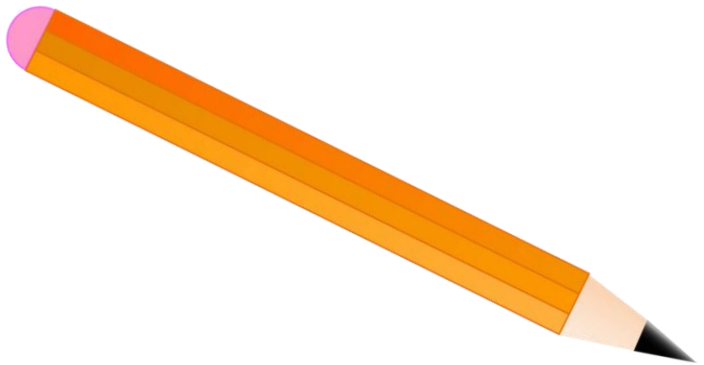
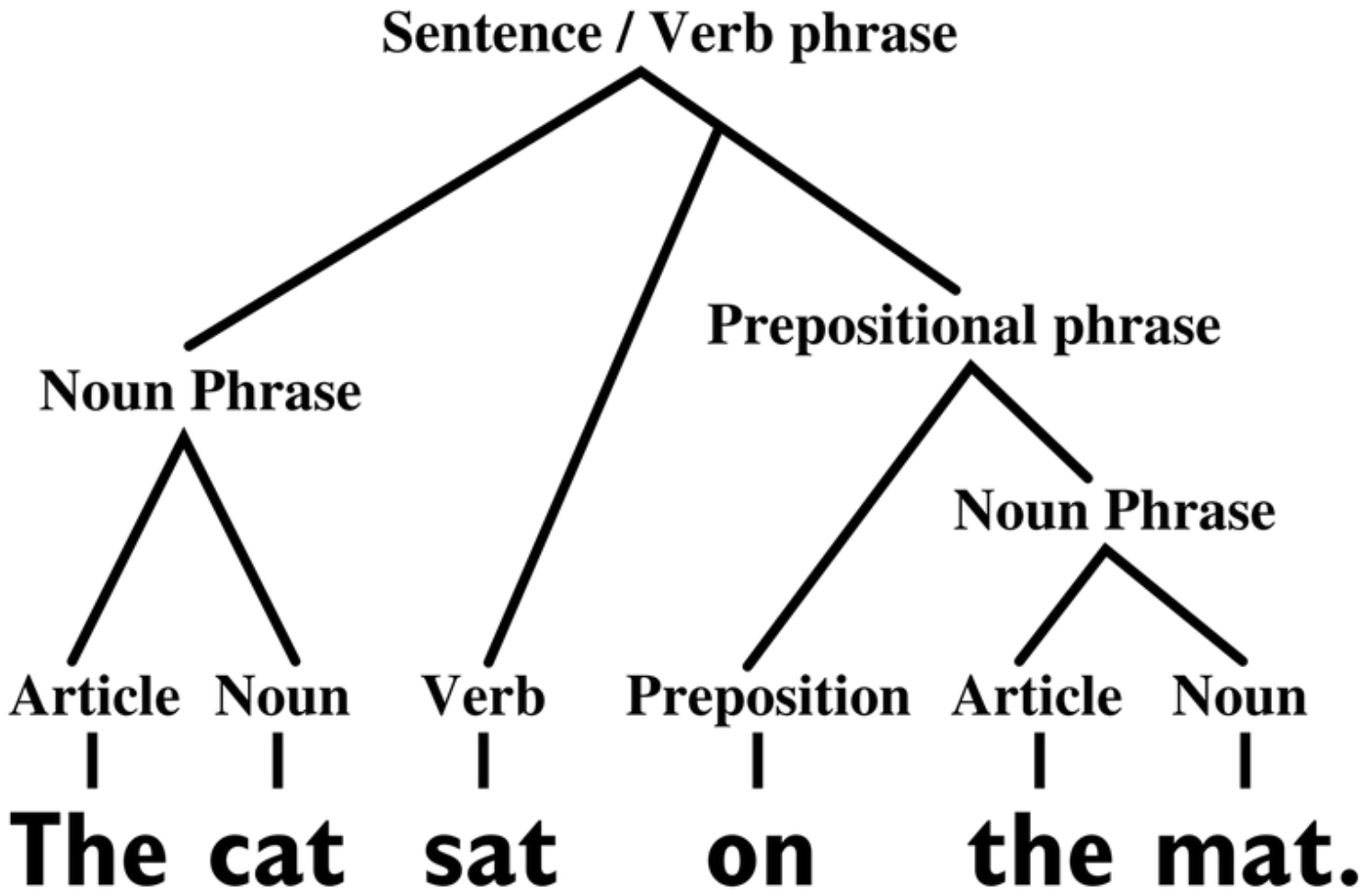
BASH Language

