```
(*
Instructor：Hiroyuki Akama
```

Misc: The Condition controls*)

In[115]:=
```
? If
```

If[*condition*, *t*, *f*] gives *t* if *condition* evaluates to True, and *f* if it evaluates to False.
If[*condition*, *t*, *f*, *u*] gives *u* if *condition* evaluates to neither True nor False. ≫

In[116]:=
```
If[ListQ[{1}], Print["Yes, This is a " Head[{1}]], Print["No, This is a " Head[{1}]]]
? ListQ
? Head
```

```
Yes, This is a  List
```

ListQ[*expr*] gives True if *expr* is a list, and False otherwise.

Head[*expr*] gives the head of *expr*. ≫

In[119]:=
```
HeadTest[x_] :=
  If[ListQ[x], Print["Yes, This is a " Head[x] ], Print["No, This is a " Head[x] ]];
HeadTest[1]
HeadTest[{1}]
? Print
```

```
No, This is a  Integer

Yes, This is a  List
```

Print[*expr*] prints *expr* as output. ≫

In[123]:=
```
? While
```

While[*test*, *body*] evaluates *test*, then *body*, repetitively, until *test* first fails to give True. ≫

In[124]:=
```
n = 1; result = 0;
While[n < 10, result = result + (1 / 2) ^n; n++]
result // N
```

Out[126]=
```
0.998047
```

In[127]:=
```
? For
```

For[*start*, *test*, *incr*, *body*] executes *start*, then repeatedly evaluates *body* and *incr* until *test* fails to give True. ≫

In[128]:=
```
Clear[n]; Clear[result];
For[n = 1; result = 0, n < 10, n++, result = result + (1 / 2) ^n]
result // N
```

Out[130]=
```
0.998047
```

In[131]:=
```
Plus @@ ((1 / 2) ^# & /@ Range[9]) // N
```

Out[131]=
```
0.998047
```

In[132]:=
```
Clear[r];
For[i = 1; r = 1 / 2, i <= 10, i++, r = r^2]
r // N
```

Out[134]= $5.562684646268003 \times 10^{-309}$

```
Nest[#^2 &, (1 / 2), 10] // N
```

$5.562684646268003 \times 10^{-309}$

In[135]:=
```
? Do
```

Do[*expr*, {i_{max}}] evaluates *expr* i_{max} times.
Do[*expr*, {*i*, i_{max}}] evaluates *expr* with the variable *i* successively taking on the values 1 through i_{max} (in steps of 1).
Do[*expr*, {*i*, i_{min}, i_{max}}] starts with $i = i_{min}$.
Do[*expr*, {*i*, i_{min}, i_{max}, *di*}] uses steps *di*.
Do[*expr*, {*i*, {i_1, i_2, ...}}] uses the successive values i_1, i_2, ...
Do[*expr*, {*i*, i_{min}, i_{max}}, {*j*, j_{min}, j_{max}}, ...] evaluates *expr* looping over different values of *j*, etc. for each *i*. ≫

In[136]:=
```
Clear[r]; r = {};
Do[r = Append[r, i / j], {i, 1, 10}, {j, 1, 10}]
r
```

Out[138]= $\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}, 2, 1, \frac{2}{3}, \frac{1}{2}, \frac{2}{5}, \frac{1}{3}, \frac{2}{7}, \frac{1}{4}, \frac{2}{9}, \frac{1}{5}, 3, \frac{3}{2}, 1, \frac{3}{4}, \frac{3}{5}, \frac{1}{2}, \frac{3}{7}, \frac{3}{8}, \frac{1}{3}, \frac{3}{10}, 4, 2, \frac{4}{3}, 1, \frac{4}{5}, \frac{2}{3}, \frac{4}{7}, \frac{1}{2}, \frac{4}{9}, \frac{2}{5}, 5, \frac{5}{2}, \frac{5}{3}, \frac{5}{4}, 1, \frac{5}{6}, \frac{5}{7}, \frac{5}{8}, \frac{5}{9}, \frac{1}{2}, 6, 3, 2, \frac{3}{2}, \frac{6}{5}, 1, \frac{6}{7}, \frac{3}{4}, \frac{2}{3}, \frac{3}{5}, 7, \frac{7}{2}, \frac{7}{3}, \frac{7}{4}, \frac{7}{5}, \frac{7}{6}, 1, \frac{7}{8}, \frac{7}{9}, \frac{7}{10}, 8, 4, \frac{8}{3}, 2, \frac{8}{5}, \frac{4}{3}, \frac{8}{7}, 1, \frac{8}{9}, \frac{4}{5}, 9, \frac{9}{2}, 3, \frac{9}{4}, \frac{9}{5}, \frac{3}{2}, \frac{9}{7}, \frac{9}{8}, 1, \frac{9}{10}, 10, 5, \frac{10}{3}, \frac{5}{2}, 2, \frac{5}{3}, \frac{10}{7}, \frac{5}{4}, \frac{10}{9}, 1\right\}$

