# SYU JOURNAL

VOL.0

# HELLO WORLD

皆さんお疲れ様です。
文系出身で算数がまったく得意でないプログラミング初心者のSYUTENGUです。
つい最近小学生の娘に帯分数と整数の掛け算の間違ったやり方を教えてしまって怒られました💦
そんな記憶もまだ新しいなか、なんと今日は「**プログラミングのすゝめ**」をテーマに
SYU JOUNRAL VOL.0をお送りします。

社会生活のなかで、私たちはいろんな場面で各種の問題に遭遇し判断を求められることがあります。
それらの問題は往々にして算数問題そのものか、数値化によってある程度理性的判断が可能になるものです。
算数または数値化できる問題を解決するには、計算道具の力を借りるのが効率的です。
その道具というのは、電卓や表計算ソフト(e.g.Excel/Google Sheets)等が挙げられます。※
**※その気さえあれば一つ前の時代の算盤や算籌ないし手の指の力を借りても問題ありませんが。**

しかし、上記だけでなくもう一つ私たちの身近にある道具、プログラミングによるアプローチも可能であること
を今回ご紹介したいです。

# 課題提起「消費税計算」

具体的な課題提起で話を展開するのが早いと思います。
社会人の皆さん、仕事のなかで「消費税の計算」はほとんど経験されているかと思います。
[タックスアンサーNo.6351 納付税額の計算のしかた](https://www.nta.go.jp/taxes/shiraberu/taxanswer/shohi/6351.htm)

本質的には四則演算レベルの単純な算数問題で、日常で電卓の税計算機能やExcel関数、(広告まみれの)ウェブアプリ等を用いて処理されている方も多いかと思います。
今回は道具を変えて、自作のプログラムで計算して答えを出してみませんか。

用意が必要なものは二つだけ、制御台と作業台です。
**A制御台**とは最終的にプログラムを実行するため、コンピュータと会話(INPUTとOUTPUT)するユーザーインターフェース(UI)です。
今回はブラウザGoogleChromeに内蔵のChromeDevToolsの「**コンソール**」(CONSOLE)を使います。
**B作業台**とはプログラムの内容(コード)を書く場所です。
今回はWindowsPCの使用を想定して「**メモ帳**」を使います。

# 制御台 CONSOLE

まずは、作るほうではなく使うほうのA制御台から確認しましょう。
`GoogleChromeを開いてファンクションキーF12`を押すと、
ウィンドウの右側にサイドバーの形で表示されるのはChromeDevToolsのUIです。
Windowsのコマンドラインインターフェース(CUI)「cmd」を利用されたことのある方もおられると思いますが、
使い方は基本一緒です。※
**※OSのCUIと比べてローカルファイルへのアクセス権限等がなくブラウザ内でしか使えなくて機能がより限定されたものです。**

以下のような入力(INPUT)をすれば、基本的な算数演算が実行され、答えが次の行(LINE)に出力(OUTPUT)されます。

```
> 1+2 //加減算
< 3
> 2*3 //乗算
< 6
> 2**4 //2の4乗、つまり2*2*2*2
< 16
> 1/3 //除算
< 0.3333333333333333
> 10%3 //剰余
< 1
```

例えば課税売上高11000円の消費税(10%)を計算するには、下記のように入力すれば答えが出力されます。
算数演算子や括弧の優先順位は基本的に小学校で習った算数通りです。

```
> 11000*(10/(10+100))
< 1000
```

はい、これで完成です。一行でも立派なプログラミングです。
これじゃ暗算や電卓、Excel関数、ウェブアプリを使ったほうが早いよとの異見もたぶん出るかと思います。
しかし個人的には他人の作ってくれたアプリよりもネット環境一切いらずにPCや携帯の電卓アプリを起動する手間もなく、いつも開いているブラウザでただ`F12キー`を押すだけで基本的な算数演算ができるほうが比較的便利だと感じます。
さらにブラウザアプリでなくてもWindowsのcmdでさえ同じことができます。

