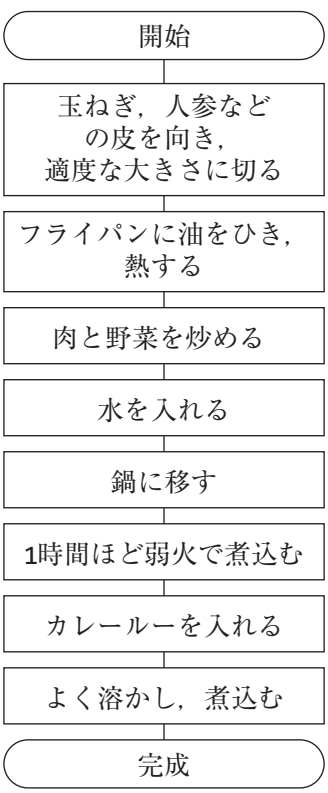
アルゴリズムのきほん

アルゴリズムとは

アルゴリズムは，「計算の手順」や「必要な処理結果を得るための手順」を意味します。**アルゴリズム**というと，難しい計算をイメージするかもしれませんが，直接的な算術計算だけを意味するものではありません。たとえば，「東京から大阪へ行くための経路を求める」といった問題を解くための手順もアルゴリズムとよんでよいのです。

例えば，カレーなどの料理を作るレシピなども，一つのアルゴリズムと捉えることができます。



また，ある問題の答えを得るための手順，すなわちアルゴリズムは複数存在します。同じ結果を得る場合でも，アルゴリズムが異なると，処理の効率が異なってくることがあります。同じ結果が得られるのであれば，より少ない(より速い)処理となるようなアルゴリズムのほうがよいといえます。

変数

変数は，データを格納する“箱”のようなものと考えてください。プログラムはこの変数を対象に処理を行うので，データをプログラムで処理する場合には，処理に先立ちデータを変数に格納しなければなりません。

①変数には名前(変数名)がついている

変数には，名前が必要です。これを**変数名**とよびます。この変数名によって，どの変数であるか判別されるわけです。

②変数の中には最初は何が入っているかわからない

プログラムの内部で変数Aを宣言すると，コンピュータシステムはデータを格納する領域をメモリ上に確保します。このとき，**確保した領域にどのようなデータが格納されていたかはわかりません**。

③変数には，値を入れることができる(代入)

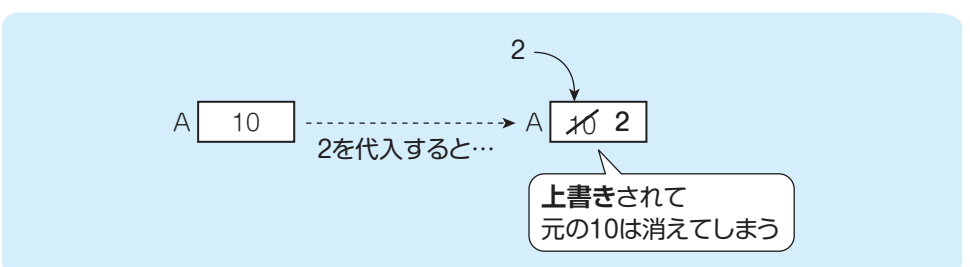
変数に値を入れることを**代入**とよびます。たとえば，変数Aに10を代入するには，

A ← 10　　(あるいは 10 → A)

のように記述します。なお，変数に**初期値**(最初の値)を代入することを，**初期化**といいます。

④変数に格納できる値は一つだけである

変数には，同時に一つの値しか格納できません。たとえば，あらかじめ変数Aに10が格納されている場合に，その変数Aに2を代入する場合を考えてみましょう。この代入により，変数Aの内容は10から2に更新されます。