In[139]:=
```
r = {1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9, 1/10, 2, 1, 2/3, 1/2, 2/5, 1/3, 2/7, 1/4, 2/9, 1/5, 3, 3/2, 1, 3/4, 3/5, 1/2,
  3/7, 3/8, 1/3, 3/10, 4, 2, 4/3, 1, 4/5, 2/3, 4/7, 1/2, 4/9, 2/5, 5, 5/2, 5/3, 5/4, 1, 5/6, 5/7, 5/8, 5/9, 1/2, 6,
  3, 2, 3/2, 6/5, 1, 6/7, 3/4, 2/3, 3/5, 7, 7/2, 7/3, 7/4, 7/5, 7/6, 1, 7/8, 7/9, 7/10, 8, 4, 8/3, 2, 8/5, 4/3,
  8/7, 1, 8/9, 4/5, 9, 9/2, 3, 9/4, 9/5, 3/2, 9/7, 9/8, 1, 9/10, 10, 5, 10/3, 5/2, 2, 5/3, 10/7, 5/4, 10/9, 1};
Select[r, # > 1 &]
? Select
```

Out[140]= $\left\{2, 3, \frac{3}{2}, 4, 2, \frac{4}{3}, 5, \frac{5}{2}, \frac{5}{3}, \frac{5}{4}, 6, 3, 2, \frac{3}{2}, \frac{6}{5}, 7, \frac{7}{2}, \frac{7}{3}, \frac{7}{4}, \frac{7}{5}, \frac{7}{6}, 8, 4, \frac{8}{3}, 2, \frac{8}{5}, \frac{4}{3}, \frac{8}{7}, 9, \frac{9}{2}, 3, \frac{9}{4}, \frac{9}{5}, \frac{3}{2}, \frac{9}{7}, \frac{9}{8}, 10, 5, \frac{10}{3}, \frac{5}{2}, 2, \frac{5}{3}, \frac{10}{7}, \frac{5}{4}, \frac{10}{9}\right\}$

Select[*list*, *crit*] picks out all elements e_i of *list* for which *crit*[e_i] is True.
Select[*list*, *crit*, *n*] picks out the first *n* elements for which *crit*[e_i] is True. ≫

In[142]:=
```
Select[r, IntegerQ]
Cases[r, x_?(IntegerQ[#] == True &)]
```

Out[142]= {1, 2, 1, 3, 1, 4, 2, 1, 5, 1, 6, 3, 2, 1, 7, 1, 8, 4, 2, 1, 9, 3, 1, 10, 5, 2, 1}

Out[143]= {1, 2, 1, 3, 1, 4, 2, 1, 5, 1, 6, 3, 2, 1, 7, 1, 8, 4, 2, 1, 9, 3, 1, 10, 5, 2, 1}

In[144]:=
```
? Cases
```

Cases[{e_1, e_2, …}, *pattern*] gives a list of the e_i that match the pattern.
Cases[{e_1, …}, *pattern* –> *rhs*] gives a list of the values of *rhs* corresponding to the e_i that match the pattern.
Cases[*expr*, *pattern*, *levelspec*] gives a list of all parts of *expr* on levels specified by *levelspec* that match the pattern.
Cases[*expr*, *pattern* –> *rhs*, *levelspec*] gives the values of *rhs* that match the pattern.
Cases[*expr*, *pattern*, *levelspec*, *n*] gives the first *n* parts in *expr* that match the pattern. ≫

