ABSTRACT
This paper explores features and complexity in the use of Put and INput functions. Put and INput are useful for converting variables from character to numeric and the reverse. This paper starts with simple examples and builds to more complex examples showing dates and nested use (e.g. Put(INput(cust_id,6.0),Z6.0)).

The Put function converts from a numeric or character input/variable/target to a character output, applying a format as part of the conversion. The Put function can be applied to a character or numeric input, but the output of the Put function is always character. INput converts only from a character input/variable/target to either numeric or character output/variables, applying an INFormat as part of the conversion. The INput function can only be applied to a character target, but the output can be character or numeric. As a quick summary, Put can take two types of input and produces one type of output (character). INput takes one type of input (character) and can produce two types of output.

We have attempted to explore all possible combinations of inputs, formats/INFormats and the two functions. The code required was very long and is included in the appendix. To make this paper more readable, we will illustrate the rules we discovered with a few examples.

INTRODUCTION
The use of Put and INput can be remembered if one recalls the original use of INFormats and formats. “In the old, old days” data sources and destinations were usually text. An illustration of reading from/writing to text is shown below.

INput statements, and INformats, are used to convert external “text” source files into SAS character and numeric variables – converting from text to two variable types.

Put statements, and formats, are used to convert SAS character and numeric variables “into character” – two types into one.

COMBINATIONS OF SITUATIONS INVOLVING USING PUT AND INPUT FUNCTIONS
In this paper, the names of formats, INFormats and variables were created in a manner so as to contain hints about the contents of the format, INFormat or variable. It is suggested that a reader pause, and study for a while, when naming logic is explained. It is hoped that an understanding of the naming logic will make the paper more readable.
Put and INput functions are difficult to understand because several factors are associated with their use and the factors combine to create a large number of situations we must code, organize, study and then master. This situation was approached as problem in statistical combinatorics. Some combinations were expected to throw errors.

However, SAS is becoming more “helpful” and more tolerant of bad typing/coding. Our combinatoric analysis led us to apply formats/INformats to values of the appropriate type (character or numeric) and also to inappropriate types. Some of these examples did cause Proc Format or the Data Step to throw errors.

As we organized the research for this paper, we discovered four possible ways to code the creation of a format statement and four ways to code an INformat statement. Accordingly we coded a Proc Format that created four formats and four INformats. Some of the syntax that our combinatorical analysis created were, in fact, invalid SAS syntax and threw errors.

In this paper, we used very simple formats/INformats. Each format/INformat mapped 123 to one other number. An example is: Proc format; value $FCCC "123" ="500"; This format only maps from one value and has a very small “mapping range”.

Our combinatoric analysis showed us we could possible apply each of these formats/INformats to targets with different characteristics. We needed to create targets that would fully test the use of formats/INformats and we created six Targets:

CV123=’123’; NV123=123; *Numeric & character version of 123–the formats map from 123;
Nout=’222’; * a numeric variable - not 123 and so outside the "mapping range";
Cout=’222’; * a character variable of the SAME length as the RHS of = in the Value statement but not in the "mapping range". Investigates "bad data" situations.
CShortOut=’9’; * a character variable of SHORTER length than the RHS of = in the Value statement and outside the "mapping range". Investigates "bad data" situations.
CLongOut=’33333’; * a character variable of LONGER length than RHS of = in the Value statement and so outside the "mapping range". Investigates "bad data"

These formats/informats and variables combined to create a large number (2 levels of Format/INformat * 4 ways of coding a format/INformat * 6 kinds of targets) of “states of nature” that had to be explored. This paper will only list and explain the rules we abstracted. The code, and results, are included in the appendix. An interested reader can copy the code from the appendix, paste it into SAS and either validate our rules or expand the research into new areas.

ABSTRACTED RULES AND EXAMPLES

FORMAT CREATION AND USE

A format changes how a SAS variable is displayed – but does not change the stored value of the variable. Some examples may clarify this statement. Think of a numeric variable X that has a value of 1. Here are some basic examples of how formats change how a value is displayed, but not how it is stored.

```
X=1;
Put @10 X weekday.--------- The 1 displays as a 7
@20 X 5.3 ------------- The 1 displays as a 1.000
@25 X 5.1 ------------- The 1 displays as a 1.0
@30 X z5.2 ------------- The 1 displays as a 01.00
@40 x worddate20.------ The 1 displays as January 2, 1960
@70 X wekddate17.----- The 1 displays as Sat, Jan 2, 1960
```

X has the value of 1 but is displayed in many ways.

We would like to discover more format rules using the code below. Below we investigate format creation.

```
proc format;  /*Create 4 char and 4 num formats - with odd coding*/
*sample character formats - a combinatoric problem;
*coding/naming convention $ is to declare to SAS that this to be a character format
F stands for format (as opposed to INformat)
C stands for character format- for human readability
CC means LHS is Char and RHS is Char
NC means LHS is Num and RHS is Char
CN means LHS is Char and RHS is Num
NN means LHS is Num and RHS is Num;
/* 1*/ value $FCCC "123" ="500";*-NOTE: Format $FCCC has been output.;
/* 2*/ value $FCNC 123 ="500";*-NOTE: Format $FCNC has been output.;
/* 3*/ value $FCCN "123" = 500 ;*-NOTE: Format $FCCN has been output.;
/* 4*/ value $FCNN 123 = 500 ;*-NOTE: Format $FCNN has been output.;
```
*sample numeric formats— a combinatoric problem;*

*coding/naming convention: The lack of a $ is to declare this to be a Numeric format  
F stands for format (as opposed to INformat)  
N stands for Numeric format— for human readability  
CC means LHS is Char and RHS is Char  
NC means LHS is Num and RHS is Char  
CN means LHS is Char and RHS is Num  
NN means LHS is Num and RHS is Num;  

/* 5*/ value FNCC "123"="500";  *ERROR: string '123' not acceptable to numeric format/INformat;  
/* 6*/ value FNNC 123 ="500";  *->NOTE: Format FNNC has been output. ;  
/* 7*/ value FNF_CN "123"=500;  *ERROR: string '123' not acceptable to numeric format/INformat;  
/* 8*/ value FNNN 123 = 500 ;  *->NOTE: Format FNNN has been output. ;

run;

We suggest that the reader note examples /* 4*/ and /* 6*/.  They seem as if they might be mis-coded. /* 4*/ is a character format (name starts with $) but the values in the value statement are both numeric. /* 6*/ creates a numeric format but the value on the right hand side (RHS) of the equal sign is character. /* 5*/ and /* 7*/ threw errors.

SAS character formats are forgiving, on creation.  When creating a character format, SAS does not care if the values on the right or left hand side of the equal sign are numeric or character.  When creating a numeric format, the value to the left of the equal sign must be numeric.  The value to the right of the equal sign can be character or numeric.

The Put function uses a format to convert from a numeric variable, a character variable, or a constant and returns a character value.  The Put function is most often used for converting a numeric value to a character value (without generating a note in the log).  We will use the formats we created above to show details of Put function use.  The Put statement has two parameters (target and format) separated by a comma.  An example is: newvar = Put (target , format.).