Syntax

A set of rules that determines the arrangement of words in a sentence





Syntax for Printing in Various Languages

C++

```
cout << “Hello World!”;
```

PHP

```
<?php echo '<p>Hello World</p>'; ?>
```

JAVA

```
System.out.println(“Hello World!”);
```

Python 3.x

```
print(“Hello World!”)
```

Bash

```
echo Hello World!
```

Basic BASH

Goal: Print “Hello World!” to the screen

Code

```
#!/bin/bash
echo Hello World!
```

First Line
Chosen Interpreter

Output

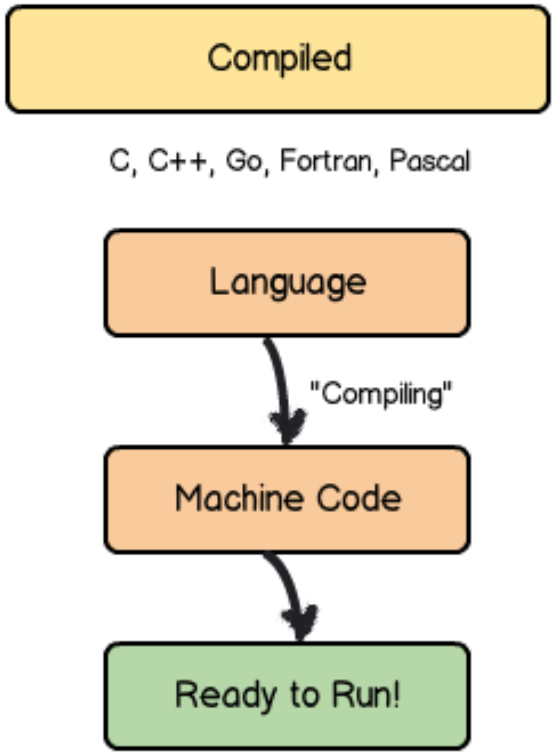
```
Hello World!
```

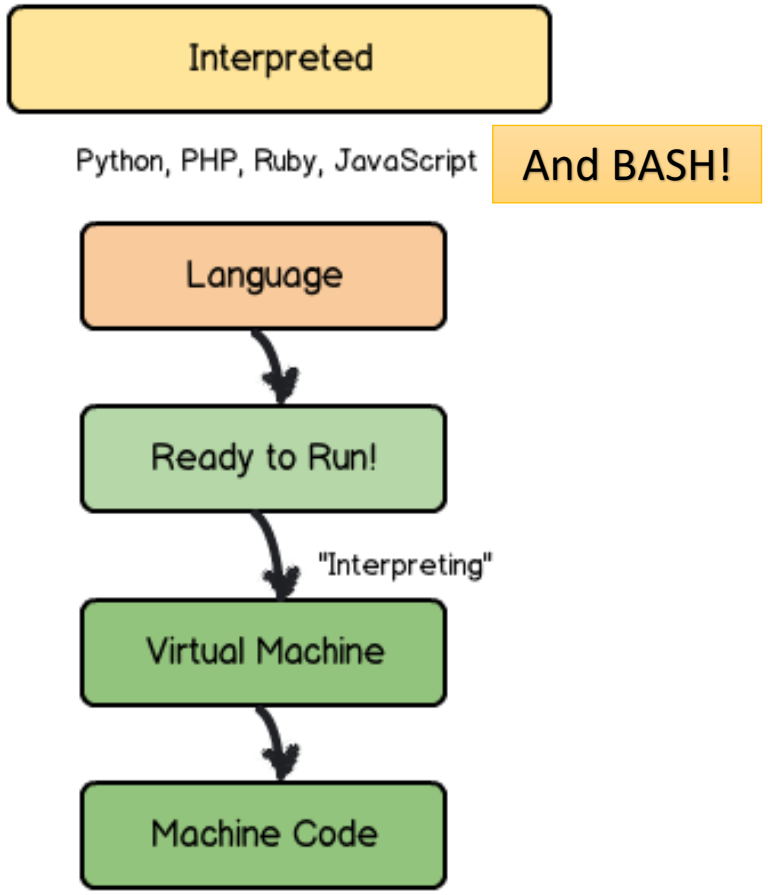
#!

