In[•]:=
```
(*© 2012-Present Computational ClassNotes,
lossofgenerality.org, Creative Commons License *)
(*https://creativecommons.org/licenses/by-nc-sa/3.0/us/  :
  Attribution-NonCommercial-ShareAlike *)

SetOptions[EvaluationNotebook[], Background → LightGray]
```

In[1]:=
```
<< Notation`

(* operators *)
Notation[ (i_ op_ x_ ) ⟹

  AppendTo[order, i_];
  AppendTo[ξ2, If[order2 =!= {},
    Row@{Highlighted[ʊρ[order2[[ctr]]], Background → LightBlue]}, Nothing]];
  ctr++;
  Construct[op_, x_ ]
                                                                                 ]

Notation[ (i_ op_ x_ y_) ⟹

  AppendTo[order, i_];
  AppendTo[ξ2, If[order2 =!= {},
    Row@{Highlighted[ʊρ[order2[[ctr]]], Background → LightBlue]}, Nothing]];
  ctr++;
  Construct[op_, x_ , y_ ]
                                                                                 ]

(*λx  push x into stack ξ1*)
(*all other interm transformations and replacements into ξ2*)
Notation[ (i_ λ x_ . expr_) ⟹
```

```
Block[{tmp},

 AppendTo[order, i_];
 AppendTo[ξ2, If[order2 =!= {}, Row@{Highlighted[℧ρ[order2[[ctr]]], Background →
       LightBlue], " ", x_, "⟶", tmp = Unique[x_], " a new copy"}, Nothing]];
 AppendTo[ξ3, Row[{x_, "⟶", tmp}]];
 AppendTo[ξ1, tmp];
 ctr++;
 Function[x_, expr_][tmp]
]

(*β conversion λ with explicit *)
Notation[ (j_ ( i_ λ x_ . expr_) e2_)  ⟹

 Block[{},
  AppendTo[order, j_];
  AppendTo[order, i_];
  AppendTo[ξ2, If[order2 =!= {}, Row@{Highlighted[℧ρ[order2[[ctr]]],
       Background → LightBlue], " ", x_, " \!\(\*OverscriptBox[\(⟶\),
        \(\"\<β conversion\>\"\)]\) ", If[order2 =!= {}, ctr++;
       ctr++;
       Highlighted[℧ρ[order2[[ctr]]], Background → LightBlue], Nothing]},
    Nothing]];
  Function[x_, expr_][e2_]]
]

(*β conversion without knowing which λ. After
 this pass a post prcessing required to complete β. *)
Notation[ (i_ (e_) f_)  ⟹  ϱ = {i_, (e_), f_} ]

Notation[ (i_ x_)  ⟹

 AppendTo[order, i_];
 AppendTo[ξ2, If[order2 =!= {},
   Row@{Highlighted[℧ρ[order2[[ctr]]], Background → LightBlue]}, Nothing]];
 ctr++;
 x_
]

(*operator to convert the arithmatic operators *)
```

```
Notation[ ℧ x_  ⟹
  StringReplace[x_, {"Plus" → "+", "Subtract" → "-", "Divide" → "/",
    "Times" → "*", "Power" → "^", "Sqrt" → "√", "Mod" → "Mod", "Log" → "Log",
    "Exp" → "Exp", "Sin" → "Sin", "Cos" → "Cos", "Abs" → "Abs", "Minus" → "-"}] ]

(*operators to seararch for variables *)
Notation[ ν x_  ⟹  Select[ToString /@ Level[x_, {-1}], NameQ] ]

Notation[ υ x_  ⟹
  Select[Table[StringTake[i, 1], {i, StringTrim /@ StringSplit[x_, "λ"]}], NameQ] ]

(*operator for initialization. *)
Notation[ ο;  ⟹  ξ1 = {}; ξ2 = {}; ξ3 = {}; ξ4 = {}; ϱ = Nothing; order = {}; ]


eval[expr_, sexpr_] := Module[{all, bound, input, output1, output2, pass1, pass2},

   order2 = {};
   ctr = 1;
   (*pass1 collect order of evaluations*)
   input = ℧ sexpr;
   ο;
   output2 = output1 = ReleaseHold[expr];

   order2 = order;
   ctr = 1;

   (*pass1 collect order of evaluations*)
   input = ℧ sexpr;
   ο;
   output2 = output1 = ReleaseHold[expr];
   bound = ToString /@ ξ1;
   ξ4 = ToExpression /@ Intersection[ν output1, bound];

   If[ϱ =!= Nothing && ξ4 =!= {},
    output2 = ϱ[[2]] /. Table[ξ4[[jj]] → ϱ[[3]], {jj, 1, Length@ξ4}]];


   pass2 = <|
```

```
      "input" -> input,
      If[output1 === output2, "output" → output1, "output1" → output1[[2]]],
      If[output1 === output2, Nothing, "output2" → output2],
      If[output1 === output2, Nothing, "output" → output2],
      "free" -> Complement[ν output1, bound],
      "bound" → Intersection[ν output1, bound],
      "stack" → ξ1,
      "trace" -> ξ2,
      "order" → order

      |>;


    pass2



  ];
```