We were particularly interested in applying formats /* 4*/ and /* 6*/ (created above) because they seemed to be constructed wrongly, or at least oddly. /* 4*/ is a character format but the values on the left and the right of the equal sign are numeric. In /* 6*/, we created a numeric format, but put a character value on the right of the equal sign.  While SAS created these formats without any difficulties we wondered if these oddly coded formats could cause problems in use.

In the appendix we take the formats and apply them to many character and numeric targets.  This allowed us to abstract the rules below and we suggest a reader keep the formats and target variables in mind as rules are discussed.  In the paper itself, we show just a few examples from the appendix to illustrate the rules we abstracted.

**Rule 1:** On format creation:
Character formats can have numeric values on LHS and RHS of the = in the value statement.

```
Proc format;  /* 4*/ value $FCNN 123 = 500 ;  *->NOTE: Format $FCNN has been output.;
```

Numeric formats must have a numeric value on LHS of the = in the value statement.

```
/* 5*/ value FNCC "123"="500";  *ERROR: string '123' not acceptable to numeric format/INformat;  
/* 6*/ value FNNC 123 ="500";  *->NOTE: Format FNNC has been output. ;  
/* 7*/ value FNF_CN "123"=500;  *ERROR: string '123' not acceptable to numeric format/INformat;  
/* 8*/ value FNNN 123 = 500 ;  *->NOTE: Format FNNN has been output. ;
run;
```

**Rule 2:** The Put statement accepts numeric or character targets, and numeric or character formats, but always produces a character value.

```
Proc format;  /* 1*/ value $FCCC "123"="500";  *->NOTE: Format $FCCC has been output.;  
/* 6*/ value FNCC 123 ="500";  *->NOTE: Format FNCC has been output.;
```

```
Data FormatUse;  
CV123="123"; NV123=123; Nout=222; Cout="222"; CShortOut="9"; CLongOut="33333";  
FCCC_1_C123 =Put(CV123,$FCCC.);  *VALUE= 500 - TYPE=CHAR3;  
FNNC_6_N123 =Put(NV123,FNNC.);  *VALUE= 500 - TYPE=CHAR3;  
Run;
```

FCCC_1_C123 and FNNC_6_N123 are both valued at 500 and char3.

*To save space, the examples will no longer show the full “data step” code.*
**Rule 3:** If the target is character we must apply a character format to it and the most interesting example is `/*4*/`. As shown in `/*4*/`, a character format can be created with numeric values on the left and right sides (LHS RHS) of the equal sign, but the format must be character (have a `$` as the first character in the format name).

```
Proc format; /* 4*/ value $FCNN 123 = 500 ; *--NOTE: Format $FCNN has been output. ;
FCNN_4_C123 =Put(CV123,$FCNN.); *VALUE= 500 - TYPE=CHAR3;
```

**Rule 4:** If the target is numeric we must apply a numeric format. The most interesting examples are `/*6*/` and `/*8*/`. The name of a numeric format can not start with a `$`. When a numeric format is created, there must be a numeric value on the left hand side (LHS) of the equal sign. When the format is created, the type (character/numeric) value on the right hand side (RHS) of the equal sign is irrelevant. SAS acts as if it converts numeric values on the RHS of the equals to character values.

```
Proc format; /* 6*/ value FNNC 123 = "500" ; *--NOTE: Format FNNC has been output. ;
FNNC_6_N123 =Put(NV123,FNNC.); *VALUE= 500 - TYPE=CHAR3;
/* 8*/ value FNNN 123 = 500 ; *--NOTE: Format FNNN has been output. ;
FNNN_8_N123 =Put(NV123,FNNN.); *VALUE= 500 - TYPE=CHAR3;
```

**Rule 5:** The output of a Put will be character. In `/*1*/` the RHS of the equal sign is character and the result of the Put is character. In `/*3*/` the RHS of the equal sign is numeric and the result of the Put is still character.

```
Proc format; /* 1*/ value $FCCC "123" ="500" ; *--NOTE: Format $FCCC has been output. ;
*FCCC_1_C123 =Put(CV123,$FCCC.); *VALUE= 500 - TYPE=CHAR3;
/* 3*/ value $FCCN "123" = 500 ; *--NOTE: Format $FCCN has been output. ;
*FCCN_3_C123 =Put(CV123,$FCCN.); *VALUE= 500 - TYPE=CHAR3;
```

**Rule 6:** As can be seen below, Put does NOT help us by performing variable type conversions if we miscode. If we apply a character format to a numeric variable, SAS sends us a message that the variable has been defined as numeric. If we apply a numeric format to a character variable, SAS sends a message that the format could not be found. The fact that SAS sends an error and grabs our attention is very helpful.

```
*FCCN_3_N123 =Put(NV123,$FCCN.); *Variable NV123 has already been defined as numeric.;
*FNNC_6_C123 =Put(CV123,FNNC.); *ERROR 48-59:The format $FNNC not found/could not be loaded
```

**Rule 7:** For put statement - treatment of targets out of “mapping range”
The “out of mapping range” issue needs separate development and different examples from the code shown above.

The treatment of values not in the “target range” can cause problems and the major issue is truncation. The default behavior of a format is to “show” any “non-mapped” values as they are in the data. This is good, if one is formatting values in a data set and less good if one is using formats to recode values. There is a danger when we pass values that are outside the mapping range to the Put. Good programming practice, when creating formats, demands the coding of an OTHER category.

**Character formats and the put**
The length of the output variable is determined by the length of the longest variable on the RHS of the equal sign ("xxxxYYY"). Notice, in the example below, that HowLongA is seven characters, not as long as the target string we fed to the Put. Only a truncated version of the target was passed through to the output variable.

```
data TestingLength;
HowLongA=put("123456789",$testB.);
HowLongB=put("10",$testB.);
HowLongC=put("210",$testB.); run;
Proc print data=TestingLength; run;
```

```
Obs   HowLongA   HowLongB   HowLongC
1      1234567     xxxx      xxxxYYY
```

If a target is not in the “mapping range” and is shorter than the longest variable on the RHS of the equals, there is no problem. If the target is longer than the longest variable on the RHS of the equals, it is truncated. Here are examples of applying a Put and formats to variables not in the “mapping range” for our formats.

Consider the variables CShortOut="9"; CLongOut="33333";

Applying a character format causes no problems for the short variable but truncates the longer variable.

```
FCNN_4_COutShort =Put(CShortOut,$FCNN.); *VALUE= 9 - TYPE=CHAR3 -passed through;
FCNN_4_COutLong =Put(CLongOut,$FCNN.); *VALUE= 333 - TYPE=CHAR3 - truncated;
```

NUMERIC FORMATS AND THE PUT

Numeric formats also take the length of the output variable from the length of the longest string on the RHS of the equals in the value statement. Numbers shorter than that length are passed through as they were stored. Numbers longer than that length are converted to scientific notation and then passed through.