Path to Interpreter

Shebang

Interpreted vs Compiled Languages





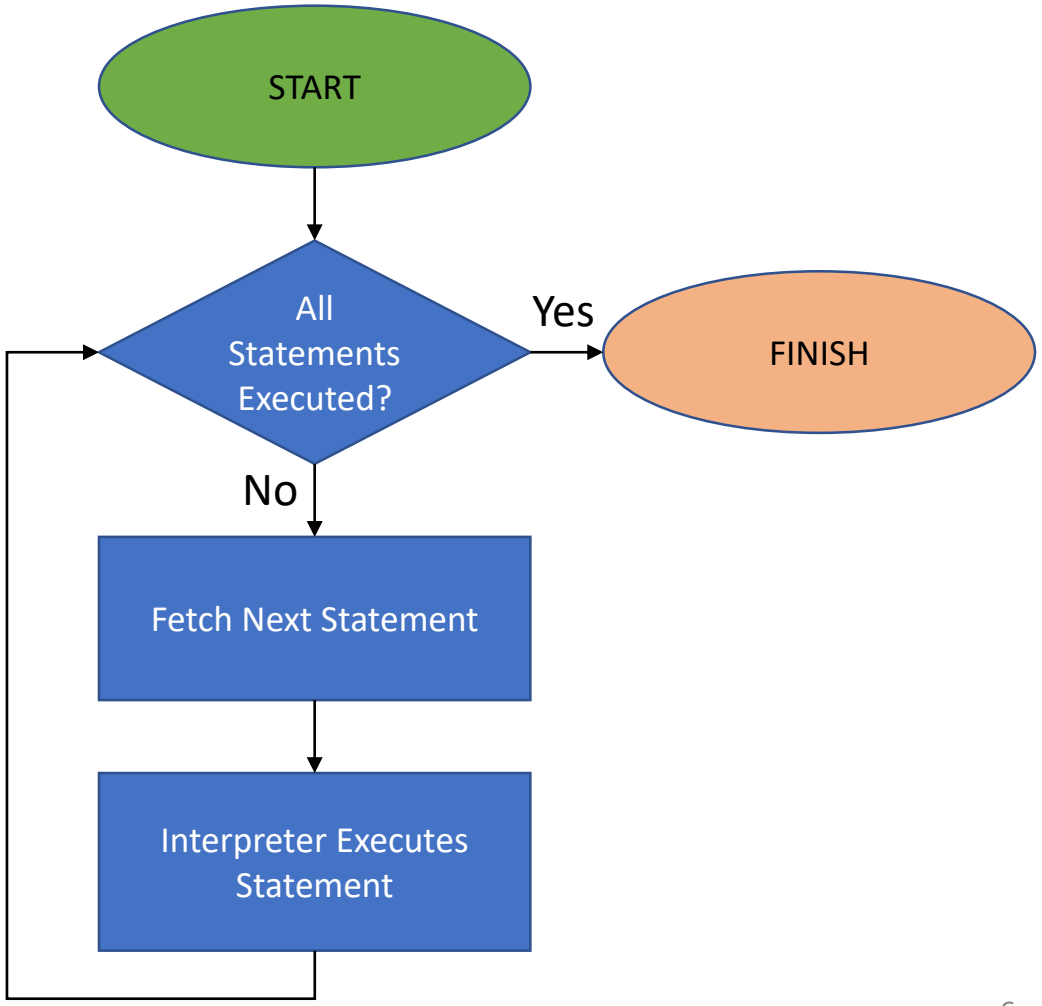
Interpreted Languages

Use an interpreter to execute code line by line

```
#!/bin/bash

addend1=3
addend2=4

echo $addend1
echo $addend2

$sum=3+4
```

Susceptible to runtime errors

```
./simple_add.sh: line 9: =3+4: command not found
```



