	JavaScript	JAVA
論理値(True,False)	リテラル true , false が使える	リテラル true , false が使える
論理積，論理和	条件1 **&&** 条件2 , 条件1 || 条件2 ＜短絡評価型＞	条件1 **&&** 条件2 , 条件1 || 条件2 ＜短絡評価型＞
2次元配列 の定数定義	var 配列 = [[0,1, ...], [0,1,...], ...]; 角カッコで括る。 xxx = 配列[0][1];	int 配列[][] = { { 0,1,...}, {0,1,...}, ... }; 波カッコで括る。 xxx = 配列[0][1];
連想配列 ハッシュ 辞書オブジェクト	var Result = { "ErrId" : 0 , ... } 波カッコで括る。 要素は "キー" : 値 xxx = Result.ErrId	関数の返却値にはクラスを使用する
関数(メソッド)の宣言 および 引数の未定義判定	function 関数名(TelCode) { 引数に型宣言なし 引数の未定義判定 [=記号は３連(厳密等価演算子)] if (TelCode === undefined) { // 未定義 } else { // 指定有り }	public static クラス 　　関数名(String TelCode) { 引数に型宣言あり 引数未定義はコンパイルエラー
変数の「型」判定 (引数の内容判定)	if (typeof TelCode == "string") { } else if (typeof TelCode == "number") {	引数宣言で「型指定」するので意識する必要なし
文字列の長さ	文字列.length	文字列.length()
型変換 文字列⇒数値 数値⇒文字列	数値 = Number(文字列); 文字列 = String(数値);	数値 = Integer.parseInt(文字列); 文字列 = String.valueOf(数値);
文字列の 大/小文字 変換	文字列.toUpperCase() 文字列.toLowerCase()	文字列.toUpperCase() 文字列.toLowerCase()
文字列の抽出	文字列.**substr**(開始位置, 文字数) 文字位置はゼロオリジン 文字数を省略すると最後まで切り出す ２～５文字目⇒(1, 4)で４文字	文字列.**substring**(開始位置, 終了位置) 文字位置はゼロオリジン 終了位置を省略すると最後まで切り出す 終了位置は切り出し範囲に含まない (終了位置の１文字前までが抽出対象) ２～５文字目⇒(1, 5) で４文字 終了位置の５は６文字目を指す

	JavaScript	JAVA
正規表現による 文字列置換 (全置換)	文字列.**replace**(/正規表現/**g**, 置換文字列) 全置換は gオプションを付ける ハイフン・括弧の除去(空文字へ置換)の例 TelCode.replace(/-|¥(|¥)/g,"")	文字列.**replaceAll**("正規表現", 置換文字列) 全置換は replaceAl ハイフン・括弧の除去(空文字へ置換)の例 TelCode.replaceAll("-|¥¥(|¥¥)","") (注) エスケープは ¥¥ (¥ (バックスラッシュ ＼))が２個
正規表現 ('0'始まり 10桁以上の 数字の例)	var Pattern10 = /^0¥d{9,}$/; / で括って、そのまま変数へ代入 (不一致) if (TelCode.match(Pattern10) == null) { (一致) if (TelCode.match(Pattern10) != null) {	Pattern Pattern10 = Pattern.compile("^0[0-9]{9,}$"); " で括って、コンパイルしたものを変数に代入 (注) 数字指定に ¥d が使えないので [0-9] (不一致) if (Pattern10.matcher(TelCode).matches() == false) { (一致) if (Pattern10.matcher(TelCode).matches() == true) {
条件分岐 (if ～ elseif ～ else ～)	elseブロック内が１個の ifブロック だけならば else ブロックの { } を 省略して **elseif 風に書ける。** if (条件1) { 処理1; } else if (条件2) { 処理2; } else if (条件3) { 処理3; } else { 処理4; }	elseブロック内が１個の ifブロック だけならば else ブロックの { } を 省略して **elseif 風に書ける。** if (条件1) { 処理1; } else if (条件2) { 処理2; } else if (条件3) { 処理3; } else { 処理4; }
条件分岐 (switch)	switch(条件項目) { case 値1: 処理1; break; case 値2: 処理2; break; default; 処理3; }	switch(条件項目) { case 値1: 処理1; break; case 値2: 処理2; break; default; 処理3; }
ループ (for)	for (カウンタ初期値; 終了条件; カウンタ増分値) { １～５ の例 for (cnt = 1; cnt <= 5; cnt++) { 処理; }	for (カウンタ初期値; 終了条件; カウンタ増分値) { １～５ の例 for (cnt = 1; cnt <= 5; cnt++) { 処理; }
文字列 同士の比較	文字列の[値]同士の比較なら == で良い。 文字列変数の[型 & 値]同士の比較は === にする。	文字列型の「変数」同士の比較は == ではなく **equals** メソッドで行なう × if (TelCode == ResultFormat.TelEdit) { ○ if (TelCode.**equals**(ResultFormat.TelEdit)) {