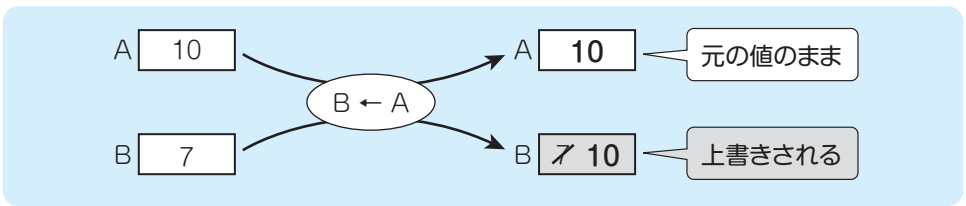
⑤変数には，ほかの変数の値を代入することができる

変数には，ほかの変数の値を代入することができます。たとえば，変数Aの内容を変数Bに代入するときには，

B ← A

のように記述します。このとき，変数Aの内容が変数Bにコピーされると考えましょ

う。移動するわけではないので，変数Aの内容は変わりません(元の値のまま)。

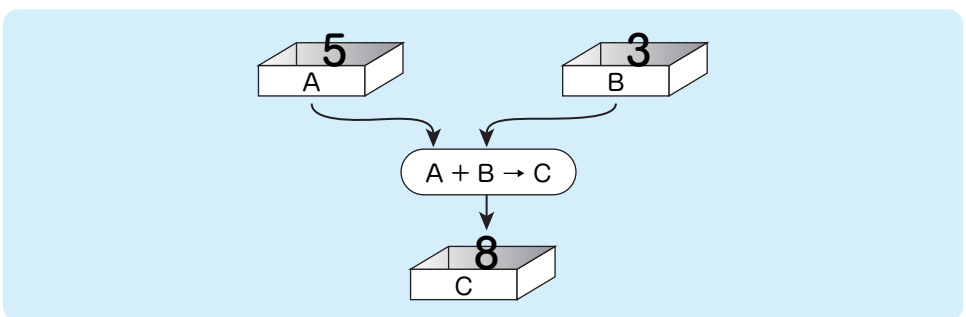


⑥変数には，演算結果を代入することができる

変数には，演算の結果を代入することができます。たとえば，

A ＋ 5 → B　　A ＋ B → C

のように記述されます。たとえば，変数Aに5，変数Bに3が格納されているとすると，「A ＋ B → C」によって変数Cに「8」が格納されることになるわけです。



計算元のデータが格納されている変数に計算結果を代入することもできます。たとえば，

A ＋ 1 → A

のように記述することができるわけです。これは，「変数Aの値に1を加えた結果を，改めて変数Aに代入する」という意味であり，結果として「変数Aの値を1増加させる」ことになります。

基本制御構造

プログラム構造の基本となるのは，

順次

選択(分岐)

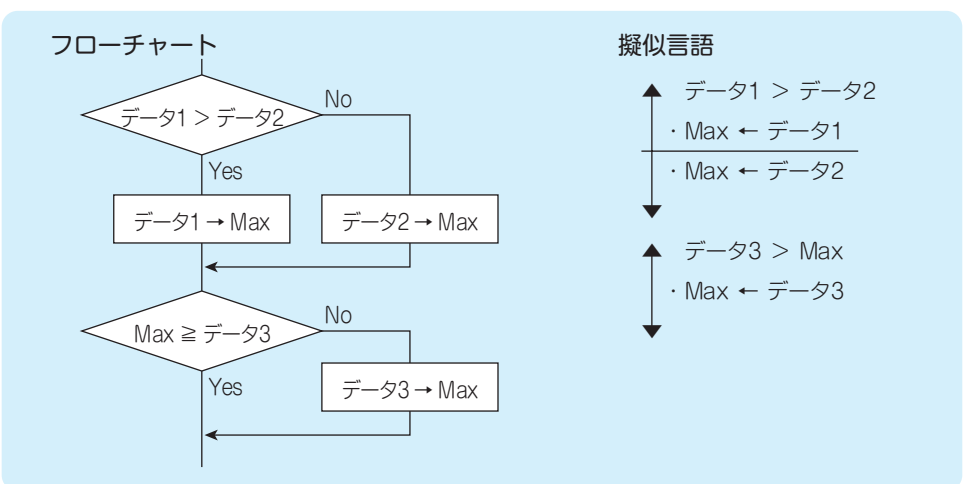
繰返し(ループ)

の三つの制御構造であり，これらを**基本制御構造**といいます。

また，この基本制御構造だけを用いて読み易いプログラムを書く，という考え方を**構造化プログラミング**といいます。

アルゴリズムの表記法

アルゴリズムは処理手順として記述できなければなりません。アルゴリズムの記述法の代表例に，**流れ図(フローチャート)**と**擬似言語**があります。



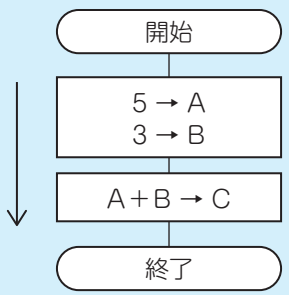
流れ図には視覚的なわかり易さがあります。その反面，記述の自由度が高く，注意しないと煩雑で分かりにくいものになりがちです。擬似言語は視覚的なわかりやすさこそ流れ図に譲りますが，プログラム言語に近くプログラミングとの相性は抜群です。