Types of Programming Errors

Syntax

```
ECHO $addend1
echo $addend2
```

Runtime

```
$sum=3+4
```

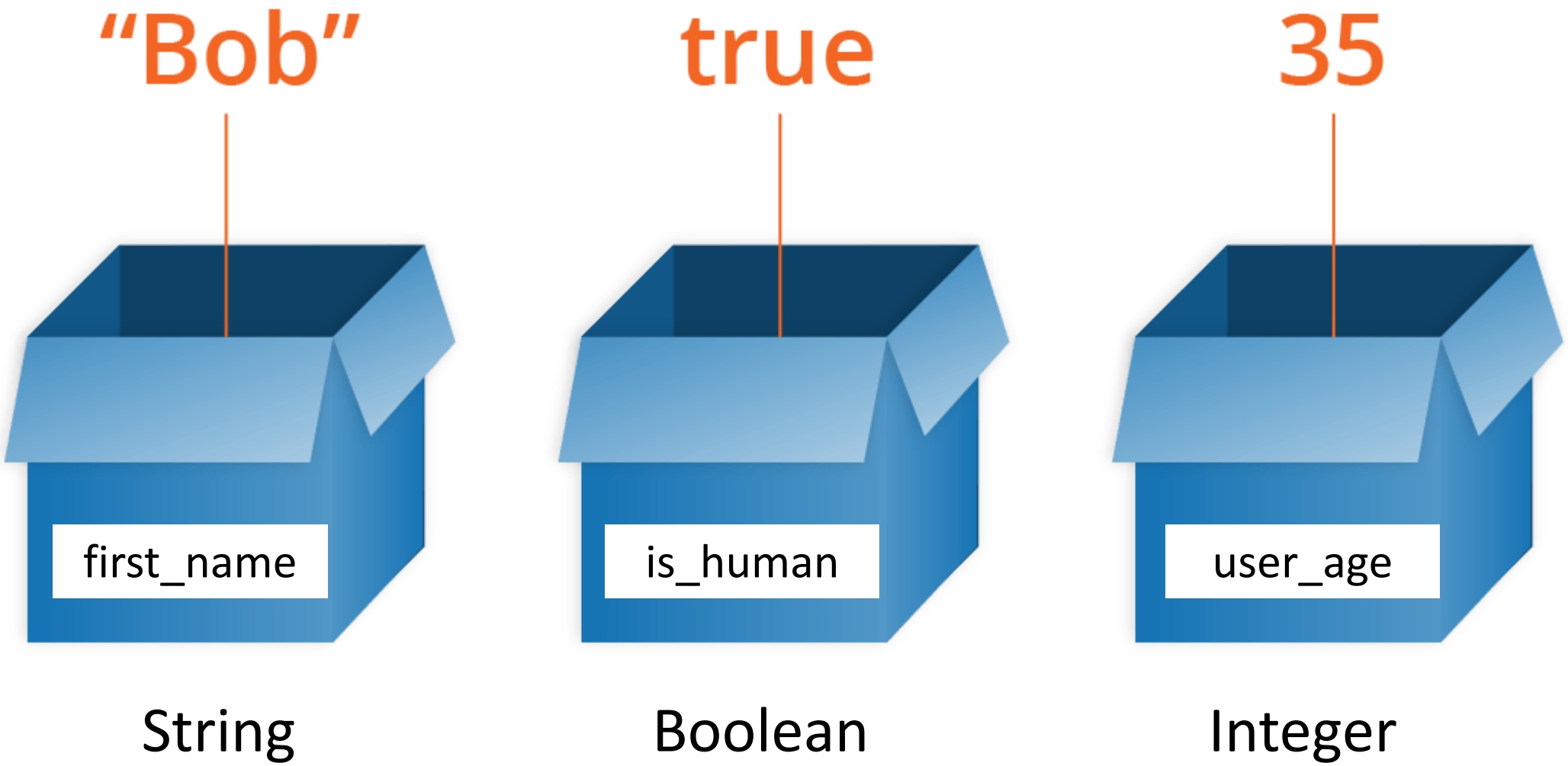
```
./simple_add.sh: line 9: =3+4: command not found
```

Logic

```
#!/bin/bash

addend1=3
addend2=4

echo $addend1
echo $addend2

sum=$(($addend1*$addend2))

echo $sum
```

```
3
4
12
```

Variables


"Bob"
true
35
first_name
is_human
user_age
String
Boolean
Integer

Variables - Types

type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB" "Hello" "2.5"`

Variables - BASH

Storing

```
[ec2-user@ip-172-31-84-203 ~]$ first_name="Bob"
[ec2-user@ip-172-31-84-203 ~]$ is_human=true
[ec2-user@ip-172-31-84-203 ~]$ user_age=35
```

Accessing

```
[ec2-user@ip-172-31-84-203 ~]$ echo $first_name
Bob
[ec2-user@ip-172-31-84-203 ~]$ echo $is_human
true
[ec2-user@ip-172-31-84-203 ~]$ echo $user_age
35
```

Operators

A symbol that tells the compiler or interpreter to perform specific mathematical, relational or logical operation and produce final result.