	JavaScript	JAVA
文字列 同士の連結	文字列連結の演算子は プラス （ + ） "ABC" + "abc" ⇒ "ABCabc"	文字列連結の演算子は プラス （ + ） "ABC" + "abc" ⇒ "ABCabc"
標準入力, 標準出力		import java.util.Scanner; Scanner scan = new Scanner(System.in); String Ans; Ans = scan.nextLine(); //改行コードは取り除かれる scan.close(); System.out.print("Test");
その他		正規表現の利用には、下記のインポートが必要 import java.util.regex.Pattern;

	PHP	Perl
論理値(True,False)	リテラル true , false が使える (TRUE , FALSE でも可)	論理値(true/false)の**予約語(リテラル)は無い**
論理積，論理和	条件1 **&&** 条件2 , 条件1 **||** 条件2 ＜短絡評価型＞	条件1 **&&** 条件2 , 条件1 **||** 条件2 ＜短絡評価型＞
2次元配列 の定数定義	$配列 = [[0,1, ...], [0,1,...], ...]; 角カッコで括る。 xxx = $配列[0][1];	my @配列 = ([0,1, ...], [0,1,...], ...); 大外は丸カッコ、中の配列は角カッコで括る。 xxx = $配列[0][1]; 配列定義, 配列全体を指定 ⇒ **@** 記号 他、配列要素の参照/代入時 ⇒ **$** 記号
連想配列 ハッシュ 辞書オブジェクト	$Result = ["ErrId" => 0 , ...] 角カッコで括る。 要素は "キー" => 値 xxx = $Result["ErrId"]	my %Result = (ErrId => 0 , ...) 丸カッコで括る。 要素は キー => 値 xxx = $Result{ ErrId } 配列定義・関数返却値の受け取り⇒ **%** 記号 他、配列要素の参照/代入時 ⇒ **$** 記号
関数(メソッド)の宣言 および 引数の未定義判定	function 関数名(string $TelCode = "") { 引数に型宣言あり 但し,**int string** は PHP7 以降で指定可能 引数の未定義判定は 省略時の既定値を指定する事で代替	sub 関数名 { my ($TelCode) = @_; 引数に型宣言なし 引数の未定義判定(defined は定義済みならTrue) if (!defined($TelCode)) { # 未定義 } else { # 指定有り }
変数の「型」判定 (引数の内容判定)	引数宣言の「int, string 型指定」が**PHP5では 指定不可**なので型判定が必要。 PHP5 では約20億を超える(32bit)と **int** ではなく **float** になる。 ※ 電話番号を数値にすると int を超える。 if (is_string($TelCode)) { } elseif (is_float($TelCode) || is_int($TelCode)) {	判定方法なし
文字列の長さ	mb_strlen(文字列)	length(文字列)
型変換 文字列⇒数値 数値⇒文字列	数値 = (int)文字列; 文字列 = (string)数値;	自動型変換を行なう言語なので、 算術演算/文字列連結で対応する。 数値 = 文字列 * 1; # 1 で乗算 文字列 = 数値 . ""; # 空文字の連結
文字列の 大/小文字 変換	mb_strtoupper(文字列) mb_strtolower(文字列)	uc(文字列) lc(文字列)
文字列の抽出	mb_substr(文字列, 開始位置, 文字数) 文字位置はゼロオリジン 文字数を省略すると最後まで切り出す ２～５文字目⇒(1, 4)で４文字	substr(文字列, 開始位置, 文字数) 文字位置はゼロオリジン 文字数を省略すると最後まで切り出す ２～５文字目⇒(1, 4)で４文字