今回は，流れ図を用いて，説明します。

順次

流れ図では，一番上の開始を表す端子から処理が始まり，一番下の終了を表す端子で処理が終わります。実行する処理は，長方形の記号の中に記述します。

流れ図に記述された処理は，原則として上から下へ処理されていきます。このような制御構造を，“**順次**”といいます。一つの長方形の中に，複数の処理を記述することもできますが，この場合も上に記述された処理が先に行われます。

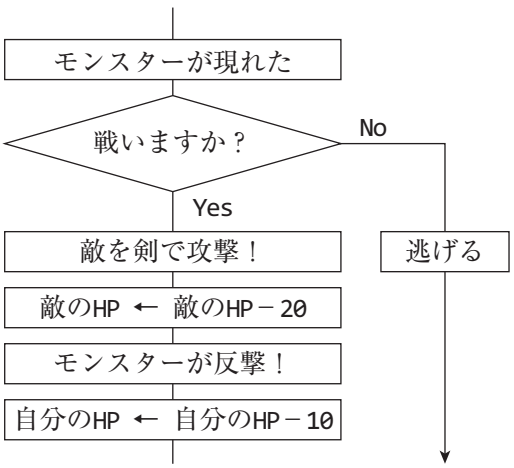


交換

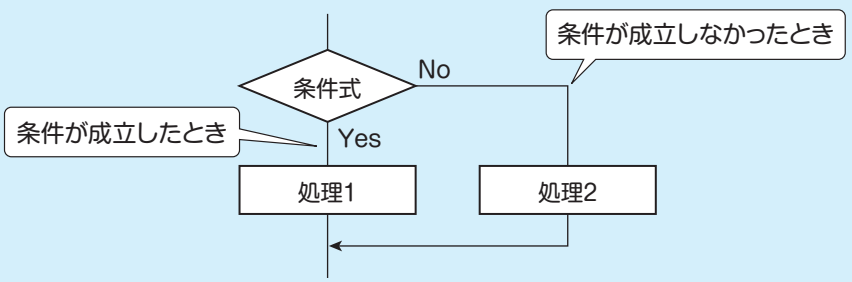
変数AとBの内容を交換してみよう！

分岐(選択)

条件によって，実行する処理を分けることを，“**分岐(選択)**”といいます。

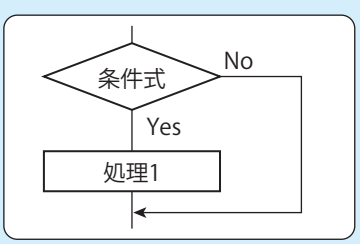


流れ図では，ひし形の記号に条件式を記述し，その条件の判定結果によって，次のようにYes，Noに分岐させます。

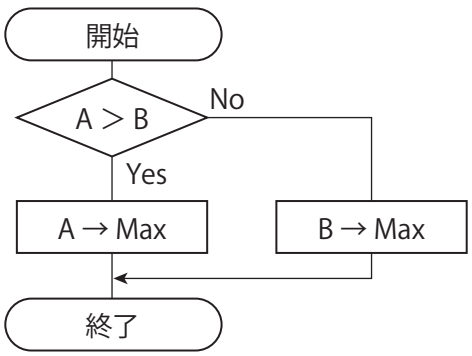


これらの例では，条件が成立した場合(条件式が真の場合)には，処理1が実行されます。このときには処理2は実行されないわけです。逆に，条件が成立しなかった場合(条件式が偽の場合)には，処理2が実行され，処理1は実行されません。

また，条件が成立しなかった場合に何も処理を行わないのであれば，次のように記述します。



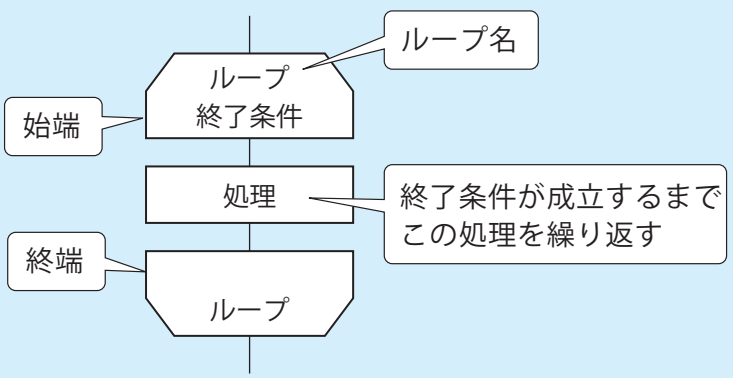
次の例は，変数A，B(共に整数型，値は格納済み)のうち，大きい方の値を変数Maxに格納します。