FIXME

```
In[25]:= ρ = <|

    0 -> "(λ t . (Power ϕ t))",
    1 → "(Power ϕ t)",
    2 → "(λ y . (Divide 5 y))",
    3 → "(Divide 5 y)",
    4 → "(λ z . (Plus 5 z))",
    5 → "(Plus 5 z)",
    6 → "(λ t . (Power ϕ t))",
    7 → "(Power ω t)",
    8 → "start",
    9 → "((λ y . (Divide 5 y))(λ z . (Plus 5 z)))",
    10 → "(λ t . (Power ϕ t))"

    |>;


res =
  eval[Hold[(8 (10 (0 λ t . (1 Power ϕ t)) (9 (2 λ y . (3 Divide 5 y)) (4 λ z . (5 Plus 5 z))))
      (6 λ t . (7 Power ω t)))], "(((λ t . (Power ϕ t)) ((λ y .
     (Divide 5 y))(λ z . (Plus 5 z)))) (λ t . (Power ω t)))"];


res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[27]= (((λ t . (^ ϕ t)) ((λ y . (/ 5 y))(λ z . (+ 5 z)))) (λ t . (^ ω t)))

Out[28]= $\phi^{\frac{5}{5+\omega^{t\$3171}}}$

(λ t . (^ φ t)) t $\xrightarrow{\beta \text{ conversion}}$ ((λ y . (/ 5 y))(λ z . (+ 5 z)))
((λ y . (/ 5 y))(λ z . (+ 5 z))) y $\xrightarrow{\beta \text{ conversion}}$ (λ z . (+ 5 z))
(λ z . (+ 5 z)) z⟶z$3170 a new copy
(+ 5 z)
(/ 5 y)
(^ φ t)
(λ t . (^ φ t)) t⟶t$3171 a new copy
(^ ω t)

Out[29]=

In[30]:=
```
ρ = <|

   1 → "(Power (λ y . (Plus 3 y)) x)",
   2 → "(λ y . (Plus 3 y))",
   3 → "(Plus 3 y)"

   |>;
res = eval[Hold[(1 Power (2 λ y . (3 Plus 3 y)) x)], "(Power (λ y . (Plus 3 y)) x)"];

res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[32]= (^ (λ y . (+ 3 y)) x)

Out[33]= $(3 + y\$3202)^x$

Out[34]=

(^ (λ y . (+ 3 y)) x)
(λ y . (+ 3 y)) y⟶y$3202 a new copy
(+ 3 y)

Constant argument passing requires () or (2) in case below, you also need to tag the paran (5 2). Ignore the gray x.

Note 2 appears again since it has to be evaluated, there is no way for the evaluator to know this is a constant since we could enter some expression quite complex.

In[35]:=

```
ρ = <|
   0 → "((λ x . (Power (λ y . (Plus 3 y)) x)))",
   1 → "((λ x . (Power (λ y . (Plus 3 y)) x)))",
   2 → "(Power (λ y . (Plus 3 y)) x)",
   3 → "(λ y . (Plus 3 y))",
   4 → "(Plus 3 y)",
   5 → "2"

   |>;

res = eval[Hold[(0 (1 λ x . (2 Power (3 λ y . (4 Plus 3 y)) x) ) (5 × 2))],
   "((λ x . (Power (λ y . (Plus 3 y)) x)) 2)"];

res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[37]= ((λ x . (^ (λ y . (+ 3 y)) x)) 2)

Out[38]= $(3 + y\$3214)^2$

Out[39]=

((λ x . (^ (λ y . (+ 3 y)) x))) x $\xrightarrow{\beta \text{ conversion}}$ 2
2
(^ (λ y . (+ 3 y)) x)
(λ y . (+ 3 y)) y⟶y$3214 a new copy
(+ 3 y)

In[40]:=
```
ρ = <|
   0 → "(λ y . (Divide 5 y))",
   1 → "(λ y . (Divide 5 y))",
   2 → "(Divide 5 y)",
   3 → "(λ z . (Plus 5 z))",
   4 → "(Plus 5 z)"