	PHP	Perl
正規表現による 文字列置換 (全置換)	preg_replace('/正規表現/', 置換文字列, 文字列) 全置換は 正規表現内の置換対処文字列を括弧で括る /(-)|(¥()|(¥))/ ... ハイフン/括弧の全置換 ハイフン・括弧の除去(空文字へ置換)の例 preg_replace('/(-)|(¥()|(¥))/', "", $TelCode)	文字列 =~ s/正規表現/置換文字列/g 上記の書き方で、正規表現に一致する 文字列の部分を置換文字列に置き換える 全置換は gオプションを付ける 正規表現・置換文字列は " で括る必要はないので ⇒ 空文字に置換は下記になる 文字列 =~ s/正規表現//g
正規表現 ('0'始まり 10桁以上の 数字の例)	$Pattern10 = "/^0¥d{9,}$/"; / で括り、更に " で括って、変数へ代入 (不一致) if (preg_match($Pattern10, $TelCode) == false) { (一致) if (preg_match($Pattern10, $TelCode) == true) {	my $Pattern10 = qr/^0¥d{9,}$/; / で括って、qr演算子で コンパイルしたものを変数に代入 (不一致) if ($TelCode !~ /$Pattern10/) { (一致) if ($TelCode =~ /$Pattern10/) {
条件分岐 (if ～ elseif ～ else ～)	if (条件1) { 処理1; } elseif (条件2) { 処理2; } elseif (条件3) { 処理3; } else { 処理4; }	if (条件1) { 処理1; } elsif (条件2) { 処理2; } elsif (条件3) { 処理3; } else { 処理4; } elseif ではなく elsif である事に注意
条件分岐 (switch)	switch(条件項目) { case 値1: 処理1; break; case 値2: 処理2; break; default; 処理3; }	な し
ループ (for)	for (カウンタ初期値; 終了条件; カウンタ増分値) { １～５ の例 for ($cnt = 1; $cnt <= 5; $cnt++) { 処理; }	for (カウンタ初期値; 終了条件; カウンタ増分値) { １～５ の例 for ($cnt = 1; $cnt <= 5; $cnt++) { 処理; } for ループを抜ける命令が break ではなく last である事に注意
文字列 同士の比較	文字列の[値]同士の比較なら == でも良い。 但し、値の**先頭に数字が有る**場合は注意が必要。 (比較対象が『数値』の場合に、先頭の 数字部分のみを数値変換した結果との比較になる) 確実な比較をする為には、 [型 & 値]同士の比較になる === にする。 電話番号(数字)の比較処理なので === で行なう。	文字列型の「変数」同士の比較は == ではなく eq 演算子で行なう × if (TelCode == ResultFormat.TelEdit) { ○ if (TelCode eq ResultFormat.TelEdit) { gt (>), ge (>=), lt (<), le (<=), ne (!=)

	PHP	Perl
文字列 同士の連結	文字列連結の演算子は ピリオド（ . ） "ABC" . "abc" ⇒ "ABCabc"	文字列連結の演算子は ピリオド（ . ） "ABC" . "abc" ⇒ "ABCabc"
標準入力, 標準出力	$Ans = trim(fgets(STDIN)); trim で改行コードを取り除く echo "Test";	$Ans = <STDIN>; chomp($Ans); # 改行コードを取り除く binmode STDOUT, ':encoding(UTF-8)'; print STDOUT "Test";
その他	グローバル変数はユーザー定義関数の中では 参照できない。参照可能にする為には、ユーザー 関数内で global 宣言が必要。 global $AreaCodeArray; // 番号翻訳テーブル	