# 作業台 CODE EDITOR

これしかできないと面白くないですね。
プログラミングの面白さの一つはどんどん改善の蓄積ができることです。
課題に戻りますが、軽減税率の適否で税率が異なるのと、毎回数式の入力が非効率だと思う方がきっとおられます。
それなら、税込金額、税率を入力するだけで税額を算出できる自動処理装置(関数)を作りましょう。
コードを一行に纏めるのは別に処理側(PC)にとっては難がないですが、
コードを書く人間にとって読み書きにくいだけなので、
複数行のコードを記述・編集できるよう、B作業台を使います。
「**メモ帳**」等お手元にあるテキストエディタを開いてください。

# 関数と引数 FUNCTIONS & PARAMETERS/ARGUMENTS

まずは空の箱、つまり関数の容器・雛形を書きます。
開いたメモ帳に下記の文字をそのまま記入します。
漢字以外、すべて半角文字を使うことが大事です。基本的に単語と単語の間には半角スペースを入れます。
`「function 」`直後の半角スペースも含めて最初はそのままコピペしたほうが早いと思います。

```
function 消費税計算(税込金額,適用税率){


}
```

ここの`「消費税計算」`は**関数**(FUNCTION)の名前で、呼び出されるときに必要です。
関数名は(一部特例を除いて)基本的に名づけが自由です。
関数名の後の括弧内にある`「税込金額」`と`「適用税率」`は**引数**(PARAMETER)名です。※
引数のところに毎回違う数値を代入できますので、代数方程式のなかのx,yと理解すればいいです。名づけも関数名同様に基本自由です。
後ほどA制御台で`「消費税計算(11000,10)」`と入力し改行すれば、
関数が呼び出され実行されて返り値が処理結果として出力されます。
*※データ受け渡しの立場や関数を見る視点によって仮引数(PARAMETERS)と実引数(ARGUMENTS)と呼び名が分かれますが、気にするほどのことではないです。*

それから、中身の処理コードを`{}`の間に記入します。
さっきA制御台（コンソール）に直接入力した`「11000*(10/(10+100))」`の`「11000」`と`「10」`を具体的数値から引数名に書き換えます。
さらに関数処理を終わらせ処理結果を返すため処理コード行の冒頭に`「return 」`を付け加えます。

```
function 消費税計算(税込金額,適用税率){

    return 税込金額*(適用税率/(適用税率+100))

}
```

A制御台（コンソール）に戻って、コードをコピペして、`改行(ENTER)`します。
`「undefined」`※と出力されますが、気にする必要がないです。
*※未定義、コードをメモリに読み込んだだけで返すもの無しという意味です。*

```
> function 消費税計算(税込金額,適用税率){
      return 税込金額*(適用税率/(適用税率+100))
}
< undefined
```

すると、関数がPCのメモリに読み込まれます。これで実行の用意ができました。
あとは使うだけです。

コンソールで試しに`消費税計算(11000,10)`を入力して`改行(ENTER)`すると、
関数「消費税計算」が呼び出されて`{}`内のコードが逐行処理されます。
最後は結果の`「1000」`をコンソールに返してくれました。

```
> 消費税計算(11000,10)
< 1000
```

毎回関数を呼び出して処理させるにはコンソールに関数を貼り付けてメモリに読み込ませる必要がありません。
しかし**関数の内容が更新されたらもう一度関数全体をコンソールに貼り付けて**メモリに読み込ませる必要があります。※

*※コンソールはあくまでも実験台のような存在で主に処理結果の確認に使われます。本格的にコードを書く場合はまた違うやり方です。*

# 条件構文１ IF

実はこれだけでは小数の税額が答えとして出たとき、引き続き経理処理ができないです。
**端数処理**(ROUNDING)が必要です。
端数処理の方法は複数存在して、どうも会社によって異なる方法が採用されることになっているようです。