*NUMERIC FORMAT AND PUT;
Proc format;
value testNB 1 = "xx"  210= "xxxx"  9999= "xxxxYY"; run;

```
data TestingLengthN;
  HowLongA=put(1,testNB.);
  HowLongB=put(210,testNB.);
  HowLongC=put(9999,testNB.);
  HowLongD=put(777777,testNB.);
  HowLongE=put(55,testNB.);
run;
```

**INFORMAT CREATION AND USE**

**INFORMAT CREATION**

IN formats change how a value is stored. This implies that IN formats are used in the movement or transformation of data starting in one “place” (maybe an external file or a SAS data set) and becoming data inside a SAS file.

For example, If 01/01/60 is read with an INformat of mmddyy8. SAS stores a zero (A SAS date is the number of days since Jan, 1 1960). If the value (500) is read with an INformat of comma10., SAS stores –500. If the value 1,000,000 is read with an INformat of comma10. SAS stores 1000000. IN formats change how a value is stored.

An INPut function uses an IN format to convert to either character or numeric but requires a character target. The INPut function has two parameters (a character target and an IN format) separated by a comma. An example is newvar = INPut (CHARtarget, INformat.); Below, we investigate IN format creation.

```
Proc format;/*Create 4 char and 4 num IN formats - with odd coding*/
  *sample character IN formats;
  *coding/naming convention: $ is to declare to SAS that this to be a character IN format
  I stands for IN format (as opposed to format)
  C stands for character format- for human readability
  CC means LHS is Char and RHS is Char
  NC means LHS is Num and RHS is Char
  CN means LHS is Char and RHS is Num
  NN means LHS is Num and RHS is Num;
  /*note IN Value is used to make an IN format*/
  /*1*/ INValue $ICCC '\123'="100"; /* ->NOTE: IN format $ICCC has been output.;
  /*2*/ INValue $ICNC 123 =100; /* ->NOTE: IN format $ICNC has been output.;
  /*3*/ INValue $ICCN '123' = 100 ; /* ->NOTE: IN format $ICCN has been output.;
  /*4*/ INValue $ICNN 123 = 100 ; /* ->NOTE: IN format $ICNN has been output.;
  *sample numeric IN formats;
  *coding/naming convention: lack of a $ declares to SAS that this is a numeric IN format
  I stands for IN format (as opposed to format)
  N stands for character format- for human readability
  CC means LHS is Char and RHS is Char
  NC means LHS is Num and RHS is Char
  CN means LHS is Char and RHS is Num
  NN means LHS is Num and RHS is Num;
  /*5*/ *IN Value INNC '123'="100"; /*ERROR: string '10' not acceptable to numeric, format/IN format;
  /*6*/ *IN Value INNN 123 =100  *ERROR: string '10' not acceptable to numeric, format/IN format;
  /*7*/ INValue INCN '123' = 100 ; /* ->NOTE: IN format INCN has been output.;
  /*8*/ INValue INNN 123 = 100; /* ->NOTE: IN format INNN has been output.; run;
```
The **IN** put function uses an **IN** format to convert a character target to either a character or numeric output. We were particularly interested in applying several of the above **IN** formats because they seemed to be constructed wrongly, or at least oddly, since the target of an **IN** put function must be character.

It seems that the **IN** formats /* 2*/, /* 4*/ and /* 7*/ will likely teach us most about the process. /*2*/ is a character **IN** format but the LHS of the equal sign is numeric. /*4*/ is a character **IN** format but the LHS and RHS of the equal sign are both numeric. /*7*/ is a numeric **IN** format but the LHS of the equal sign is character.

In the appendix we take the **IN** formats, created above, and apply them to many character and numeric targets. This allowed us to abstract the rules below and we suggest a reader keep the **IN** formats and target variables in mind as rules are discussed. In the paper itself, we show just a few examples from the appendix to illustrate the rules we abstracted.

**Rule 1:**
Like character formats, character **IN** formats are forgiving. When creating a character **IN** format, SAS does not care if the values on the right or left hand side of the equal sign are character. When creating a numeric format, the value to the right of the equal sign must be numeric. The value to the left of the equal sign can be character or numeric.

```plaintext
/* 4*/ INvalue $ICNN 123 = 100 ;* ->NOTE: INformat $ICNN has been output.;
/* 5*/ *INvalue INCC "123"="100"; *ERROR:string '10' not acceptable to numeric. format/INformat;
/* 6*/ INvalue INNC 123 = "100"; *ERROR:string '10' not acceptable to numeric. format/INformat;
/* 7*/ INvalue INCN "123"= 100 ;* ->NOTE: INformat INCN has been output.;
/* 8*/ INvalue INNN 123 = 100 ;* ->NOTE: INformat INNN has been output.;
```

**Rule 2:** The **IN** put function wants character targets, and either numeric or character **IN** formats. **IN** put produces either character or numeric values.

```plaintext
ICCC_1_C123 =INput(CV123,$ICCC.); *VALUE= 100 - TYPE=CHAR3;
INCN_7_C123 =INput(CV123,INCN.); *VALUE= 100 - TYPE=Num 8;
```

**Rule 3:** If we pass a numeric target to an **IN** put function, SAS will convert the target to character and write a note in the log. It will create the variable in the PDV and send it to the output data set, **but the variable will not be valued.** It “appears” as if SAS helped us by converting and that the **IN** put was successful. We consider this a subtle error.

```plaintext
ICCC_1_N123 =INput(NV123,$ICCC.); *Not valued-TYPE=CHAR3- Num-to-char note in log;
ICNC_2_NoutSame =INput(Nout,$ICNC.); *Not valued-TYPE=CHAR3- Num-to-char note in log;
```

The note in the log is placed after the data step and looks like this.

<p>| NOTE: Numeric values have been converted to character values at the places given by: |</p>
<table>
<thead>
<tr>
<th>Line:</th>
<th>Column:</th>
</tr>
</thead>
</table>

**Rule 4:** If the **IN** format applied is character (the **IN** format name starts with $), **IN** put returns a character. The target is always character.

```plaintext
ICNN_4_C123 =INput(CV123,$ICNN.); *VALUE= 100 - TYPE=CHAR3;
```

**Rule 5:** If the **IN** format applied is numeric, the function returns a numeric value. The target is always character.

```plaintext
INNN_8_C123 =INput(CV123,INNN.); *VALUE= 100 - TYPE=Num 8;
```

**Rule 6:** **INPUT** - TREATMENT OF TARGETS OUT OF “MAPPING RANGE”
The “out of mapping range” issue needs separate development and examples different from the ones used above. The way **IN** put and **IN** formats treat values not in the “target range” can cause problems. Good programming practice, when creating **IN** formats, demands the coding of an OTHER category.

The default behavior of an **IN** format is to let values that are not in the “mapping range” flow as they are into the new variable. A major issue is truncation and determining how many positions will be passed through to the output variable. This is complicated by the fact that **IN** put and **IN** formats create both character and numeric variables.
INPUT - CHARACTER TO CHARACTER CONVERSION - TARGETS OUT OF CHARACTER “MAPPING RANGE”

Truncation is the major issue here. In our investigation, the first step is to create a character INformat that will let us determine how the length of the “output” string is determined. Remember for a character INformat, we can have character values on both the LHS and the RHS of the equals. Values outside the mapping range will be bolded.