	Python	Ruby
論理値(True,False)	リテラル True , False が使える	擬似変数 true , false が使える 他の言語と異なり 0 は true になる
論理積，論理和	条件1 and 条件2 , 条件1 or 条件2 ＜短絡評価型＞	条件1 && 条件2 , 条件1 || 条件2 ＜短絡評価型＞
2次元配列 の定数定義	配列 = [[0,1, ...], [0,1,...], ...] 角カッコで括る。 xxx = 配列[0][1]	配列 = [[0,1, ...], [0,1,...], ...] 角カッコで括る。 xxx = 配列[0][1]
連想配列 ハッシュ 辞書オブジェクト	result = { 'ErrId' : 0 , ... } 波カッコで括る。 要素は 'キー' : 値 xxx = result['ErrId']	result = { ErrId: 0 , ... } 波カッコで括る。 要素は シンボル: 値 xxx = result[:ErrId]
関数(メソッド)の宣言 および 引数の未定義判定	def 関数名(tel_code = ''): 引数に型宣言なし 引数の未定義判定は 省略時の既定値を指定する事で代替	def self.関数名(tel_code = '') 引数に型宣言なし 引数の未定義判定は 省略時の既定値を指定する事で代替
変数の「型」判定 (引数の内容判定)	if (type(tel_code) is str): elif (type(tel_code) is int):	if (tel_code.is_a?(String)) elsif (tel_code.is_a?(Integer))
文字列の長さ	len(文字列)	文字列.length
型変換 文字列⇒数値 数値⇒文字列	数値 = int(文字列) 文字列 = str(数値)	数値 = 文字列.to_i 文字列 = 数値.to_s
文字列の 大/小文字 変換	文字列.upper() 文字列.lower()	文字列.upcase 文字列.downcase
文字列の抽出	文字列[開始位置, 終了位置] 文字位置はゼロオリジン 終了位置を省略すると最後まで切り出す 終了位置は切り出し範囲に含まない (終了位置の１文字前までが抽出対象) ２～５文字目⇒(1, 5) で４文字 終了位置の５は６文字目を指す	文字列[開始位置, 文字数] 文字位置はゼロオリジン **文字数は省略できない** (範囲外までの文字数を指定すると nil が返る) ２～５文字目⇒(1, 4)で４文字

	Python	Ruby
正規表現による 文字列置換 (全置換)	re.sub(r'正規表現', 　　置換先文字列, 文字列 [, 置換回数]) 全置換は 置換回数を省略	文字列.gsub(/正規表現/, 置換先文字列) gsub で全置換 sub は最初の１個のみ置換 gsub! にすると［文字列］自体も更新される
正規表現 ('0'始まり 10桁以上の 数字の例)	pattern10 = re.compile(r'^0¥d{9,}$') ' で括り r を前置して コンパイルしたものを変数に代入 (不一致) if (pattern10.fullmatch(tel_code) == None): (一致) if (pattern10.fullmatch(tel_code) != None):	pattern10 = /^0¥d{9,}$/ / で括って、そのまま変数へ代入 (不一致) if (tel_code !~ pattern10) (一致) if (tel_code =~ pattern10)
条件分岐 (if ～ elseif ～ else ～)	if (条件1): 　　処理1 elif (条件2): 　　処理2 elif (条件3): 　　処理3 else: 　　処理4 (注) else ブロックの処理が無い場合は 　代わりに pass を記述する elseif ではなく elif である事に注意	if (条件1) 　　処理1 elsif (条件2) 　　処理2 elsif (条件3) 　　処理3 else 　　処理4 end elseif ではなく elsif である事に注意
条件分岐 (switch)	な　し	case (条件項目) 　when 値1 　　処理1 　when 値2 　　処理2 　else 　　処理3 end 他言語の break に類するものは必要なし
ループ (for)	for カウンタ in イテラブルオブジェクト: 他言語での [$cnt = 1; $cnt <= 5; $cnt++]の部分を rangeオブジェクトで指定する。 range(start, stop, step)　　step を省略すると増分１ [start ≦ カウンタ ＜ stop]である事に注意 １～５ の例 for cnt in range(1, 6): 　処理	for カウンタ in 範囲オブジェクト 範囲オブジェクトは ..演算子で定義する start .. stop [start ≦ カウンタ ≦ stop]になります １～５ の例 for cnt in 1 .. 5 　処理 end
文字列 同士の比較	== 演算子による比較で良い。 但し、Python２系で日本語文字列を含む場合は str型 vs unicode型で注意が必要。 (尚、TelephoneCheck では日本語文字列は扱っていない)	== 演算子(メソッド)による比較で良い。 尚、同じ機能で eql? メソッドもある。 [文字列1 == 文字列2] 　　イコール [文字列1.eql? 文字列2]