端数処理
切り下げ/rounding down(floor)/下取整//1.8->1,-1.8->-2
切り上げ/rounding up(ceil)/上取整//1.8->2,-1.8->-1
切り捨て/rounding toward zero(truncate)/截尾取整 //1.8->1,-1.8->-1
四捨五入/rounding half up/四舍五入//1.5->2,-1.5->-1
五捨六入//五舍六入 //1.5->1,1.6->2
四捨六入//四舍六入五留双 //1.4->1,1.5->1.5,1.6->2

とりあえずよく採用されそうな端数処理方法「切り下げ」、「切り上げ」、「四捨五入」を全部対応できるようにしましょう。
そのために、第三の引数「端数処理」を導入します。引数名の後に`「=0」`を書いておくと、デフォルトで0が自動適用されます。※

*※この引数の値の入力が省略された場合、0が入力されたと見做して処理します。現代的プログラミング言語ならではの便利な仕組みでしょうか。*

また、同じコードの断片を何回も記入するとコードが読みにくくなりますので、関数内で端数処理前の消費税額を名前付けて一時保存します。

```
function 消費税計算(税込金額,適用税率,端数処理=0){
    let 消費税額 = 税込金額*(適用税率/(適用税率+100)) //10
    if(端数処理==45) 消費税額 = Math.round(消費税額) //20
    if(端数処理==0) 消費税額 = Math.floor(消費税額) //30
    if(端数処理==1) 消費税額 = Math.ceil(消費税額) //40
    return 消費税額 //50
}
```

上のように、
3番目の端数処理という引数の与えられた値が、
`「0」`(または無入力)の場合、端数の切り下げ処理が行われます。
`「1」`の場合、端数の切り上げ処理が行われます。
`「45」`の場合、端数の四捨五入処理が行われます。

更新後の関数を`コンソールに貼り付けて改行して`使ってみますと、下記のような結果が得られます。

```
> 消費税計算(10000,10,0)
< 909
> 消費税計算(10000,10,45)
< 909
> 消費税計算(10000,10,1)
< 910
```

# 重要概念　変数と条件構文

なぜ関数内に記入している５行で前述の処理が行われるかと自然に疑問が出るかと思います。
それについて、プログラミングの最重要仕組みとも言えることを二つほど紹介したいです。

# 変数 VARIABLES

一つ目は変数(VARIABLE)です。
変数の概念を簡単に説明すると、あとで使うためにPCのメモリに一時保存される名前付きデータです。
名前をつけて変数を無から作ることを**宣言**(DECLARATION)といい、
それに値を入れて一時保存することを**代入**(ASSIGNMENT)といい、
利用することを**参照**(REFERENCE)といいます。
10行目では、`「消費税額」`という変数が宣言され、`「税込金額*(適用税率/(適用税率+100))」`の計算結果の数値が代入されます。
その変数が20行目～40行目で参照され新しい数値を再代入されたりして50行目で返り値として出力されます。
「変数」なのでもちろん数値の代入は何度行ってもよいものです。つまり変数の値は更新可能です。
これによって書く側(人間)にとってはコードを簡潔に記述できて、
処理する側(PC)にとっては重複計算による処理負担が軽減されます。

とても素晴らしい仕組みです。

```
let 消費税額 = 税込金額*(適用税率/(適用税率+100)) //10
```

# 条件構文 2 CONDITIONAL STATEMENTS

二つ目は条件構文(CONDITIONAL STATEMENTS)です。
本質的にコンピュータの機能は算盤や電卓同様に計算ですので、取捨判断ができないです。
ですので、コードを書く人が事前に各々の条件に満たす場合にすべき異なる処理を指示しておかないといけません。
条件Aに満たす場合イの処理を行い、
条件Bに満たす場合ロの処理を行い、
どちらも当てはまらない場合はハの処理を行うというふうに指示を書きます。

```
if(端数処理==45) 消費税額 = Math.round(消費税額) //20
if(端数処理==0) 消費税額 = Math.floor(消費税額) //30
if(端数処理==1) 消費税額 = Math.ceil(消費税額) //40
```