   |>;


res = eval[Hold[(0 (1 λ y . (2 Divide 5 y)) (3 λ z . (4 Plus 5 z)))],
   "((λ y . (Divide 5 y))(λ z . (Plus 5 z)))"];

res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[42]= ((λ y . (/ 5 y))(λ z . (+ 5 z)))

Out[43]= $\frac{5}{5 + z\$3235}$

Out[44]=

(λ y . (/ 5 y))	y $\xrightarrow{\beta\text{ conversion}}$	(λ z . (+ 5 z))
(λ z . (+ 5 z))	z⟶z$3235 a new copy	
(+ 5 z)		
(/ 5 y)		

In[45]:=
```
ρ = <|

   1 → "(λ y . (Sqrt y))",
   2 → "(Sqrt y)",
   3 → "(λ z . (Plus 5 z))",
   4 → "(plus 5 z)"

   |>;


res = eval[Hold[(1 λ y . (2 Sqrt y))], "(λ y . (Sqrt y))"];

res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[47]= (λ y . (√ y))

Out[48]= $\sqrt{\text{y\$3252}}$

Out[49]=

(λ y . (√ y))	y⟶y$3252 a new copy
(√ y)	

λ passing

In[50]:=
```
ρ = <|

   0 -> "(λ t . (Power ϕ t))",
   1 → "(Power ϕ t)",
   2 → "(λ y . (Divide 5 y))",
   3 → "(Divide 5 y)",
   4 → "(λ z . (Plus 5 z))",
   5 → "(Plus 5 z)",
   6 → "(λ t . (Power ω t))",
   7 → "hi",
   8 → "(λ y . (Divide 5 y)",
   9 -> "(Power ω t)",
   10 → "(λ t . (Power ϕ t))"

   |>;


res =
  eval[Hold[(7 (10 (0 λ t . (1 Power ϕ t)) (8 (2 λ y . (3 Divide 5 y)) (4 λ z . (5 Plus 5 z))))
      (6 λ t . (9 Power ω t)))], "(((λ t . (Power ϕ t)) ((λ y .
    (Divide 5 y))(λ z . (Plus 5 z)))) (λ t . (Power ω t)))"];


res["input"]

res["output"]

Column[res["trace"], Dividers → All]
```

Out[52]= (((λ t . (^ ϕ t)) ((λ y . (/ 5 y))(λ z . (+ 5 z)))) (λ t . (^ ω t)))

Out[53]= $\phi^{\frac{5}{5+\omega^{t\$3262}}}$

Out[54]=

(λ t . (^ ϕ t)) t $\xrightarrow{\beta\ \text{conversion}}$ (λ y . (/ 5 y))
(λ y . (/ 5 y)) y $\xrightarrow{\beta\ \text{conversion}}$ (λ z . (+ 5 z))
(λ z . (+ 5 z)) z⟶z$3261 a new copy
(+ 5 z)
(/ 5 y)
(^ ϕ t)
(λ t . (^ ω t)) t⟶t$3262 a new copy
(^ ω t)

In[55]:=
```
ρ = <|

   0 -> "(λ t . (Power ϕ t))",
   1 → "(Power ϕ t)",
   2 → "(λ y . (Divide 5 y))",
   3 → "(Divide 5 y)",
   4 → "(λ z . (Plus 5 z))",
   5 → "(Plus 5 z)",
   6 → "(λ t . (Power ω t))",
   7 → "hi",
   8 → "(λ y . (Divide 5 y)",
   9 -> "(Power ω t)",
   10 → "(λ t . (Power ϕ t))"

   |>;



res =
  eval[Hold[(10 (0 λ t . (1 Power ϕ t)) (8 (2 λ y . (3 Divide 5 y)) (4 λ z . (5 Plus 5 z))))],
   "(((λ t . (Power ϕ t)) ((λ y . (Divide 5
     y))(λ z . (Plus 5 z)))) (λ t . (Power ω t)))"];



res["input"]

res["output"]