*INPut - the longest string is on the RHS of the =;
Proc format ; /* $ in name and the conversion is Char -> Char */
invalue $CTestCN "12" ="joy12" /*2 char on left and 5 char on right*/
"123" ="Gladi123" /*3 char on left and 7 char on right*/
"1234"="happy1234"; /*4 char on left and 9 char on right*/

data TestingLengthV2; *will create a Char. variable;
    HowLongA=INput("123",$CTestCN.); /*Mapped*/
    HowLongB=INput("1234",$CTestCN.); /*Mapped*/
    HowLongC=INput("12345",$CTestCN.); /*Will not be truncated*/
    HowLongD=INput("9876543210",$CTestCN.); /*truncated*/
    HowLongE=INput("1234987654",$CTestCN.); /*truncated*/
run;
proc content data=TestingLengthV2;run;
proc print data=TestingLengthV2;
1     GLadi123   happy1234   12345     987654321    123498765

In the above example, the longest string in the value statement was 9 characters and was on the RHS of the equal sign. SAS passed ONLY nine characters, from values that were outside of the “mapping range” to created character variables.

Let’s see if SAS does something different if the longest string is on the LHS of the equal sign, as is shown below.

*INPut - the longest string is on the LHS of the =;
Proc format ; /* $ in name and Char=Char */
invalue $CT_CN "12" ="joy12" /*2 char on left and 5 char on right*/
"123" ="Gladi123" /*3 char on left and 7 char on right*/
"123happy"="happyR"; /*8 char on left and 6 char on right*/

data TestingLengthV3;
    HowLongA=INput("123",$CT_CN.); /*will create a Char. variable;
    HowLongB=INput("1234",$CT_CN.); /*Not Mapped Not Truncated*/
    HowLongC=INput("12345",$CT_CN.); /*Not Mapped Not Truncated*/
    HowLongD=INput("9876543210",$CT_CN.); /*truncated*/
    HowLongE=INput("1234987654",$CT_CN.); /*truncated*/
run;
proc content data=TestingLengthV3;run;
proc print data=TestingLengthV3;
1     GLadi123   1234     12345     987654321    123498765

In the above example, the longest string in the value statement was 8 characters and was on the LHS of the equal sign. SAS passed ONLY eight characters, from values outside the “mapping range” to created character variables.

It seems, for user-created character INformats, SAS looks at the longest string, on either the RHS or LHS of the equals, and uses that to determine how many characters it should take from values outside the “mapping range”.

With user-created INformats, character values that are out of “mapping range” are truncated to the length of the longest value in the INformat.
INPUT - CHARACTER TO NUMERIC CONVERSION - TARGETS OUT OF “MAPPING RANGE”

This situation initially appears to be simpler than the one above since SAS throws an error if we try to create a numeric INformat with a character value to the right of the equal sign. Additionally, an INput, with a numeric INformat creates a numeric variable and we need not worry about the length of a created numeric variable. However, numeric out of “mapping ranges” can have their own subtle problems.

Let’s first examine a character to numeric conversion.

```
*Numeric INformat;
Proc format; /*no $ in name: Do a Char->Numeric conversion */
invalue NT_CN "1" =1000 /*1 char on left of = */
"12"=10000 /*2 char on left of = */
"123"=100000; /*3 char on left of = */
data TestingLengthV4;
HowLongA=INput("1",NT_CN.); /*Maps*/
HowLongB=INput("12",NT_CN.); /*Maps*/
HowLongC=INput("123",NT_CN.); /*Maps*/
HowLongD=INput("9876543210",NT_CN.); /*Truncates*/
HowLongE=INput("1234987654",NT_CN.); /*Truncates & Maps*/
run;
proc contents data=TestingLengthV4; run;
Proc print data=TestingLengthV4; run;
```

The effect of out of “mapping range” is complicated and is best understood by reading the comments below.

<table>
<thead>
<tr>
<th>1 was in “mapping range”</th>
<th>12 was in “mapping range”</th>
<th>123 was in mapping range</th>
<th>Target was truncated to 987</th>
<th>Target was truncated to 123</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10000</td>
<td>100000</td>
<td>987</td>
<td>100000</td>
</tr>
<tr>
<td>and mapped to numeric 1000</td>
<td>and was mapped to numeric 10000</td>
<td>and was mapped to numeric 10000</td>
<td>987 is not in “mapping range” and was passed through as numeric 987</td>
<td>123 is in “mapping range” and was mapped to numeric 100000.</td>
</tr>
</tbody>
</table>

Passing out of “mapping range” values to user-created numeric INformats, seems to be a dangerous practice.

COMBINING PUT AND INPUT AND TRICKS FOR UNUSUAL SITUATIONS

Often SAS programmers are more interested in tricks they can use in their programs than in tedious explanations of SAS internal processes. In an attempt to please those readers we offer a few examples of how to use put and INput.

One example is the commonly encountered need to zero fill character fields (like zip codes that had been converted to numeric at some time, losing their leading zeros in the process).

The second example is converting text representations of dates into SAS dates.
“ZERO FILLING” EXAMPLE / TRICK

It is common to get data from clients in a variety of formats and with a variety of data quality issues.

A simple version of this “zero filling” problem is zip codes being delivered as numeric, without leading zeros.

To convert numeric zips to zero filled character, use a Put. 

CharZip=Put(NumZip , Z5.);

If the zip code is character, without leading zeros, the problem is more complex. The solution is illustrated below.

In a more complex example, character product codes have been entered oddly- with the main issues being

1) a lack of leading zeros for some of the product codes
2) alignment.

The business situation is:

Product codes are all six place character information.

For a valid product code: a letter is allowed only in position one. All other places should be zeros or digits.

Data entry was not very disciplined but the only error in our example is a lack of leading zeros.

Product codes starting with letters (A12345) were all entered correctly.

The code below uses Put and InPut to fix this problem. The trick is shown below in blue.

```
data combo;
  infile datalines truncover;
  input @1 odd_prod $char6.;
  if upcase(substr(odd_prod,1,1)) GE "A" and /* GE, LE logic used here*/
    upcase(substr(odd_prod,1,1)) LE "Z" then NewProd=odd_prod; /*this obs started with a letter - do not fix this obs*/
  else NewProd= Put(INput(odd_prod,6.0),Z6.0); /* there is no Zn.n Informat*/
  datalines;
  A12345 <-OK as it is
  B54321 <-OK as it is
  1 <-want to 0 fill
  21 <-want to 0 fill
  321 <-want to 0 fill
  4321 <-want to 0 fill
  54321 <-want to 0 fill
  654321 <-OK as it is
; run;
```

proc print ;run;

If the programmer has access to V9, s/he could use the following code and save a bit of typing.

```
data Combo2;
  infile datalines truncover firstobs=2;
  input @1 odd_name $char6.;
  if anyalpha(odd_name,1) = 1 then Newvar=odd_name; /* anyalpha is a sas 9 function*/
  else NewVar= Put(INput(odd_name,6.0),Z6.0);
  datalines;
  From  To
  A12345 A12345
  B54321 B54321
  1 000001
  21 000021
  321 000321
  4321 004321
  54321 054321
  654321 654321
```

Business rule: If the string starts with a letter, use that string. If not, we must “zero fill”/“zero pad”.