20行目から40行目までは、自然言語で説明すると、
端数処理の値が`45`に等しい場合は`四捨五入`の端数処理した消費税額の数値を変数「消費税額」に再代入し、
端数処理の値が`0`に等しい場合は`切り下げ`の端数処理した消費税額の数値を変数「消費税額」に再代入し、
端数処理の値が`1`に等しい場合は`切り上げ`の端数処理した消費税額の数値を変数「消費税額」に再代入します。
ここの端数処理の値はスイッチのように処理を分岐させます。※
*※`「45,0,1」`でなく`「3は四捨五入、6は切り下げ、9は切り上げ」`と自由に定義してもまったく問題がありません。*

条件構文がないとプログラミング自体が成り立たないぐらいとてもとても基本で重要な仕組みです。

上記の二つほど重要でないですが、
日常的によく使われる実用機能として補足説明したいのは標準組み込みオブジェクト(STANDARD BUILT-IN OBJECT)の「**Math**」です。
ご覧の通り、計算によく使う一連の定数(CONSTANT)と関数を事前に標準として組み込まれたものです。※
*※定数は変数と違って、宣言と値の代入は同時にする必要がありなおかつ一回きりです。更新不可だからこそ定数と呼ばれます。*

例えば小数点以下を四捨五入したい場合、`Math.round(1.7)`で処理すれば、結果の`「2」`が返されます。※
*※Excel関数のご経験がある方なら自然に理解できるかと思います。*

ほかに例えばMath.random()(引数入力なし)は毎回0以上1未満の小数をランダムに返してくれます。
サイコロやくじ引き機能の実装に使えそうですね。

```
➔  > Math
➔  < Math { … }
➔  //定数
➔  E: 2.718281828459045 //オイラー定数
➔  LN10: 2.302585092994046 //10の自然対数
➔  LN2: 0.6931471805599453 //2の自然対数
➔  LOG10E: 0.4342944819032518 //10を底としたeの対数
➔  LOG2E: 1.4426950408889634 //2を底としたeの対数
➔  PI: 3.141592653589793 //円周と直径の比率
➔  SQRT1_2: 0.7071067811865476 //1/2の平方根
➔  SQRT2: 1.4142135623730951 //2の平方根
➔  //関数
➔  abs: function abs() //絶対値
➔  acos: function acos() //逆余弦 (アークコサイン)
➔  acosh: function acosh() //双曲線逆余弦 (ハイパーボリックアークコサイン)
➔  asin: function asin() //逆正弦 (アークサイン)
➔  asinh: function asinh() //双曲線逆正弦 (ハイパーボリックアークサイン)
➔  atan: function atan() //逆正接(アークタンジェント)
➔  atan2: function atan2() //Math.atan2(y,x)に対して点(0,0)から点(x,y)までの半直線と正のx軸の間の平面上での角度を返す
➔  atanh: function atanh() //双曲線逆正接 (ハイパーボリックアークタンジェント)
➔  cbrt: function cbrt() //立方根
➔  ceil: function ceil() //引数として与えた数値以上の最小の整数を返す
➔  clz32: function clz32() // 32ビットバイナリ表現での先頭の0の個数を返す
➔  cos: function cos() //余弦(コサイン)
➔  cosh: function cosh() //双曲線余弦 (ハイパーボリックコサイン)
➔  exp: function exp() //ex (xは引数、eは自然対数の底)
➔  expm1: function expm1() //ex - 1 (xは引数、eは自然対数の底)
➔  floor: function floor() //引数として与えた数値以下の最大の整数を返す
➔  fround: function fround() //ある Number を表す最も近い32ビット単精度浮動小数点数を返す
➔  hypot: function hypot() //各引数の二乗の合計値の平方根を返す
➔  imul: function imul() //2つの引数でC言語風の32ビット乗算を行った結果を返す
➔  log: function log() //(e を底とした) 数値の自然対数を返す
➔  log10: function log10() //数値の10を底とした対数を返す
➔  log1p: function log1p() //1 + 数値の(eを底とする)自然対数を返す
➔  log2: function log2() //数値の2を底とした対数を返す
➔  max: function max() //入力引数として与えられた0個以上の数値のうち最大の数を返す
➔  min: function min() //引数で渡されたもののうち最小の値を返す
➔  pow: function pow() //底base ** 冪指数exponent = 冪power
➔  random: function random() //0以上1未満(0は含むが、1は含まない)の範囲で浮動小数点の擬似乱数を返す
➔  round: function round() //引数として与えた数を四捨五入して、もっとも近似の整数を返す
➔  sign: function sign() //引数として渡された数値の符号が正か負かを表す +/- 1 を返す
➔  sin: function sin() //正弦 (サイン)
➔  sinh: function sinh() //双曲線正弦 (ハイパーボリックサイン)
➔  sqrt: function sqrt() //平方根
➔  tan: function tan() //正接(タンジェント)
➔  tanh: function tanh() //双曲線正接(ハイパーボリックタンジェント)
➔  trunc: function trunc() //引数として与えた数の小数部の桁を取り除くことによって整数部を返す
```

