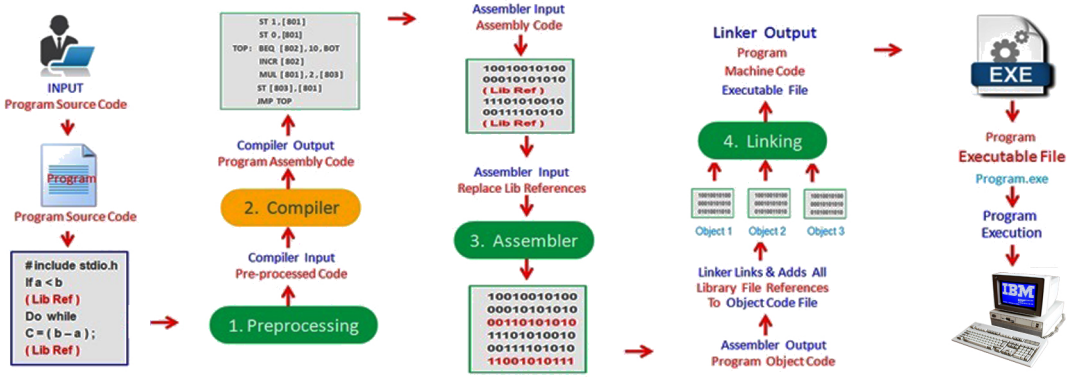
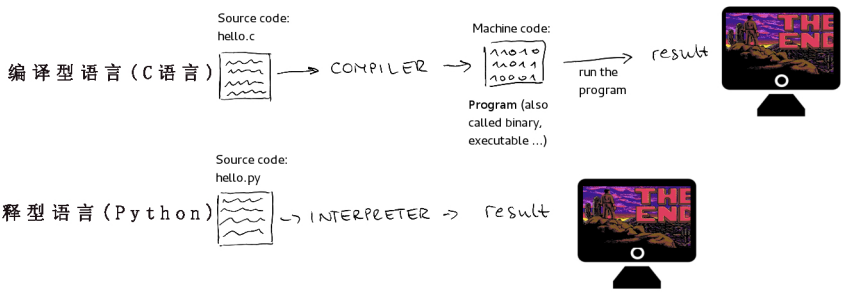
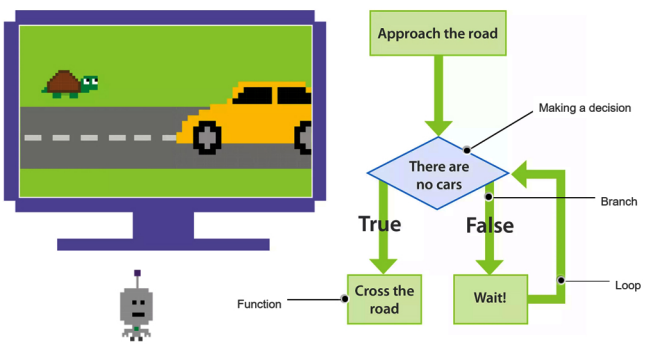
高级语言的编译：





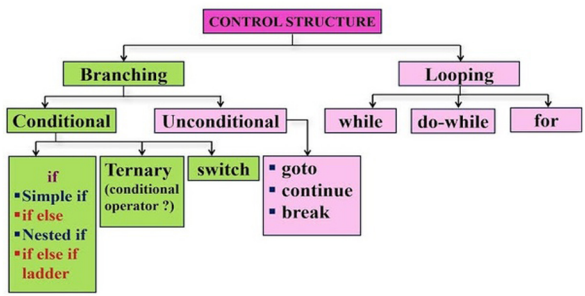
高级语言中的基本概念：



$p \wedge q \to r$

$p \vee q \to r$

$\neg p \to r$

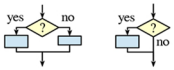


思考：条件分支还可以等价为哪些数学形式？？

Conditional Statement

statement block executed only ***if*** *a condition is true*

```
if logical condition:
    statements block
```



Can go with several *elif*, *elif*... and only one final *else*. Only the block of first true condition is executed.

```
if age<=18:
    state="Kid"
elif age>65:
    state="Retired"
else:
    state="Active"
```

with a var **x**:

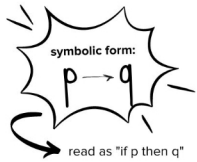
```
if bool(x)==True:  ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

CONDITIONAL STATEMENT

a statement that can be written in "if - then" form

in the sentence, after if comes the **hypothesis**

and after then comes the **conclusion**



Conditional Loop Statement

statements block executed ***as long as*** *condition is true*

```
while logical condition:
    statements block
```


```
s = 0
i = 1
while i <= 100:
    s = s + i**2
    i = i + 1
print("sum:",s)
```

initializations ***before*** *the loop*

condition with a least one variable value (here `i`*)*

make condition variable change !

Algo:

$$s = \sum_{i=1}^{i=100} i^2$$

Loop Control

`break` *immediate exit*

`continue` *next iteration*

`else` *block for* ***normal*** *loop exit.*

beware of infinite loops!

Iterative Loop Statement

statements block executed ***for each*** *item of a container or iterator*

```
for var in sequence:
    statements block
```

Go over sequence's **values**

```
s = "Some text"
cnt = 0
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

initializations ***before*** *the loop*

loop variable, ***assignment managed by*** `for` ***statement***

Algo: count number of `e` *in the string.*

loop on dict/set ⇔ loop on keys sequences

use *slices* to loop on a subset of a sequence

Go over sequence's **index**

- modify item at index
- access items around index (before / after)

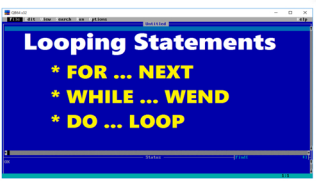
```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Algo: limit values greater than 15, memorizing of lost values.

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

good habit : don't modify loop variable



Function Definition

function name (identifier)

named parameters

```
def fct(x,y,z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

`fct`

result value of the call, if no computed result to return: `return None`

parameters and all variables of this block exist only *in* the block and *during* the function call (think of a "black box")

Advanced: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`

**args variable positional arguments (→`tuple`), default values,*

***kwargs variable named arguments (→`dict`)*

Function Call

```
r = fct(3,i+2,2*i)
```

storage/use of returned value

one argument per parameter

this is the use of function name *with parentheses* which does the call

Advanced:
**sequence*
***dict*