```
{importthis}
   print
   print
  10
{}#10#11
{}printprint
     0xhex()0oct()
int()
{}printprintprint
{}
int()str()
{}
printprint
{}
     bin()float()chr()ord()
round()
{}printprint
     complex( )abs( )
{}print {}
     divmod(x,y)
isinstance()
{}printprintprint
{}
```

```
{}
       print
- print "Hello World"
- print "Hello", <Variable Containing the String>
- print "Hello" + <Variable Containing the String>
- print "Hello %s" % <variable containing the string>
 \{\} print
   }print
   }print
       string 1\,string 2
 {}print
       print
 {}print
{}
%s
{}print%s
{}
- %s -> string
- %d -> Integer
- %f -> Float
- %o -> Octal
- %x -> Hexadecimal
- %e -> exponential
{}print%d18print%f18print%o18print%x18print%e18

m print\%s\%s
   print%%%s
   print
    print"n"n"n"n"n"n
   print ""
    print"t"t"n"t
    %f
    \%.5f
    \%9.5f
    \%020.5f
   print% 9fprint% 9f
   print\%+9fprint\% 9f
    %-9.3f
```

```
}print
  print
  print
  {printprint
    len()
    \min(\ )\max(\ )
 }in
  in
    max( )min( )max( )min( )
  {printprint
 {printprint
max()max()min()
{}
printprint
{}
    \max_{list()}()\min()
    append()
{}
{}
{}
    count( )
    append()
{{} print
    extend( )
{}print
    index()
    insert(x,y)append(\ )
```

```
{printprintprint
 printprint
    find()-1
 }printprint
 }print
    find()
{}printprint
    capitalize()
{}print
{}
    center()
    zfill()
expandtabs(), "t"t"t"ttprintprintprint
    index( )find( )index( )
{}printprintprint
    endswith()
\{\}print
 printprint
    count()
{}printprint
{}
    join()
    join()
{}printprint {}
    join()
{}print {}
    split( )join()
{}print {}
    split()
{}printprint {}
    lower()
{}printprint {}
    upper()
    replace()
    strip( )
    strip( )
```

{}
<pre>algorithm {}ifprint {}</pre>
algorithm
algorithm {}ifprintelseprint {}
algorithm
algorithm
algorithm {}ifprintelifprintelseprint {}
if printel if printif printels eprintels eprint {}
algorithm
{} for inprint
{} for inprint {}
{} for inforing rint {}
algorithm
{}whileprintprint
{} for inprint if break
{ forinif print continue elif print
forinprint
{{forin
{{forinif
forinifforin
\mathcal{V}

```
"," Document String","
statements
return <value> {}printprint
  defprintprint
firstfunc()
    secondfunc()firstfunc()
{}defprintprintdef
  defreturn
    times()
 }print
 def '''This multiplies the two input arguments'''return
 {print
    times( )times( )help( )
  defreturn
  print"n"n"n
  defreturn
    implicitadd( )
implicitadd( )
  defforinprintreturn
  defdefprintprintprintprint
    global
 defdefglobalprintprintprintprint
map()
 {
m f lambdaprint}
```

```
\{\}
 \begin{tabular}{ll} Functions \\ \{\} classFirstClasspass \\ pass \end{tabular} 
    init( )init( )
 {classFirstClassdef
  printprint
     dir()
     dir()
 {classFirstClassdef}
  printprint
  printprint
  classFirstClassdef
  printprint
  classFirstClassdef
  print
  class First Class def de freturn de freturn de freturn
  printprint
  {\bf class Software Engineer def def print}
  {\bf class Art ist def def print def print}
  classArtistdefprint
  {\bf class Art ist def print def print print}
  {\bf classemptylist defdefdefdef}
```