# 条件構文 3 IF ELSE

これで日本国内現行の消費税率による消費税額の計算に対応できるようですね。
でもこの関数をもう一段進化させたいです。
中国の増値税(計算方法は日本の消費税と同じ)にも対応できたらと思います。
この場合は、日本の消費税にはあり得ないこと、いわゆる小数点以下n位までの端数処理です。
中国元の場合、小数点以下２位の「分」までの端数処理が求められます。
これに対応するためには、第四の引数「小数位」を導入します。

```
function 消費税計算(税込金額,適用税率,端数処理=0,小数位=0){
        const 端数処理精度 = 10**(小数位) //5
        let 消費税額 = 税込金額*(適用税率/(適用税率+100)) //10
        if(端数処理==45) 消費税額 = Math.round(消費税額*端数処理精度)/端数処理精度 //20
        //if(端数処理==0) 消費税額 = Math.floor(消費税額*端数処理精度)/端数処理精度 //30
        else if(端数処理==1) 消費税額 = Math.ceil(消費税額*端数処理精度)/端数処理精度 //40
        else 消費税額    = Math.floor(消費税額*端数処理精度)/端数処理精度//45
        return 消費税額 //50
}
```

結果としてどんな税率を適用したときでも端数処理をさせたいので、
まず30行目の前に`「//」`をつけて**コメントアウト**して、※
*※コメントはコードの注釈として情報そのものは文面に残しますが、処理時にはとばされて処理されない仕組みです。*

代わりに45行目のコードを記入します。
さらに40行目の冒頭に`「else 」`を追記します。
これで20行目～45行目は下記のように条件分岐処理されます。
端数処理という引数の入力が`「45」`の場合、`四捨五入`の端数処理が行われます。。*※20行目*
そうじゃなくて端数処理が`「1」`の場合、`切り上げ`の端数処理が行われます。*※40行目*
上記二つの場合以外、すべて`切り下げ`の端数処理が行われます。*※45行目*

```
> 消費税計算(10000,13,0,2)
< 1150.44
> 消費税計算(10000,6,45,2)
< 566.04
> 消費税計算(10000,6,45,-1) //しようと思えば一の位での四捨五入端数処理も可能です
< 570
```

詳しくは分かりませんが、中国の増値税はEU加盟国の範囲内で使用されている増値税と類似の仕組みが採用されているそうです。※
[EU加盟国のVAT税率表](https://taxation-customs.ec.europa.eu/system/files/2021-06/vat_rates_en.pdf#page=3)

