Control Flow

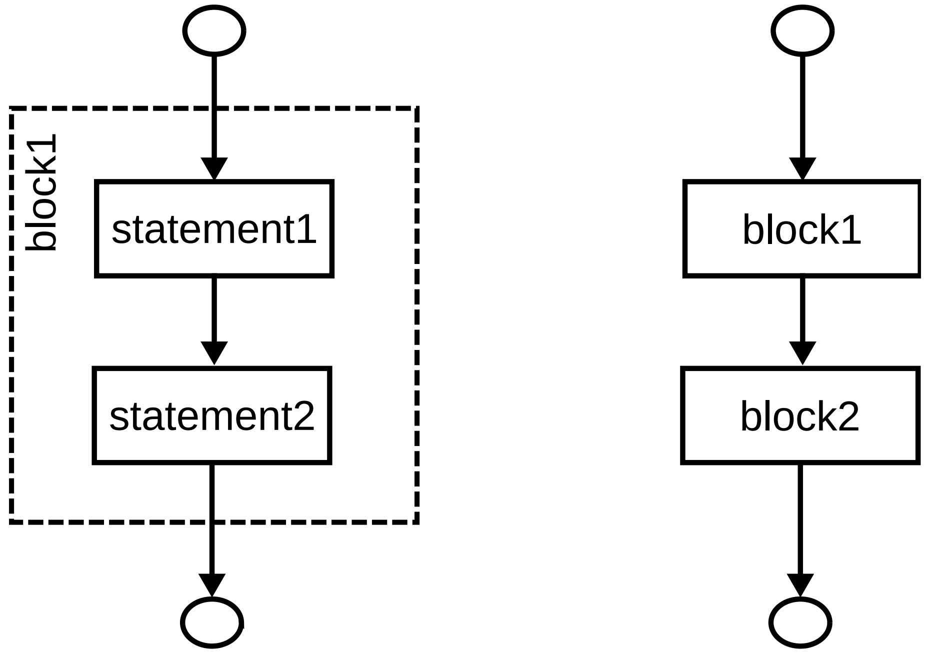
- programs are made up of blocks

- blocks are made up of statements
- and sub-blocks

- control flow: order in which blocks are executed

Sequential Execution

- blocks are executed one after the other



Conditions

- Boolean expressions
- result is either `True` or `False`

- comparison operators: `<`, `<=`, `>`, `>=`, `==`, `!=`

Comparison Operator Examples

expression	result
`4 < 2`	`False`
`4 > 2`	`True`
`4 >= 2`	`True`
`4 == 2`	`False`
`4 != 2`	`True`

Compound Expressions

- not
- and: True if both operands are True, False otherwise
- or: False if both operands are False, True otherwise