繰返し処理(ループ処理)

コンピュータの処理では，同じ処理を繰り返す場面が数多く登場します。そのような場合に使用するのが，“**繰返し**(**ループ**)処理です。

流れ図では，繰返し処理を次のように記述します。始端と終端には，同じループであることを表す“ループ名”が記述されます。



流れ図では，擬似言語と異なり，**終了条件**(ループを止める条件)を記述します。そのため，流れ図の繰返し処理では，終了条件が真となるまで，処理を繰り返すことになります。

配列

一つの変数には一つの値しか格納できないので，たとえば「クラス全員のテスト点数をまとめて管理したい」といった場合，変数を人数分だけ用意し，それぞれ別の名前を付けなければなければなりません。このようなときに便利なのが，**配列**です。

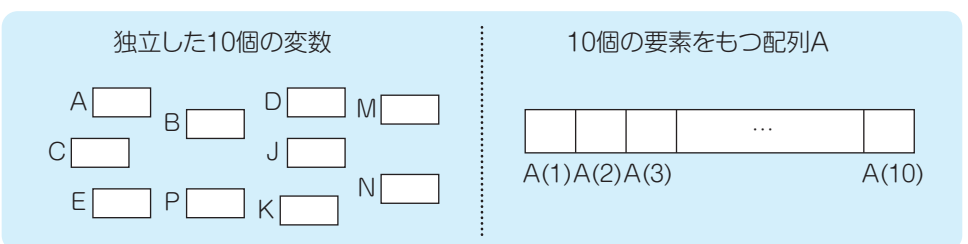
配列は，同じデータ型をもつ複数の要素の集まりと考えればよいでしょう。各要素は，配列名と，「その配列内で何番目か」を表す番号を指定することで識別できます。この番号のことを**添字**といいます。

添字は，配列名の後に()や[]といった括弧で囲んで指定するのが一般的です。また，添字の付け方としては，**先頭要素を0とする場合と，1とする場合があります**(これはプログラム言語などに依存します)。

たとえば,「10個の要素をもつ配列A()を用いる。先頭要素の添字を1とする」といった場合は,

A(1)，A(2)，A(3)，…，A(10)