もしかしてこの関数による増値税額または消費税額の計算はより広い範囲で適用できるかもしれません。

また、しようと思えばより多様な端数処理方法への対応や税抜金額の同時出力、消費税と地方消費税の分別計算等、さまざまな新しい機能を後から追加できます。

# 命名 NAMING

最後に、ユーザー環境を考慮してプログラミングの世界ではあまり漢字や仮名等の全角文字を関数名や変数名として使用していません。※

**※でも最近開発段階ならあえて漢字仮名を使ったほうが読みやすいと思うようなこともあります。好き嫌いの問題ですね。**

課題の関数に登場した名前を全部アルファベットに置き換えると、以下のようなコードとなります。

```
function calcVat(included, rate, round = 0, digits = 0) {
    const PREC = 10 ** (digits)
    let vat = included * (rate / (rate + 100))
    if (round == 45) vat = Math.round(vat * PREC) / PREC
    else if (round == 1) vat = Math.ceil(vat * PREC) / PREC
    else vat = Math.floor(vat * PREC) / PREC
    return vat
}
```

# 結びに

いかがでしょうか。消費税計算という単純な算数問題を事例にプログラミングの基礎知識のごく一部を駆け足で紹介しました。
冒頭で述べたように、算数または数値化できる問題の解決の助けになる道具はほかにもたくさん存在します。
でもプログラミングは電卓アプリやExcel関数等と比べるとより**きめ細かいチューニング**ができて、
**意のままのデータ自動処理**装置が実装しやすいことに少しでも同感できたでしょうか。

もとより少々抽象的な分野ですし、限られた紙幅でプログラミングの面白さを語り尽くすことは難しいですが、
今後機会があれば、また他の事例を通じてプログラミングの異なる一面をご紹介できればと思います。

ゼロからプログラミングの勉強、特に独学の場合は
(少なくとも私のような算数得意でない人にとっては)決してコストが低いとは言えません。※
ただ、もし新しいことの勉強がお好きな方でしたら、ぜひチャレンジしてみてください。
方法論と世界観を更新できるぐらい面白くて実用性も高いとても「お得な」知識分野ですから。

*※逆に算数が嫌いだからこそプログラミングを習得して面倒な計算は全部コンピュータに任せようという勉強の動機付けもできるかと思います。*

# REFERENCE

[MDN Web Docs](https://developer.mozilla.org/en-US/)
[W3schools Online Web Tutorials](https://www.w3schools.com/)

# 補足 ダイス機能の実装

文中に紹介したJavascript標準組み込みオブジェクトMathを使ってダイス機能を下記のように実装できます。

```
//ālea iacta est
function dice(num = 1, side = 6) {
    return Array.from({ length: num }).map(e => Math.floor(Math.random() * side + 1))
}
```

引数はダイスの個数と面数です。デフォルト（無入力）で1個の6面ダイスが引数の値として与えられます。
下線部以外のコードはダイス個数分の結果を配列で返す仕組みで、とりあえずそのままコピペしても大丈夫です。
下線部のコードは計算方法です。( 0以上1未満の乱数 × ダイスの面数 ) + 1 を小数点以下切り下げ処理します。
例えば６面ダイスの場合、
0以上1未満の乱数 × 6の結果は0以上６未満です。①
①の結果に1を足すと、結果が1以上7未満です。②
②の結果を小数点以下切り下げ処理すると、整数1,2,3,4,5,6となります。

```
> dice() //6面ダイス1個
< [3]
> dice(2,8) //8面ダイス2個
< [1,7]
> dice(10,20) //20面ダイス10個
< [ 14, 19, 13, 9, 17, 20, 2, 16, 1, 12 ]
```

実物の20面ダイスを作るのはけっこう手間がかかりそうですが、プログラムだと一行程度ですね。