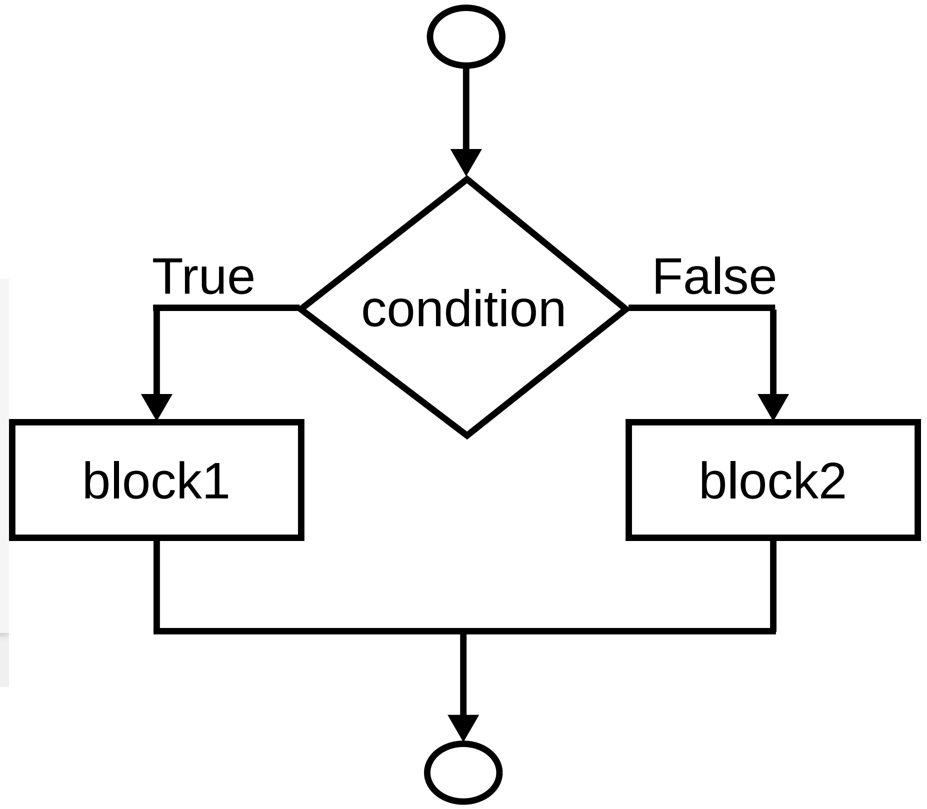
Compound Expression Examples

p	q	p and q	p or q	not (p or q)
True	True	True	True	False
True	False	False	True	False
False	True	False	True	False
False	False	False	False	True

Conditional Statement

- based on condition,
  execute one of two blocks

```
if CONDITION:
    BLOCK1
else:
    BLOCK2
```



Conditional Execution Example

```python
raw_midterm = input("Midterm: ")
midterm = int(raw_midterm)
raw_final = input("Final: ")
final = int(raw_final)
total = midterm * 0.45 + final * 0.55
if total >= 40:
    print("Passed")
else:
    print("Failed")
```

Conditional Execution - 2

- false branch may be omitted

```
if CONDITION:
    BLOCK
```

Conditional Execution Example - 2

```python
raw_midterm = input("Midterm: ")
midterm = int(raw_midterm)
raw_final = input("Final: ")
final = int(raw_final)
total = midterm * 0.45 + final * 0.55
if total >= 40:
    print("Passed")
```

Nested Conditions

- conditional blocks can be nested

```
if CONDITION1:
    STATEMENT1
    if CONDITION1a:
        BLOCK1a1
    else:
        BLOCK1a2
else:
    BLOCK2
```

Nested Condition Example

```
response = input("Please enter your birth year: ")
birth_year = int(response)
if birth_year >= 2000:
    print("You are a post-millenial.")
else:
    if birth_year >= 1980:
        print("You are a millenial/gen-Y.")
    else:
        if birth_year >= 1960:
            print("You are a gen-X.")
        else:
            if birth_year >= 1940:
                print("You are a baby-boomer.")
            else:
                print("Nobody can remember what you are.")
```

Multiple Comparisons

- simpler syntax: `if - elif - else`

```
if CONDITION1:
    BLOCK1
elif CONDITION2:
    BLOCK2
elif CONDITION3:
    BLOCK3
...
else:
    BLOCK IF ALL FALSE
```

Multiple Comparison Example

```python
response = input("Please enter your birth year: ")
birth_year = int(response)

if birth_year >= 2000:
    print("You are a post-millenial.")
elif birth_year >= 1980:
    print("You are a millenial/gen-Y.")
elif birth_year >= 1960:
    print("You are a gen-X.")
elif birth_year >= 1940:
    print("You are a baby-boomer.")
else:
    print("Nobody can remember what you are.")
```

Conditional Expression

- based on condition,
  evaluate one of two expressions

```
EXPRESSION1 if CONDITION else EXPRESSION2
```

Conditional Expression Example

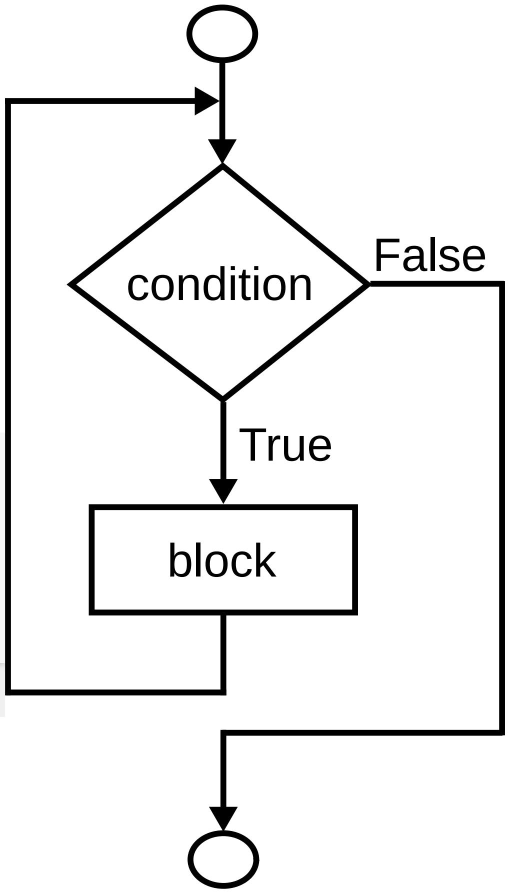
- number of days in February:

```
29 if year % 4 == 0 else 28
```