Relational	Assignment	Logical
<	=	&
>	+=	|
<=	-=	!
>=	*=	&&
==	/=	||
!=		

Operators - BASH

Bash Shell Test Operators

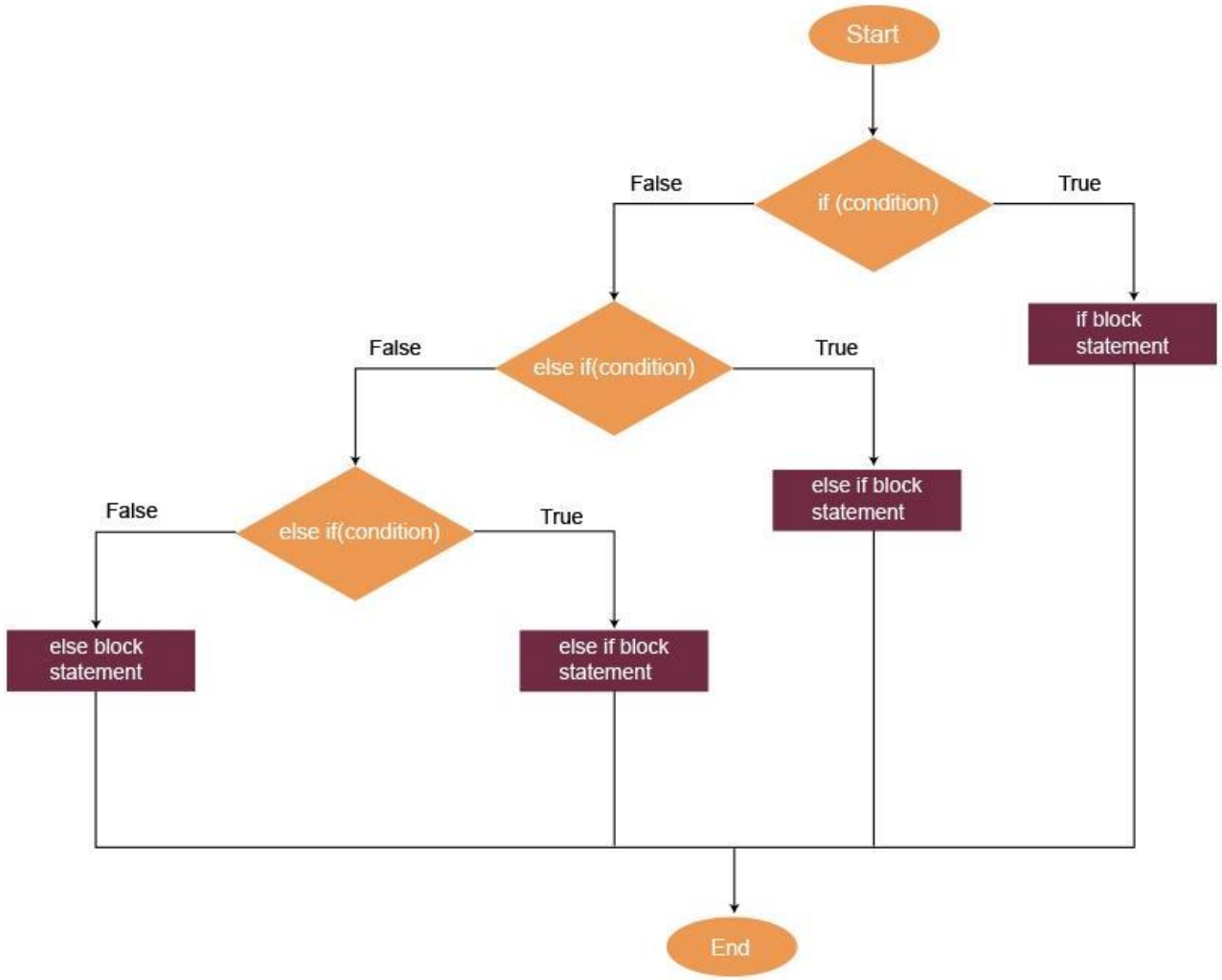
Integer Comparisons	Function
-gt	Greater than
-lt	Less than
-ge	Greater than or equal to
-le	Less than or equal to
-eq	Equal to
-ne	Not equal to
String Comparisons	**Functions**
-z	Test for empty string
=	Test for equality of strings
!=	Test for inequality of strings