	Python	Ruby
文字列 同士の連結	文字列連結の演算子は プラス （ + ） "ABC" + "abc" ⇒ "ABCabc" 文字列型に数値型は連結できない。 (str 関数で型変換が必要)	文字列連結の演算子は プラス （ + ） "ABC" + "abc" ⇒ "ABCabc" 文字列型に数値型は連結できない。 (.to_s メソッドで型変換が必要)
標準入力, 標準出力	ans = input('Test? (Y/N) > ') プロンプトメッセージが指定可能 改行コードは取り除かれる print ('Test')	ans = gets.chomp [.chomp]指定で改行を取り除いてくれる print ("Test")
その他		グローバル変数の名前には接頭辞($)が必要。 参照時も $ 付きの名前で参照する。 $area_code_array = [

	VBA/VB6	VB.net
論理値(True,False)	リテラル True , False が使える	リテラル True , False が使える
論理積，論理和	条件1 And 条件2 , 条件1 Or 条件2 ＜完全評価型＞	条件1 And 条件2 , 条件1 Or 条件2 ＜完全評価型＞ 条件1 AndAlso 条件2 , 条件1 OrElse 条件2 ＜短絡評価型＞
2次元配列 の定数定義	配列の定数定義は出来ない。 Array関数で実行時に作成する。 Dim 配列 As Variant 配列 = Array(Array(0,1,…),Array(0,1,…),…) xxx = 配列(0)(1) セミコロン/カンマ区切りのテキストデータとして 定数定義し、Split関数で２次元配列に分解する。 (行継続文字(_)の連続使用制限もあるので注意)	Dim 配列(,) = { { 0,1,...}, {0,1,...}, ... } 波カッコで括る。 複数行に渡って記述する場合は行継続文字(_)を 記述する(行継続文字の連続使用制限なし) xxx = 配列(0 , 1)
連想配列 ハッシュ 辞書オブジェクト	関数の返却値にはユーザー定義型を使用する	関数の返却値には構造体を使用する
関数(メソッド)の宣言 および 引数の未定義判定	Public Function 関数名 (ByVal TelCode As String) As ～ 引数に型宣言あり 引数未定義はコンパイルエラー	Public Function 関数名 (ByVal TelCode As String) As ～ 引数に型宣言あり 引数未定義はコンパイルエラー
変数の「型」判定 (引数の内容判定)	引数宣言で「型指定」するので意識する必要なし	引数宣言で「型指定」するので意識する必要なし
文字列の長さ	Len(文字列)	文字列.Length
型変換 文字列⇒数値 数値⇒文字列	数値 = CInt(文字列) or Val(文字列) 文字列 = CStr(数値)	数値 = CInt(文字列) or Val(文字列) 文字列 = CStr(数値)
文字列の 大/小文字 変換	UCase(文字列) LCase(文字列)	文字列.ToUpper 文字列.ToLower
文字列の抽出	Mid(文字列, 開始位置, 文字数) 文字位置は１オリジン 文字数を省略すると最後まで切り出す ２～５文字目⇒(2, 4)で４文字 Left(文字列, 文字数) , Right(文字列, 文字数)	Mid(文字列, 開始位置, 文字数) 文字位置は１オリジン 文字数を省略すると最後まで切り出す ２～５文字目⇒(2, 4)で４文字 Left(文字列, 文字数) , Right(文字列, 文字数)