Iterative Execution

- based on condition, repeatedly execute block
- loop

```
while CONDITION:
    BLOCK
```



Infinite Loops

- block has to affect the outcome of the condition
- otherwise: infinite loop

Iterative Execution Example

- Fibonacci numbers: $1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$
- next number is sum of previous two numbers

- print the first n numbers

Iterative Execution Example - Code

```python
raw_n = input("How many numbers? ")
n = int(raw_n)

num1 = 1
print(num1)
num2 = 1
print(num2)

i = 3
while i <= n:
    num3 = num1 + num2
    print(num3)
    num1 = num2
    num2 = num3
    i = i + 1
```

Lists

- list: a collection of items of the same type
- literals: within square brackets
- number of items: `len`

```
>>> grades = [85, 26, 40, 71, 85, 95]
>>> len(grades)
6
```

Accessing List Items

- list indexing: `list_var[index]`

- index of first item: `0`
- index of last item: `len(list_var) - 1`

List Indexing Example

```
>>> grades
[85, 26, 40, 71, 85, 95]
>>> grades[0]
85
>>> grades[1]
26
>>> grades[5]
95
```

Overstepping Bounds

- what if:

```
grades[6]
```

Membership Check

- whether an item is a member of a list or not:

```
ITEM in LIST_VAR
```

```
>>> grades
[85, 26, 40, 71, 85, 95]
>>> 26 in grades
True
>>> 61 in grades
False
```

Changing Items

- list items can be changed

```
>>> grades
[85, 26, 40, 71, 85, 95]
>>> grades[2] = 77
>>> grades
[85, 26, 77, 71, 85, 95]
```

String Indexing

- strings can be indexed the same way

```
>>> group = "Monty Python"
>>> group[0]
'M'
>>> group[9]
'h'
```

Changing Strings

- strings can NOT be changed

```
>>> group[4] = 'e'
```

List Concatenation

- addition on lists: concatenation

```
>>> fibs1 = [1, 1, 2, 3, 5]
>>> fibs2 = [8, 13, 21, 34, 55, 89]
>>> fibs1 + fibs2
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

List Slicing

- selecting a sublist from a list:

```
LIST_VAR[START_INDEX:STOP_INDEX]
```

- if start index is not given, start from 0
- if stop index is not given, stop at end

List Slicing Examples

```
>>> grades
[85, 26, 77, 71, 85, 95]
>>> grades[2:5]
[77, 71, 85]
>>> grades[3:]
[71, 85, 95]
>>> grades[:4]
[85, 26, 77, 71]
>>> grades
[85, 26, 77, 71, 85, 95]
```

List Slicing Examples - 2

- assign back to the same variable

```
>>> group[4] = 'e'

>>> group
'Monty Python'
>>> group = group[:4] + 'e' + group[5:]
>>> group
```

Deleting Items

- removing an item from a list:

```
del LIST_VAR[INDEX]
```

```
>>> grades
[85, 26, 77, 71, 85, 95]
>>> del grades[3]
[85, 26, 77, 85, 95]
>>> grades
[85, 26, 77, 85, 95]
```

Iterating over Indexes

- template:

```
i = 0
while i < len(LIST_VAR):
    ITEM = LIST_VAR[i]
    # process ITEM
    i = i + 1
```

List Iteration Example - 1

- are all numbers in a list the same?

```python
# nums = [4, 4, 4, 4, 4]
value = nums[0]
all_same = True
i = 0
while i < len(nums):
    num = nums[i]
    if num != value:
        all_same = False
    i = i + 1
print(all_same)
```

List Iteration Example - 2

```python
# nums = [4, 4, 4, 4, 4]
value = nums[0]
all_same = True
i = 0
while all_same and (i < len(nums)):
    num = nums[i]
    if num != value:
        all_same = False
    i = i + 1
print(all_same)
```

Stopping Iteration

- if result of iteration is decided: `break`
- get out of the innermost loop

```python
# nums = [4, 4, 4, 4, 4]
value = nums[0]
all_same = True
i = 0
while i < len(nums):
    num = nums[i]
    if num != value:
        all_same = False
        break
    i = i + 1
print(all_same)
```

Iterating over Items

- template:

```
for ITEM in LIST_VAR:
    # process ITEM
```

List Iteration Example - 3

```python
# nums = [4, 4, 4, 4, 4]
value = nums[0]
all_same = True
for num in nums:
    if num != value:
        all_same = False
        break
print(all_same)
```

Counter Iteration

- function for generating counter sequence
- `range(start, stop, step)`