Column[res["trace"], Dividers → All]
```

Out[57]= (((λ t . (^ ϕ t)) ((λ y . (/ 5 y))(λ z . (+ 5 z)))) (λ t . (^ ω t)))

Out[58]= $\phi^{\frac{5}{5+z\$3293}}$

Out[59]=

(λ t . (^ ϕ t)) t $\xrightarrow{\beta\ \text{conversion}}$ (λ y . (/ 5 y)
(λ y . (/ 5 y) y $\xrightarrow{\beta\ \text{conversion}}$ (λ z . (+ 5 z))
(λ z . (+ 5 z)) z⟶z$3293 a new copy
(+ 5 z)
(/ 5 y)
(^ ϕ t)

In[60]:=
```
ρ = <|

   1 → "(((λ y . (Divide 5 y))(λ z . (Plus 5 z))) xxx)",
   2 → "(λ y . (Divide 5 y))",
   3 → "(Divide 5 y)",
   4 → "(λ z . (Plus 5 z))",
   5 → "(Plus 5 z)",
   6 → "((λ y . (Divide 5 y))"

   |>;


res = eval[Hold[(1 (6 (2 λ y . (3 Divide 5 y)) (4 λ z . (5 Plus 5 z))) xxx)],
   "(((λ y . (Divide 5 y))(λ z . (Plus 5 z))) xxx)"];

res["input"]
res["output1"]
res["output2"]
Column[res["trace"], Dividers → All]
```

Out[62]= (((λ y . (/ 5 y))(λ z . (+ 5 z))) xxx)

Out[63]= $\frac{5}{5 + z\$3318}$

Out[64]= $\frac{5}{5 + xxx}$

Out[65]=

((λ y . (/ 5 y)) y $\xrightarrow{\beta\text{ conversion}}$ (λ z . (+ 5 z))
(λ z . (+ 5 z)) z⟶z$3318 a new copy
(+ 5 z)
(/ 5 y)

λ x

place a space between λ and variable name otherwise Mathematica interpret then as a single symbol.

Replace operators with math symbols, for now only arithmetic operations supported but we could make more..

℧ is a custom defined operator which coverts the arithmetic operator names to symbols. Had to do this otherwise Mathematica evaluates / in the Lambda expressions which outputs nonsense:

In[∘]:=
```
℧"λ z . (Power 5 z)"
```

Out[∘]=
```
λ z . (^ 5 z)
```

Evaluate

ο is an operator that empties the stack. Normally in the online interpreters this is done invisibly, but since we are using Mathematica's interpreter we need to manually empty the stack if needed .
ξ is the stack, both are weird letters to make sure not used in general.

z is free z$3133 is bound.

In[66]:=
```
ρ = <|

    1 → "(λ z . (Divide b z))",
    2 → "(Divide b z)"

    |>;


res = eval[Hold[(1 λ z . (2 Divide b z))], "(λ z . (Divide b z))"];

res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[68]=
```
(λ z . (/ b z))
```

Out[69]= $\frac{b}{z\$3335}$

Out[70]=

(λ z . (/ b z))	z⟶z$3335 a new copy
(/ b z)	

$\frac{b}{z\$}$ is the result of the evaluation of *λ* expression and z is a bound variable.

Notice z is treated as a placeholder it is replaced by new variables e.g. r and notice the bound variable stack *ξ* is empty

In[71]:=
```
ρ = <|

   1 → "(λ z . (Divide b z)) r",
   2 → "(Divide b z)",
   3 → "r",
   4 → "(λ z . (Divide b z))"

   |>;


res = eval[Hold[(4 (1 λ z . (2 Divide b z)) (3 r))], "(λ z . (Divide b z) r)"];

res["input"]
res["output"]
Column[res["trace"], Dividers → All]
```

Out[73]= (λ z . (/ b z) r)

Out[74]= $\frac{b}{r}$

Out[75]=

(λ z . (/ b z)) z $\xrightarrow{\beta\ \text{conversion}}$ r
r
(/ b z)

Notice z is NOT added to to the stack.