In[145]:=
```
Select[r, # > 5 & || # < 2 &]
```

Out[145]= $\left\{1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}, \frac{1}{9}, \frac{1}{10}, 1, \frac{2}{3}, \frac{1}{2}, \frac{2}{5}, \frac{1}{3}, \frac{2}{7}, \frac{1}{4}, \frac{2}{9}, \frac{1}{5}, \frac{3}{2}, 1, \frac{3}{4}, \frac{3}{5}, \frac{1}{2}, \frac{3}{7}, \frac{3}{8}, \frac{1}{3}, \frac{3}{10}, \frac{4}{3}, 1, \frac{4}{5}, \frac{2}{3}, \frac{4}{7}, \frac{1}{2}, \frac{4}{9}, \frac{2}{5}, \frac{5}{3}, \frac{5}{4}, 1, \frac{5}{6}, \frac{5}{7}, \frac{5}{8}, \frac{5}{9}, \frac{1}{2}, \frac{3}{2}, \frac{6}{5}, 1, \frac{6}{7}, \frac{3}{4}, \frac{2}{3}, \frac{3}{5}, \frac{7}{4}, \frac{7}{5}, \frac{7}{6}, 1, \frac{7}{8}, \frac{7}{9}, \frac{7}{10}, \frac{8}{5}, \frac{4}{3}, \frac{8}{7}, 1, \frac{8}{9}, \frac{4}{5}, \frac{9}{5}, \frac{3}{2}, \frac{9}{7}, \frac{9}{8}, 1, \frac{9}{10}, \frac{5}{3}, \frac{10}{7}, \frac{5}{4}, \frac{10}{9}, 1\right\}$

In[146]:=
```
Select[r, # > 2 && # < 5 &]
Select[r, 2 < # < 5 &]
% == %%
? %
```

Out[146]= $\left\{3, 4, \frac{5}{2}, 3, \frac{7}{2}, \frac{7}{3}, 4, \frac{8}{3}, \frac{9}{2}, 3, \frac{9}{4}, \frac{10}{3}, \frac{5}{2}\right\}$

Out[147]= $\left\{3, 4, \frac{5}{2}, 3, \frac{7}{2}, \frac{7}{3}, 4, \frac{8}{3}, \frac{9}{2}, 3, \frac{9}{4}, \frac{10}{3}, \frac{5}{2}\right\}$

Out[148]= True

% *n* or Out[*n*] is a global object that is assigned to be the value produced on the nth output line.
% gives the last result generated.
%% gives the result before last. %% … % (*k* times) gives the kth previous result. ≫

In[150]:=
```
alphabet = CharacterRange["a", "z"]
len = Length[%]
num = Range[len]
```

Out[150]= {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}

Out[151]= 26

Out[152]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26}

In[153]:=
```
numalphabetrule = #[[1]] → #[[2]] & /@ Transpose[{num, alphabet}]
```

Out[153]= {1 → a, 2 → b, 3 → c, 4 → d, 5 → e, 6 → f, 7 → g, 8 → h, 9 → i, 10 → j, 11 → k, 12 → l, 13 → m, 14 → n, 15 → o, 16 → p, 17 → q, 18 → r, 19 → s, 20 → t, 21 → u, 22 → v, 23 → w, 24 → x, 25 → y, 26 → z}

In[154]:=
```
num /. numalphabetrule
ReplaceAll[num, numalphabetrule]
? ReplaceAll
```

Out[154]= {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}

Out[155]= {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}

expr /. *rules* applies a rule or list of rules in an attempt to transform each subpart of an expression *expr*. >>

In[157]:=
```
num /. x_ /; x > 15 → "b"
```

Out[157]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, b, b, b, b, b, b, b, b, b, b, b}

In[158]:=
```
# /. (x_ /; x > 15 → b) & /@ num
```

Out[158]= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, b, b, b, b, b, b, b, b, b, b, b}

In[159]:=
```
If[# > 15, #, # /. numalphabetrule] & /@ num
```

Out[159]= {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26}