という10個の要素で構成された配列を用いることになります。

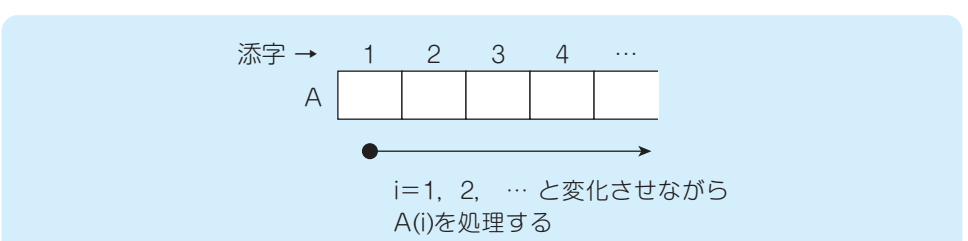


配列を用いることで，同様の処理の繰返しを簡潔に記述することができます。たとえば10人分の成績を出力するような場合は，添字の指定に変数を用いて

変数iを1～10まで変化させながら,

A(i)の値を出力することを繰り返す

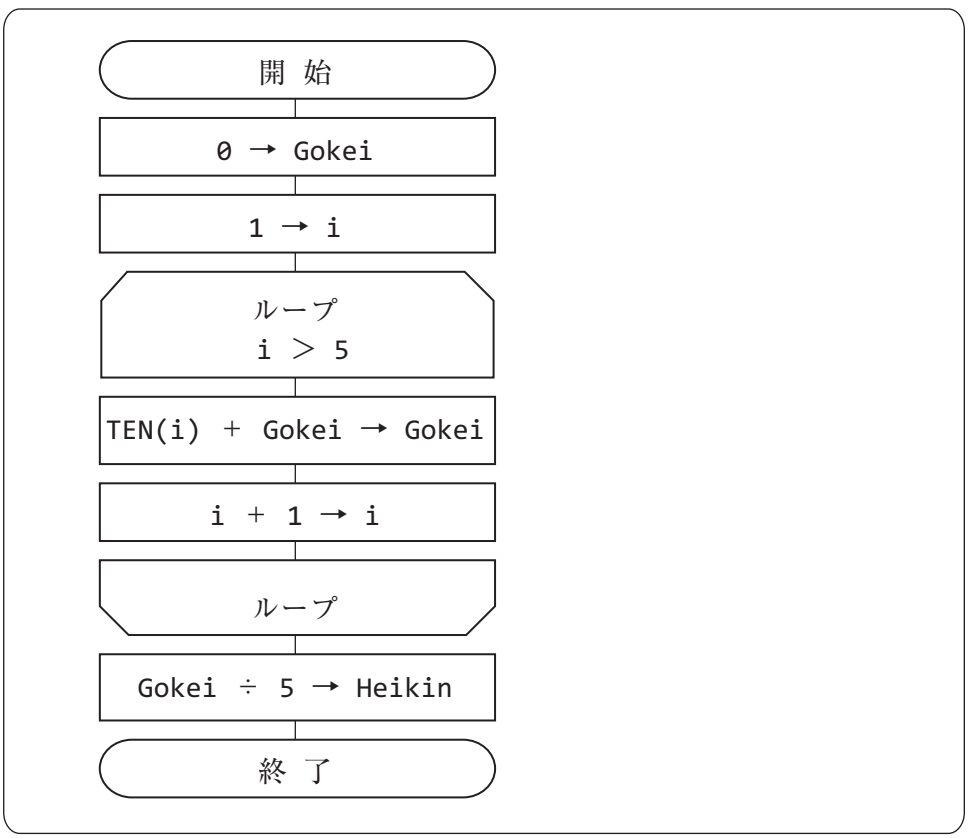
と記述すればよいことになります。



配列処理

配列の各要素を処理するときは，繰返し処理(ループ処理)を用いると，効率良く処理を行うことができます。アルゴリズムでは，配列データを扱うことが非常に多いため，**配列処理がアルゴリズムの基本的な処理となります**。

たとえば5個の要素からなる配列TEN(1) ～ TEN(5)にそれぞれテストの点数が格納されており，その合計と平均を求めたい場合は，次のように記述できます。iの値が，各要素を指定するための添字として使われています。



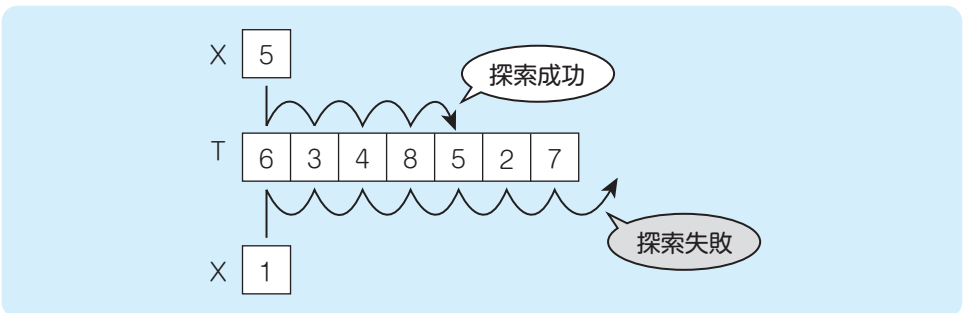
① **ループの前に，添字の初期化が必要！**
② **ループ内に，添字の更新処理が必要！**
③ **ループの条件に，添字を使った条件を使うことが多い！**

線形探索

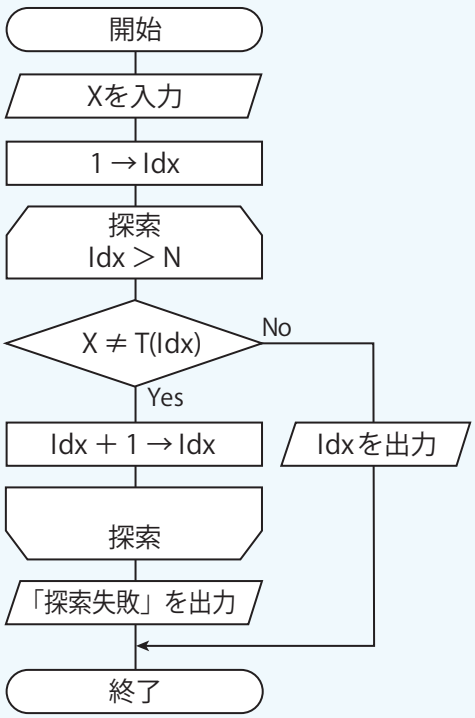
線形探索は，各要素を並んでいる順番に一つずつ調べていく方法です。たとえば配列Tの中にXという値がないかどうか探索する場合，