Logical Operators	Function
-a	Logical AND
-o	Logical OR
!	Logical NOT
File Test Operators	**Function**
-f	File exists and is a regular file
-s	File is not empty
-r	File is readable
-w	File can be written to and modified
-x	File is executable
-d	Filename is a directory name

Decision Making

IF, ELSE IF, ELSE

- IF blocks do not need an ELSE IF or ELSE block
- ELSE IF and ELSE blocks need at least one IF block in the chain and an IF or ELSE IF directly above



Decision Making - BASH - IF

```
if [ conditional expression ]
then
        statement1
fi
```

DO NOT FORGET THE SPACES!!!

```
#!/bin/bash
count=100
if [ $count -eq 100 ]
then
  echo "Count is 100"
fi
```

```
Count is 100
```

Decision Making - BASH - IF (Single Line)

if [conditional expression]; then statement1; fi

DO NOT FORGET THE SPACES!!!

Notice the addition of the semicolons

```
#!/bin/bash
count=100
if [ $count -eq 100 ]; then echo "Count is 100"; fi
```

```
Count is 100
```

Decision Making - BASH - IF..ELSE

```
if [ conditional expression ]
then
        statement1
else
        statement2
fi
```

```
#!/bin/bash
count=99
if [ $count -eq 100 ]
then
  echo "Count is 100"
else
  echo "Count is not 100"
fi
```

```
Count is not 100
```

Decision Making - BASH - IF..ELIF..ELSE

```
if [ conditional expression ]
then
        statement1
elif [ conditional expression ]
then
        statement2
else
        statement3
fi
```

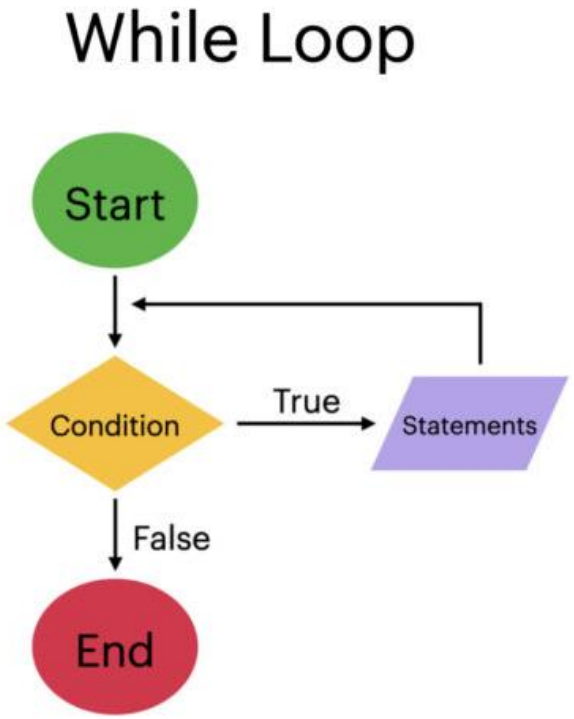
```
#!/bin/bash
count=101
if [ $count -eq 100 ]
then
  echo "Count is 100"
elif [ $count -gt 100 ]
then
  echo "Count is greater than 100"
else
  echo "Count is less than 100"
fi
```