Find variables

bound variables are instanced with trailing $ and unique random integer.

In[76]:=
```
ρ = <|

   1 → "(λ z . (Divide b z))",
   2 → "(Divide b z)"

   |>;

res = eval[Hold[(1 λ z . (2 Divide b z))], "(λ z . (Divide b z))"]
```

Out[77]=
```
⟨|input → (λ z . (/ b z)), output → b/z$3357,
 free → {b}, bound → {z$3357}, stack → {z$3357},
 trace → { (λ z . (/ b z))  z⟶z$3357 a new copy,  (/ b z) }, order → {1, 2}|⟩
```

No free variables

In[78]:=
```
ρ = <|

   1 → "(λ z . (Divide 5 z))",
   2 → "(Divide 5 z)"

   |>;

res = eval[Hold[(1 λ z . (2 Divide 5 z))], "(λ z . (Divide 5 z))"]
```

Out[79]=
```
⟨|input → (λ z . (/ 5 z)), output → 5/z$3366,
 free → {}, bound → {z$3366}, stack → {z$3366},
 trace → { (λ z . (/ 5 z))  z⟶z$3366 a new copy,  (/ 5 z) }, order → {1, 2}|⟩
```

Passing λ

```
ρ = <|

   1 → "(λ x . (Divide 5 x))",
   2 → "(Divide 5 x)",
   3 → "(λ z . (Plus (-3) z))",
   4 → "(Plus (-3) z)",
   5 → "(λ x . (Divide 5 x))",
   6 → "(Minus 3)"

   |>;

(* -3 shorthand does not work, we need to explictly use the operator Minus*)
res = eval[Hold[(5 (1 λ x . (2 Divide 5 x)) (3 λ z . (4 Plus (6 Minus 3) z)))],
  "((λ x . (Divide 5 x))(λ z . (Plus (-3) z)))"]

Column[res["trace"], Dividers → All]
```

Out[81]= ⟨| input → ((λ x . (/ 5 x))(λ z . (+ (-3) z))),
output → $\frac{5}{-3 + z\$3375}$, free → {}, bound → {z$3375}, stack → {z$3375},
trace → {(λ x . (/ 5 x)) x $\xrightarrow{\beta\ \text{conversion}}$ (λ z . (+ (-3) z)), (λ z . (+ (-3) z)) z ⟶z$3375 a new copy, (+ (-3) z), (- 3), (/ 5 x)}, order → {5, 1, 3, 4, 6, 2} |⟩

Out[82]=

(λ x . (/ 5 x)) x $\xrightarrow{\beta\ \text{conversion}}$ (λ z . (+ (-3) z))
(λ z . (+ (-3) z)) z⟶z$3375 a new copy
(+ (-3) z)
(- 3)
(/ 5 x)

Notice the left most λ expressions variable name does not matter:

```
In[83]:= ρ = <|

    1 → "(λ W . (Divide 5 W))",
    2 → "(Divide 5 W)",
    3 → "(λ z . (Plus (-3) z))",
    4 → "(Plus (-3) z)",
    5 → "(λ x . (Divide 5 x))",
    6 → "(Minus 3)"

    |>;

res = eval[Hold[(5 (1 λ W . (2 Divide 5 W)) (3 λ z . (4 Plus (6 Minus 3) z)))],
  "((λ W . (Divide 5 W))(λ z . (Plus (-3) z)))"]

Column[res["trace"], Dividers → All]
```

Out[84]= ⟨| input → ((λ W . (/ 5 W))(λ z . (+ (-3) z))),

output → $\frac{5}{-3 + z\$3412}$, free → {}, bound → {z$3412}, stack → {z$3412},

trace → {(λ x . (/ 5 x)) W $\xrightarrow{\beta\text{ conversion}}$ (λ z . (+ (-3) z)), (λ z . (+ (-3) z)) z ⟶z$3412 a new copy, (+ (-3) z), (- 3), (/ 5 W)}, order → {5, 1, 3, 4, 6, 2} |⟩

Out[85]=

(λ x . (/ 5 x)) W $\xrightarrow{\beta\text{ conversion}}$ (λ z . (+ (-3) z))
(λ z . (+ (-3) z)) z⟶z$3412 a new copy
(+ (-3) z)
(- 3)
(/ 5 W)