على عقدة inject، يجب أن تعرض لك نافذة الإخراج [5,10,15,20,25]عند تبويب debug .

الآن بعد إنشاء عقدة لوظيفة التكرار في التدفق الفرعي يمكنك استخدامها في أي من التدفقات الخاصة بك. فسيتم التكرار على اي مصفوفة تحتوي على قيم صحيحة وتطبيق الدالة المحددة.