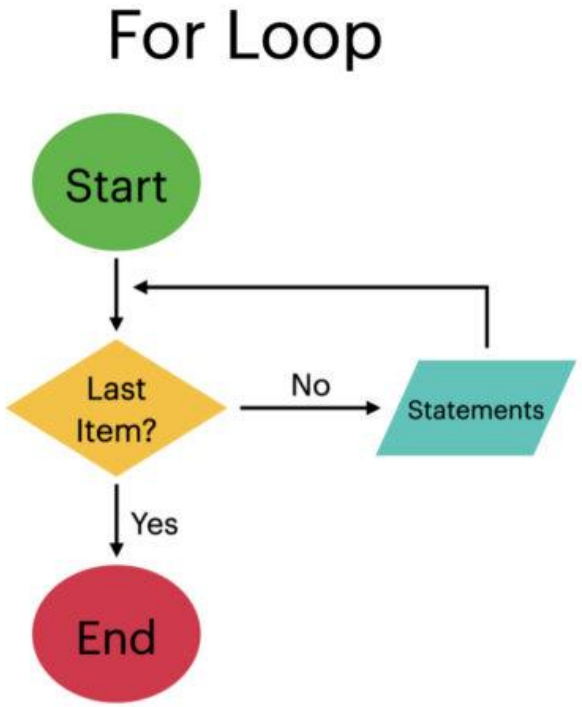
```
Count is greater than 100
```

Repetition

WHILE, FOR

Repeat until not true or condition met



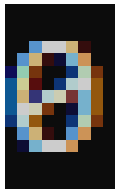


Repetition - BASH - While

```
while [ conditional expression ]
do
        statement1
done
```

```
#!/bin/bash

start=0

while [ $start -lt 5 ]
do
  echo $start
  start=5
done
```

MAKE SURE YOU MODIFY THE VARIABLE REFERENCED IN THE CONDITION TO AVOID INFINITE LOOPS!



Repetition - BASH - FOR

```
for VARIABLE in 1 2 3 4 5 .. N
do
        statement1
done
```

Iterates through a list
Could be files, numbers, etc.

```
#!/bin/bash

for i in 1 2 3 4 5
do
  echo $i
done
```

```
1
2
3
4
5
```

Repetition - BASH - FOR Set

```
for VARIABLE in {1..5}
do
        statement1
done
```

Set includes all numbers

```
#!/bin/bash

for i in {1..5}
do
  echo $i
done
```

```
1
2
3
4
5
```

Repetition - BASH - FOR Files

```
for VARIABLE in PATH
do
        statement1
done
```

Can be relative or absolute path

```
#!/bin/bash

for f in *
do
  echo $f
done
```

```
comment.sh
for_files.sh
for_set.sh
for.sh
hello_world.sh
if_elif_else.sh
if_else.sh
if.sh
if_single.sh
simple_add.sh
while.sh
```