	VBA/VB6	VB.net
正規表現による 文字列置換 (全置換)	Replace(文字列, 対象文字列, 置換文字列) 正規表現は使わずにひとつずつ置換する ハイフン・括弧の除去(空文字へ置換)の例 TelNumber = Replace(TelCode, "-", "") TelNumber = Replace(TelNumber, "(", "") TelNumber = Replace(TelNumber, ")", "")	文字列.Replace(対象文字列, 置換文字列) 正規表現は使わずにひとつずつ置換する ハイフン・括弧の除去(空文字へ置換)の例 TelNumber = TelCode.Replace("-"c, ""c) TelNumber = TelNumber.Replace("("c, ""c) TelNumber = TelNumber.Replace(")"c, ""c) 一文字置換の場合には char型で指定する
正規表現 ('0'始まり 10桁以上の 数字の例)	VBScript.RegExp を使用する Dim RegExp As Object Set RegExp = CreateObject("VBScript.RegExp") With RegExp .Pattern = "^0¥d{9,}$" .IgnoreCase = False .Global = True End With (不一致) If (RegExp.Test(TelCode) = False) Then (一致) If (RegExp.Test(TelCode) = True) Then	Dim Pattern10 = As New System.Text.RegularExpressions. Regex("^0¥d{9,}$") " で括って、コンパイルしたものを変数に代入 (不一致) If (Pattern10.IsMatch(TelCode) = False) Then (一致) If (Pattern10.IsMatch(TelCode) = True) Then
条件分岐 (if ～ elseif ～ else ～)	If (条件1) Then 処理1 ElseIf (条件2) Then 処理2 ElseIf (条件3) Then 処理3 Else 処理4 End If	If (条件1) Then 処理1 ElseIf (条件2) Then 処理2 ElseIf (条件3) Then 処理3 Else 処理4 End If
条件分岐 (switch)	Select Case (条件項目) Case 値1 処理1 Case 値2 処理2; Case Else 処理3 End Select	Select Case (条件項目) Case 値1 処理1 Case 値2 処理2; Case Else 処理3 End Select
ループ (for)	For カウンタ = 初期値 To 終了値 Step 増分 １～５ の例 For cnt = 1 To 5 Step 1 処理 End For For ループを抜ける命令は Exit For [終了条件]指定は Do While(継続条件) ～ Loop	For カウンタ = 初期値 To 終了値 Step 増分 １～５ の例 For cnt = 1 To 5 Step 1 処理 End For For ループを抜ける命令は Exit For [終了条件]指定は While(継続条件) ～ End While
文字列 同士の比較	= 演算子による比較で良い。	= 演算子による比較で良い。

	VBA/VB6	VB.net
文字列 同士の連結	文字列連結の演算子は アンパサンド （ & ） "ABC" & "abc" ⇒ "ABCabc"	文字列連結の演算子は アンパサンド （ & ） "ABC" & "abc" ⇒ "ABCabc"
標準入力, 標準出力	イミディエイトウィンドウに表示 Debug.Print("テスト")	Ans = Console.ReadLine() （改行コードは取り除かれる） コンソールウィンドウに表示 Console.WriteLine("テスト")
その他		関数の戻り値で複数の型(Integer or String 等)を返す事が出来ない(Variant/Any 型が無い)。 引数の数を変えて**オーバーロード**定義すれば可能。 PhoneNumber関数で対応