As we convert we will want to use the Z NUMERIC FORMAT to zero pad, and to create a character output.

We must do this in two steps, because there is no Zn.n Informat.

INput takes character input and produces character, or numeric, depending on the type of InFormat applied. Use input to convert odd_prod to numeric and to convert it as if the odd_prod had been 6.0.

Put takes numeric or character input and always produces character. The look of the resulting character variable depends on the format applied. Use Put to convert from numeric to character as if the numeric value had been associated with a Z6.0 format.

Knowledge of SAS Sorting order helps understanding the GE, LE logic in code below

The background on show SAS sorts character data.

The ASCII English-language ASCII sequence sorts from the smallest to the largest in the order shown below,

```
blank ! $ % & ' ( ) * , - . / 0 1 2 3 4 5 6 7 8 9 < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _ a b c d e f g h i j k l m n o p q r s t u v w x y { } ~
```

*/ Background on how SAS sorts character data.*/
The ASCII English-language ASCII sequence sorts from the smallest to the largest in the order shown below,
The logic is as follows. What happens is that the original string, odd_prod, starts as character string with the problems shown above.

We want character output, zero filled and the Z W.w numeric format looks like a good tool to use in this task. The problem is that, we want to apply the Z format (there is no Z Informat in SAS) and the Z format must be applied to a numeric variable. Since our starting point is a character variable, we must first convert the character value to numeric and then convert it back to character with the Z format. Let’s step through the process with the value “54321”. We must nest the use of Put and Input to solve this problem.

Input(odd_prod, 6.0) will take the character value in the variable odd_prod and convert it to numeric, as if it were six digits long with zero digits to the right of the decimal. After this inner function executes, SAS is storing a “working numeric value” of 54321.

Put(“working numeric value”, Z6.0) makes a character string out of the numeric 54321 and creates the character string as if the numeric value had associated with a Z6.0 format – a zero filled format. The result is “054321”.

CONVERTING A CHARACTER DATE TO A SAS DATE EXAMPLE / TRICK

If one is fortunate enough to be sent character dates that happen to have the same structure as an Informat, the character value can be easily converted into a SAS date using the Input function and an Informat. Below are examples of this technique using several commonly encountered date Informats.

```
Data makedate;
  good1 = Input("03/16/99", MMDDYY8.);
  good2A = Input("mar99", monyy5.);
  good2B = Input("mar1999", monyy7.); /*SAS returns the first day of month*/
  good3A = Input("99/03/16", yymmdd10.);
  good3B = Input("1999-03-16", yymmdd10.);
  good4 = Input("199903", yymmN6.);
run;
```

The N, in the Informat used to make the variable called good4, must be used and indicates that you did not separate the year and month values by blanks or by special characters. SAS automatically adds a day value of 01 to the value to make a valid SAS date variable.

The data set MakeDate is shown below.

```
<table>
<thead>
<tr>
<th></th>
<th>good1</th>
<th>good2A</th>
<th>good2B</th>
<th>good3A</th>
<th>good3B</th>
<th>good4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14319</td>
<td>14304</td>
<td>14304</td>
<td>14319</td>
<td>14319</td>
<td>14304</td>
</tr>
</tbody>
</table>
```

CONCLUSION

There are a bewildering number of combinations one can create when coding formats and Informats, Put and Input. We can abstract the following rules describing how Put and Input behave.

Rules for Put function and formats:

**Rule 1:** On format creation:
Character formats can have numeric values on LHS and RHS of the = in the value statement. Numeric formats must have a numeric value on RHS of the = in the value statement.

**Rule 2:** The Put statement accepts numeric or character targets, and numeric or character formats, but always produces a character value.

**Rule 3:** If the target is character we must apply a character format and the most interesting example is /* 4*/. As shown in /* 4*/ , a format can be created with numeric values on the left and right sides (LHS RHS) of the equal sign, but the format must be declared character (have a $ as the first character in the format name).

**Rule 4:** If the target is numeric we must apply a numeric format. The most interesting examples are /* 6*/ and /* 8*/. The format name must not start with a $. When the format was created there must have been a numeric value on the left hand side of the equal sign. When the format was created, the type (character/numeric) value on the right hand side (RHS) of the equal sign is irrelevant. SAS seems to convert num. values on the RHS of the equals to char.

**Rule 5:** The output of a Put will be character. In /* 1*/ the RHS of the equal is character and the result of the Put is character. In /* 3*/ the RHS of the equal is numeric and the result of the Put is still character.

**Rule 6:** Put does NOT help us by performing variable type conversions if we miscode.

**RULE 7:** Avoid passing “outside of mapping range” values to a format. The rules are complex.
Rules for INPUT function and INformats:

Rule 1: Like character formats, character INformats are forgiving. When creating a character INformat, SAS does not care if the values on the right or left hand side of the equal sign are character. When creating a numeric format, the value to the right of the equal sign must be numeric but the value to the left of the equal sign can be character or numeric.

Rule 2: The INput function wants character targets, and either numeric or character INformats. INput produces either character or numeric values.

Rule 3: If we pass a numeric target to an INput function, SAS will convert the target to character and write a note in the log. It will create the variable in the PDV and send it to the output data set, but the variable will not be valued. This is a subtle error;

Rule 4: If the INformat applied is character (the name starts with a $), the function returns a character value.

Rule 5: If the INformat applied is numeric, the function returns a numeric value.

RULE 6: Avoid passing values that are “outside the mapping range” to an INformat. The rules are complex.

CONTACT
Your comments and questions are valued and encouraged. Contact the authors at:

Yanmei Lu  Contractor for Data Means
Hillsborough, NJ, 08844
Email: Yanmeilu@hotmail.com

Russell Lavery  Contractor for Numeric Resources
Ardmore, PA, 19003,
Email: russ.lavery@verizon.net

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.
Program: from_paper_V2.sas
By RL and YL
Usage: Run locally
Modifications:

Section: FORMAT Creation and USE

**Create 4 char and 4 num formats - with odd coding**
*sample character formats - a combinatoric;  
coding/nameing convention: $ is to declare to SAS that this to be a character format
 F stands for format (as opposed to Informat)
 C stands for character format- for human readability
 CC means LHS is Char and RHS is Char
 NC means LHS is Num and RHS is Char
 CN means LHS is Char and RHS is Num
 NN means LHS is Num and RHS is Num;

/* 1*/ value $FCCC "123" ="500";-->NOTE: Format $FCCC has been output.;
/* 2*/ value $FCNC 123 ="500";-->NOTE: Format $FCNC has been output.;
/* 3*/ value $FCCN "123" = 500 ;-->NOTE: Format $FCCN has been output.;
/* 4*/ value $FCNN 123 = 500 ;-->NOTE: Format $FCNN has been output.;

*sample numeric formats - a combinatoric;
*coding/nameing convention: The lack of a $ is to declare this to be a Numeric format
 F stands for format (as opposed to Informat)
 N stands for Numeric format- for human readability
 CC means LHS is Char and RHS is Char
 NC means LHS is Num and RHS is Char
 CN means LHS is Char and RHS is Num
 NN means LHS is Num and RHS is Num;