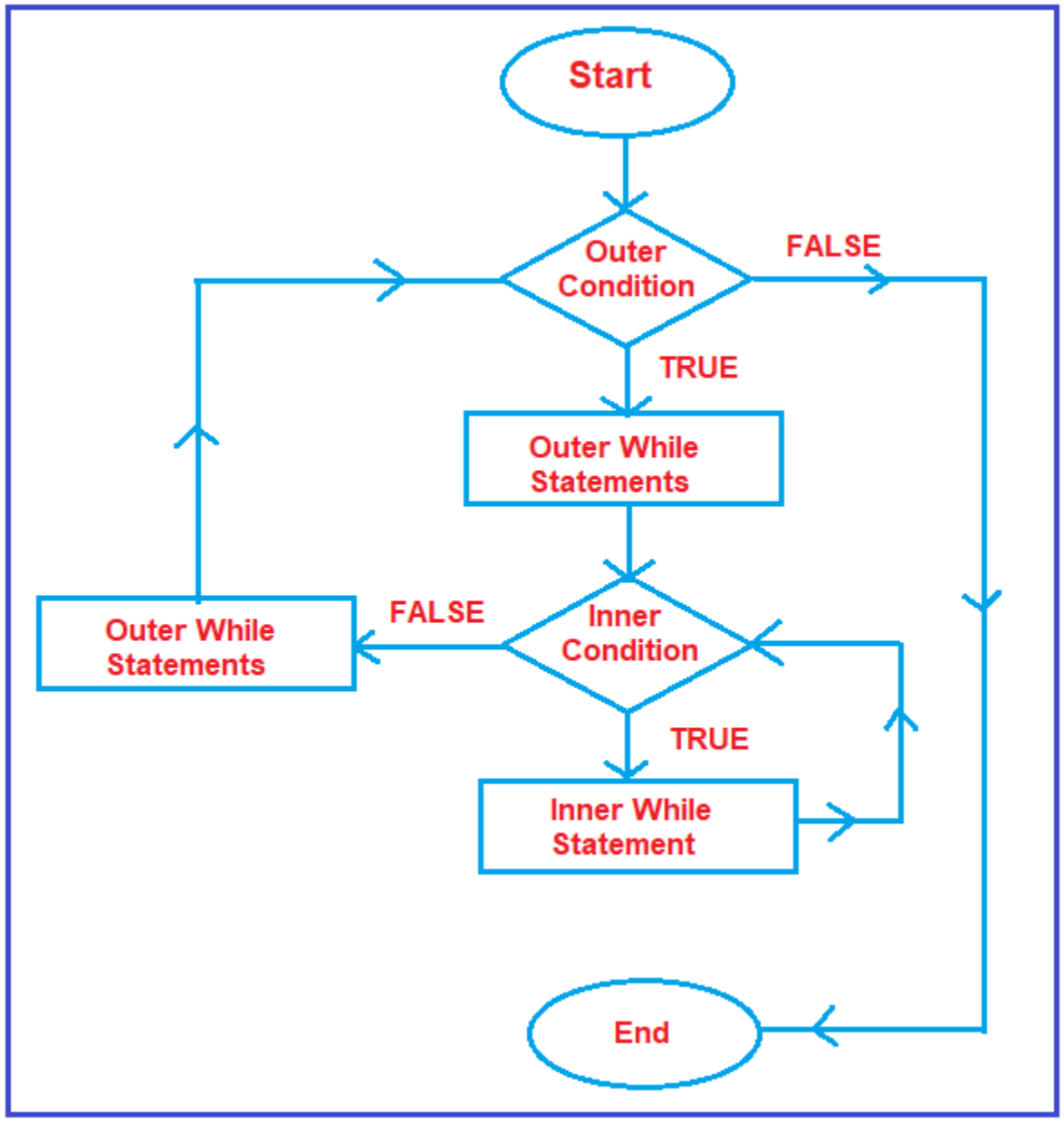
Nesting

- A combination of conditional or repetition blocks “nested” inside each other

Outer Loop

Inner Loop

```
for (int row = 1; row <= 3; row++)
{
    for (int col = 1; col <= 5; col++)
    {
        System.out.print("*");
    }
    System.out.println();
}
```

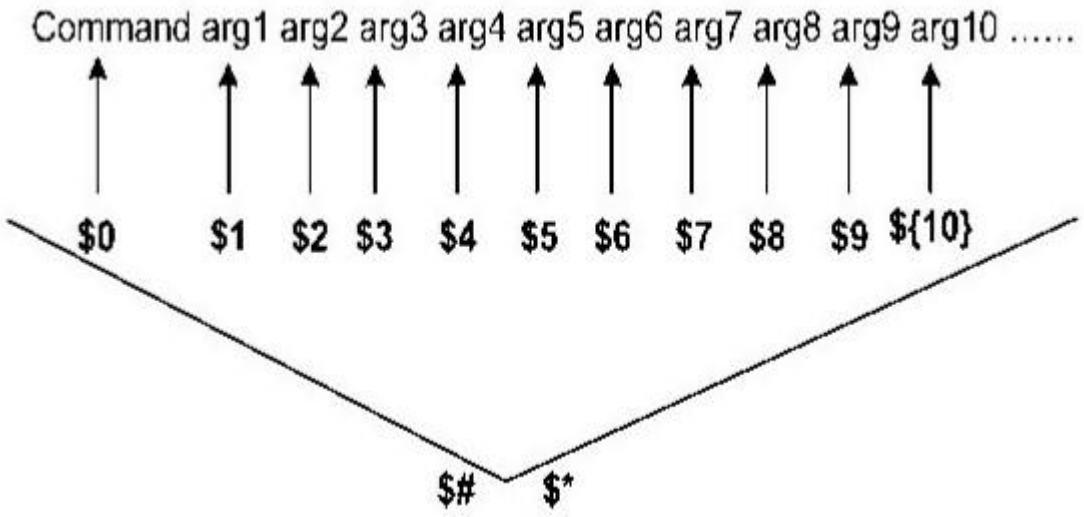


Arguments

- Used to pass data to scripts
- Accessed with numerical position of argument

$# - number of arguments

$* - set of arguments



Arguments - BASH

```
#!/bin/bash

echo $1
echo $2

echo $#
echo $*
```

```
$ ./arguments.sh first_arg second_arg
```

```
first_arg
second_arg
2
first_arg second_arg
```

Commenting

Not executed by interpreter

```
#!/bin/bash

# Comments arent executed
# echo Hi from a comment
echo Hi from outside of a comment
```

```
Hi from outside of a comment
```

Interpreting Quotes

“ ” – weak – allows substitution

```
username="Carol Smith"
echo "Hello $username!"
```

```
Hello Carol Smith!
```

‘ ’ – strong – prevent substitution

```
username="Carol Smith"
echo 'Hello $username!'
```

```
Hello $username!
```