まず，X と T(1) を比較
次に，X と T(2) を比較
：

というように，先頭から順に比較を行っていきます。この比較は，探索する値に出会う(探索成功)か，すべての要素を比較し終わる(探索失敗)まで続けます。



次に示す流れ図では，探索値Xを入力し，探索対象の配列T(1) ～ T(N)の先頭の要素から順に比較していきます(Nは，配列Tの要素数)。探索値と要素の内容が一致したところで，繰返し処理を抜け，そのときの添字を出力することで，探索値のデータがどの要素に格納されているかを示します。探索対象の配列の最後の要素まで比較しても一致しない場合は，探索ループを終了し「探索失敗」を出力することになります。

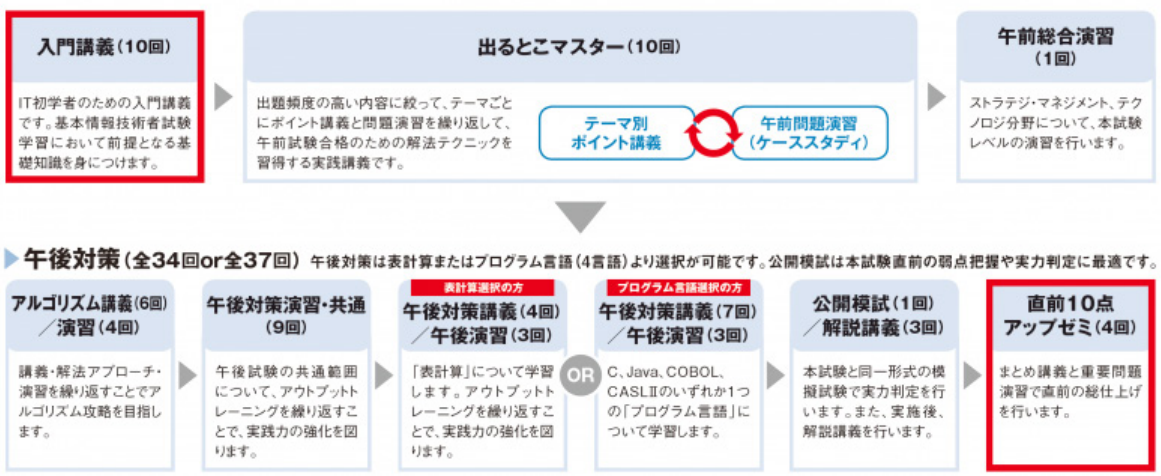


TAC講座へのご案内

基本情報技術者は学習範囲が広く，覚えるべき項目(専門用語や公式など)が多い試験です。独学でやろうとすると，「学校の課題で忙しい」，「仕事が忙しい」，「今日くらいはいいや」となってしまい，どうしても計画どおりに進められないことが多いです。その結果，「試験に間に合わない！」「試験が近いのに，午前も午後もテキストすら終わってない」というパニック状態におちいってしまいます。

そこで，そんな危険を感じる人には，TACで実施している「基本情報技術者講座」の受講をお勧めします。講座では「オリジナル教材」を使用して，講師の迫力ある講義，テスト，質問フォローなど合格に必要なすべてがパッケージされていますので，短期間に無駄なく，効果的に，しかもペースを守って学ぶことができます。

(ＴＡＣ基本情報技術者　本科生プラスのカリキュラム)



●合格のためのアドバイス

・毎回の講義を絶対に欠席しないこと。欠席した場合はフォロー制度を利用して，早めに挽回すること(学習ペースを崩さない)。

・復習を中心とした自己学習をテキスト・問題集でしっかり行うこと。

・疑問点は講師に積極的に質問しましょう。

・「継続は力なり！」諦めないこと。

・試験勉強を生活のリズムの中に組み込みましょう。

・楽しんで学習すること！