/* 5*/ value FNCC "123" ="500";
   * ERROR:string '123' not acceptable to numeric format/informat. ;
/* 6*/ value FNNC 123 ="500";-->NOTE: Format FNNC has been output. ;
/* 7*/ value FNF_CN "123" = 500 ;
   *ERROR:string '123' not acceptable to numeric format/informat. ;
/* 8*/ value FNNN 123 = 500 ;-->NOTE: Format FNNN has been output. ;

Data FormatUse;
*the formats map only one value: 123->500;* these formats have very small (only one value) "mapping ranges";
*We will use Put to apply each of the 8 formats, to six values:
* a character 123 and a numeric 123 ;
* a numeric value that is not in the defined range 222
* a character value of the same length as the RHS of the format but not in the defined range "222"
* a character value shorter than the RHS of the format and not in the defined range "9"
* a character value Longer than the RHS of the format and not in the defined range "33333";

**variable names are created to minimize the need to refer back to the code;
*THE VARIABLE NAME IS A CONCATINATION OF:
  1)THE FORMAT APPLIED 2) THE FORMAT NUMBER AND 3)THE VARIABLE TO WHICH IT WAS APPLIED
*FCCC_2_C123 can be decomposed to:
  F stands for: format
  C stands for: character. The format was declared to be character (the name starts with $)
CC stands for: the value statement had character on the left of the = and on the right
---alternatively FCCC can be read as the name of the format we will apply
_1_ is the format number.  Here _1_ corresponds to /* 1*/
all the above was applied, using a put, to a variable that we named
C123 stands for: the test variable C123.  C123 is character and contains the value 123
Test variables are created immediately below and their names have meaning;
CV123="123";  NV123=123;  *Numeric and character version of 123- the formats map 123 -> 500;
Nout="222";  * a numeric variable - not 123 so outside the "mapping range" of the format;
Cout="222";  * a character variable of the SAME length as the "mapping range"
but not 123 so outside the mapping range of the format;
CShortOut="9";  * a character variable of SHORTER length than the "mapping range"
and outside the "mapping range" of the format;
CLongOut="33333";  * a character variable of LONGER length than the "mapping range"
and outside the "mapping range" of the format;
/*Character formats*/
/* apply format 1*/ /*apply1="***"*/
FCCC_1_C123 =put(CV123,$FCCC.);  *VALUE= 500 - TYPE=CHAR3;
FCCC_1_N123 =put(NV123,$FCCC.);  *Variable NV123 has already been defined as numeric.;
FCCC_1_NOutSame =put(Nout,$FCCC.);  *Variable Nout has already been defined as numeric.;
FCCC_1_COutSame =put(Cout,$FCCC.);  *VALUE= 222 - TYPE=CHAR3;
FCCC_1_COutShort =put(CShortOut,$FCCC.);  *VALUE= 9 - TYPE=CHAR3;
FCCC_1_COutLong =put(CLongOut,$FCCC.);  *VALUE= 333 - TYPE=CHAR3;
/* apply format 2*/ /*apply2="***"*/
FCNC_2_C123 =put(CV123,$FCNC.);  *VALUE= 500 - TYPE=CHAR3;
FCNC_2_N123 =put(NV123,$FCNC.);  *Variable NV123 has already been defined as numeric.;
FCNC_2_NOutSame =put(Nout,$FCNC.);  *Variable Nout has already been defined as numeric.;
FCNC_2_COutSame =put(Cout,$FCNC.);  *VALUE= 222 - TYPE=CHAR3;
FCNC_2_COutShort =put(CShortOut,$FCNC.);  *VALUE= 9 - TYPE=CHAR3;
FCNC_2_COutLong =put(CLongOut,$FCNC.);  *VALUE= 333 - TYPE=CHAR3;
/* apply format 3*/ /*apply3="***"*/
FCCN_3_C123 =put(CV123,$FCCN.);  *VALUE= 500 - TYPE=CHAR3;
FCCN_3_N123 =put(NV123,$FCCN.);  *Variable NV123 has already been defined as numeric.;
FCCN_3_NOutSame =put(Nout,$FCCN.);  *Variable Nout has already been defined as numeric.;
FCCN_3_COutSame =put(Cout,$FCCN.);  *VALUE= 222 - TYPE=CHAR3;
FCCN_3_COutShort =put(CShortOut,$FCCN.);  *VALUE= 9 - TYPE=CHAR3;
FCCN_3_COutLong =put(CLongOut,$FCCN.);  *VALUE= 333 - TYPE=CHAR3;
/* apply format 4*/ /*apply4="***"*/
FCNN_4_C123 =put(CV123,$FCNN.);  *VALUE= 500 - TYPE=CHAR3;
FCNN_4_N123 =put(NV123,$FCNN.);  *Variable NV123 has already been defined as numeric.;
FCNN_4_NOutSame =put(Nout,$FCNN.);  *Variable Nout has already been defined as numeric.;
FCNN_4_COutSame =put(Cout,$FCNN.);  *VALUE= 222 - TYPE=CHAR3;
FCNN_4_COutShort =put(CShortOut,$FCNN.);  *VALUE= 9 - TYPE=CHAR3;
FCNN_4_COutLong =put(CLongOut,$FCNN.);  *VALUE= 333 - TYPE=CHAR3;
/* Each of the commented out statements produced two notes in the log
WARNING: Variable Nout has already been defined as numeric.
ERROR 48-59: The format FCNN was not found or could not be loaded.
*/
/*Numeric formats*/
/* apply format 5*/ /* format creation threw an error*/ spacer5="**";

/* apply format 6*/ spacer6="**";

*FNNC_6_C123 = put(CV123,FNNC.);
*ERROR 48-59: The format $FNNC was not found or could not be loaded;
FNNC_6_N123 = put(NV123,FNNC.);
*VALUE= 500 - TYPE=CHAR3;
FNNC_6_NOutSame = put(Nout,FNNC.);
*VALUE= 222 - TYPE=CHAR3;
*FNNC_6_COutSame = put(Cout,FNNC.);
*ERROR 48-59: The format $FNNC was not found or could not be loaded;
*FNNC_6_COutShort = put(CShortOut,FNNC.);
*ERROR 48-59: The format $FNNC was not found or could not be loaded;
*FNNC_6_COutLong = put(ClongOut,FNNC.);
*ERROR 48-59: The format $FNNC was not found or could not be loaded;

/* apply format 7*/ /* format creation threw an error*/ spacer7="**";

*/ apply format 8*/ spacer8="**";

*FNNN_8_C123 = put(CV123,FNNN.);
*ERROR 48-59: The format $FNNN was not found or could not be loaded;
FNNN_8_N123 = put(NV123,FNNN.);
*VALUE= 500 - TYPE=CHAR3;
FNNN_8_OutSame = put(Nout,FNNN.);
*VALUE= 222 - TYPE=CHAR3;
*FNNN_8_COutSame = put(Cout,FNNN.);
*ERROR 48-59: The format $FNNN was not found or could not be loaded;
*FNNN_8_COutShort = put(CShortOut,FNNN.);
*ERROR 48-59: The format $FNNN was not found or could not be loaded;
*FNNN_8_COutLong = put(ClongOut,FNNN.);
*ERROR 48-59: The format $FNNN was not found or could not be loaded;

run;

Proc contents data=FormatUse position;	 proc print data=FormatUse(obs=5); run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>CV123</th>
<th>NV123</th>
<th>Nout</th>
<th>Cout</th>
<th>Out</th>
<th>Out</th>
<th>spacer1</th>
<th>C123</th>
<th>COutSame</th>
<th>Short</th>
<th>COutLong</th>
<th>spacer2</th>
<th>C123</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>123</td>
<td>222</td>
<td>222</td>
<td>9</td>
<td>3333</td>
<td>**</td>
<td>500</td>
<td>222</td>
<td>9</td>
<td>333</td>
<td>**</td>
<td>500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>COOutSame</th>
<th>Short</th>
<th>COutLong</th>
<th>spacer3</th>
<th>C123</th>
<th>COOutSame</th>
<th>Short</th>
<th>COutLong</th>
<th>spacer4</th>
<th>C123</th>
<th>COutSame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>222</td>
<td>9</td>
<td>333</td>
<td>**</td>
<td>500</td>
<td>222</td>
<td>9</td>
<td>333</td>
<td>**</td>
<td>500</td>
<td>222</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>Short</th>
<th>COutLong</th>
<th>spacer5</th>
<th>spacer6</th>
<th>N123</th>
<th>NOutSame</th>
<th>spacer7</th>
<th>spacer8</th>
<th>N123</th>
<th>OutSame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>333</td>
<td>**</td>
<td>**</td>
<td>500</td>
<td>222</td>
<td>**</td>
<td>**</td>
<td>500</td>
<td>222</td>
</tr>
</tbody>
</table>
Section: __ INFORMAT Creation and USE
****************************************************************
Proc format; /*Create 4 char and 4 num INformats - with odd coding*/
*sample character INformats;
*coding/naming convention: $ is to declare to SAS that this to be a character INformat
I stands for INformat (as opposed to format)
C stands for character format- for human readability
/*note INValue is used to make an INformat*/
*/ 1*/ INvalue $ICCC "123"="100"; /*NOTE: Informat $ICCC has been output. ;
*/ 2*/ INvalue $ICNC 123 ="100"; /*NOTE: Informat $ICNC has been output. ;
*/ 3*/ INvalue $ICCN "123" =100; /*NOTE: Informat $ICCN has been output. ;
*/ 4*/ INvalue $ICNN 123 =100; /*NOTE: Informat $ICNN has been output. ;
*sample numeric INformats;
*coding/naming convention: lack of a $ declares to SAS that this to be a numeric INformat
I stands for INformat (as opposed to format)
N stands for character format- for human readability
/*note INValue is used to make an INformat*/
*/ 5*/ *INValue INCC "123"="100"; *ERROR:string '10' not acceptable to numeric. format/informat. ;
*/ 6*/ *INValue INNC 123 ="100"; *ERROR:string '10' not acceptable to numeric. format/informat. ;
*/ 7*/ INValue INCN "123" = 100; /*NOTE: Informat INCN has been output. ;
*/ 8*/ INValue INNN 123 = 100; /*NOTE: Informat INNN has been output. ;
run;

Data INFormatUse;
/*the INformats map only one value: 123->100;* these INformats have very small (only one value) "mapping ranges";
*We will use INput to apply each of the 8 INformats, to six values:
*a character 123 and a numeric 123 ;
*a numeric value that is not in the defined range, 222
*a character value of the same length as the RHS of the INformat but not in the defined range "222"
*a character value shorter than the RHS of the INformat and not in the defined range "9"
*a character value longer than the RHS of the INformat and not in the defined range "33333";
**variable names are created to minimize the need to refer back to the code;
*THE VARIABLE NAME IS A CONCATINATION OF:
1) THE INFORMAT APPLIED 2) THE INFORMAT NUMBER AND 3) THE VARIABLE TO WHICH IT WAS APPLIED
*ICCC_2_C123 can be decomposed to :
I stands for: INformat
C stands for: character. The INformat was declared to be character (the name starts with $)
CC stands for: the value statement had character on the left of the = and on the right
---alternatively ICC can be read as the name of the INformat we will apply
_2_ is the INformat number. Here _2_ corresponds to /* 1*/
_1_ is the INformat number. Here _1_ corresponds to /* 1*/
all the above was applied, using an INPut, to a variable that we named
C123 stands for: the test variable C123. C123 is character and contains the value 123

Test variables are created immediately below and their names have meaning;
CV123="123"; NV123=123; *Numeric and character version of 123 - the INformats map 123 -> 100;
Nout=222;* a numeric variable - not 123 so outside the "mapping range" of the INformat;
Cout="222"; * a character variable of the same length as the "mapping range"
but not 123 so outside the mapping range of the INformat;
CShortOut="9"; * a character variable of SHORTER length than the "mapping range" and outside the "mapping range" of the INFormat;
CLongOut="33333"; * a character variable of LONGER length than the "mapping range" and outside the "mapping range" of the INFormat;

/*Character INFormats*/
/*apply INFormat 1*/ spacer1="***";
   INCC_1_C123 =INPut(CV123,$ICCC.); *VALUE= 100 - TYPE=CHAR3;
   INCC_1_N123 =INPut(NV123,$ICCC.); *Not valued - TYPE=CHAR3- Num to char note in log;
   INCC_1_NOutSame =INPut(Nout,$ICCC.); *Not valued - TYPE=CHAR3- Num to char note in log;
   INCC_1_COutSame =INPut(Cout,$ICCC.); *VALUE= 9 - TYPE=CHAR3;
   INCC_1_COutLong =INPut(CLongOut,$ICCC.); *VALUE= 333 - TYPE=CHAR3 TRUNCATED;
/*apply INFormat 2*/ spacer2="***";
   INCN_2_C123 =INPut(CV123,$ICNC.); *VALUE= 100 - TYPE=CHAR3;
   INCN_2_N123 =INPut(NV123,$ICNC.); *Not valued - TYPE=CHAR3- Num to char note in log;
   INCN_2_NOutSame =INPut(Nout,$ICNC.); *Not valued - TYPE=CHAR3- Num to char note in log;
   INCN_2_COutSame =INPut(Cout,$ICNC.); *VALUE= 9 - TYPE=CHAR3;
   INCN_2_COutShort =INPut(CShortOut,$ICNC.); *VALUE= 9 - TYPE=CHAR3;
   INCN_2_COutLong =INPut(CLongOut,$ICNC.); *VALUE= 333 - TYPE=CHAR3 TRUNCATED;
/*apply INFormat 3*/ spacer3="***";
   ICCN_3_C123 =INPut(CV123,$ICCN.); *VALUE= 100 - TYPE=CHAR3;
   ICCN_3_N123 =INPut(NV123,$ICCN.); *Not valued - TYPE=CHAR3- Num to char note in log;
   ICCN_3_NOutSame =INPut(Nout,$ICCN.); *Not valued - TYPE=CHAR3- Num to char note in log;
   ICCN_3_COutSame =INPut(Cout,$ICCN.); *VALUE= 9 - TYPE=CHAR3;
   ICCN_3_COutShort =INPut(CShortOut,$ICCN.); *VALUE= 9 - TYPE=CHAR3;
   ICCN_3_COutLong =INPut(CLongOut,$ICCN.); *VALUE= 333 - TYPE=CHAR3 TRUNCATED;
/*apply INFormat 4*/ spacer4="***";
   ICNN_4_C123 =INPut(CV123,$ICNN.); *VALUE= 100 - TYPE=CHAR3;
   ICNN_4_N123 =INPut(NV123,$ICNN.); *Not valued - TYPE=CHAR3- Num to char note in log;
   ICNN_4_NOutSame =INPut(Nout,$ICNN.); *Not valued - TYPE=CHAR3- Num to char note in log;
   ICNN_4_COutSame =INPut(Cout,$ICNN.); *VALUE= 9 - TYPE=CHAR3;
   ICNN_4_COutShort =INPut(CShortOut,$ICNN.); *VALUE= 9 - TYPE=CHAR3;
   ICNN_4_COutLong =INPut(CLongOut,$ICNN.); *VALUE= 333 - TYPE=CHAR3 TRUNCATED;
/*Numeric INFormats*/
/*apply INFormat 5*/ INFormat creation threw an error*/ spacer5="***";
/*apply INFormat 6*/ INFormat creation threw an error*/ spacer6="***";
/*apply INFormat 7*/ spacer7="***";
   ICNC_1_C123 =INPut(CV123,INCN.); *VALUE= 100 - TYPE=Num 8;
   ICNC_1_N123 =INPut(NV123,INCN.); *Not valued - TYPE=Num 8- Num to char note in log;
   ICNC_1_NOutSame =INPut(Nout,INCN.); *Not valued - TYPE=Num 8- Num to char note in log;
   ICNC_1_COutSame =INPut(Cout,INCN.); *VALUE= 222 - TYPE=Num 8;
   ICNC_1_COutShort =INPut(CShortOut,INCN.); *VALUE= 9 - TYPE=Num 8;
   ICNC_1_COutLong =INPut(CLongOut,INCN.); *VALUE= 333 - TYPE=Num 8;
/*apply INFormat 8*/ spacer8="***";
   ICNN_2_C123 =INPut(CV123,ICNN.); *VALUE= 100 - TYPE=Num 8;
   ICNN_2_N123 =INPut(NV123,ICNN.); *VALUE= 100 - TYPE=Num 8- Num to char note in log;
   ICNN_2_NOutSame =INPut(Nout,ICNN.); *VALUE= 222 - TYPE=Num 8- Num to char note in log;
   ICNN_2_COutSame =INPut(Cout,ICNN.); *VALUE= 222 - TYPE=Num 8;
   ICNN_2_COutShort =INPut(CShortOut,ICNN.); *VALUE= 9 - TYPE=Num 8;
   ICNN_2_COutLong =INPut(CLongOut,ICNN.); *VALUE= 33333 - TYPE=Num 8 NOT TRUNCATED;
run;
Proc contents data=INFormatUse position; run;
Proc Print data=INFormatUse; run;
Section:__ What sets the length of the output variable LHS or RHS??
****************************************************************
*FORMAT AND PUT;
 Proc format ;
 value $testA "10" = "xxxx"
 "210"= "xxxxYYY";
 run;

data TestingLength;
 HowLongA=put("123456789",$testA.);
 HowLongB=put("10",$testA.);
 HowLongC=put("210",$testA.);
 run;
 Proc print data=TestingLength;
 run;

*FORMAT AND PUT;
 Proc format ;
 value $testB "10" = "xxxx"
 "210"= "xxxxYYY";
 "A longlong LHS"= "xxxxYYY";
 run;

data TestingLength;
 HowLongA=put("123456789",$testB.);
 HowLongB=put("10",$testB.);
 HowLongC=put("210",$testB.);
 run;
 Proc print data=TestingLength;
 run;

*Numeric FORMAT AND PUT;
 Proc format ;
 value testNB 1 = "xx"
 210= "xxxx"
 99999= "xxxxYY";
 run;

data TestingLengthN;
 HowLongA=put(1,testNB.);
 HowLongB=put(210,testNB.);
 HowLongC=put(99999,testNB.);
 HowLongD=put(777777,testNB.);
 HowLongE=put(55,testNB.);run;
 Proc print data=TestingLengthN; run;

*INFORMAT AND INPUT;
*Character informat; /*no $ in name and Char=Numeric */
 Proc format ; /*no $ in name and Char=Numeric */
 invalue $CTestCN "12" ="joy12" /*2 char on left and 5 on right*/
 "123" ="Glad123" /*3 char on left and 7 on right*/
 "1234"="happy1234"; /*4 char on left and 9 on right*/
data TestingLengthV2;
### Code Snippet

```sas
HowLongA=INput("123",$CT_testCN.); *will create a numeric variable;
HowLongB=INput("1234",$CT_testCN.);
HowLongC=INput("12345",$CT_testCN.);
HowLongD=INput("9876543210",$CT_testCN.);
HowLongE=INput("1234987654",$CT_testCN.);
run;
proc contents data=TestingLengthV2;
run;
Proc print data=TestingLengthV2;
run;

Proc format; /*$ in name and Char=Numeric */
invalue $CT_CN "12" = "joy12" /*2 char on left and 5 on right*/
   "123" = "Gladi123" /*3 char on left and 7 on right*/
   "123happy" = "happy8" /*8 char on left and 6 on right*/
data TestingLengthV3;
HowLongA=INput("123",$CT_CN.); *will create a numeric variable;
HowLongB=INput("1234",$CT_CN.);
HowLongC=INput("12345",$CT_CN.);
HowLongD=INput("9876543210",$CT_CN.);
HowLongE=INput("1234987654",$CT_CN.);
run;
proc contents data=TestingLengthV3;
run;
Proc print data=TestingLengthV3;
run;

*numeric Informat;
Proc format; /*no $ in name and Char=Numeric */
invalue NT_CN "1" = -1000 /*2 char on left and 5 on right*/
   "12" = -10000 /*3 char on left and 7 on right*/
   "123" = -100000; /*8 char on left and 6 on right*/
data TestingLengthV4;
HowLongA=INput("1",NT_CN.); *will create a numeric variable;
HowLongB=INput("12",NT_CN.);
HowLongC=INput("123",NT_CN.);
HowLongD=INput("9876543210",NT_CN.);
HowLongE=INput("1234987654",NT_CN.);
run;
proc contents data=TestingLengthV4;
run;
Proc print data=TestingLengthV4;
run;
```

---

This code snippet demonstrates how to use the `INput` function in SAS to create numeric variables from character strings of varying lengths. It also showcases how to format the output of these variables using different numeric formats. The code includes examples of how to handle character strings with different lengths and how to apply specific numeric formats to these strings.