Control Flow

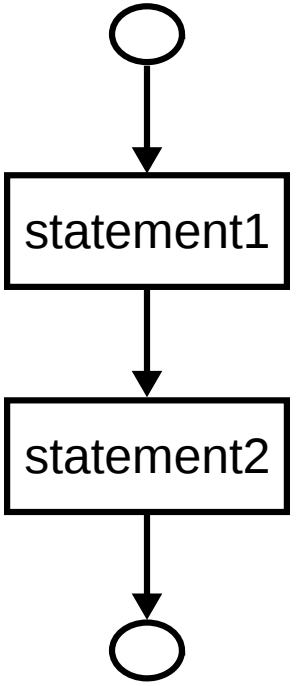
- programs are made up of blocks
- block: sequence of statements

- control flow: order in which blocks are executed

- sequential
- conditional
- iterative

Sequential Execution

- statements 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

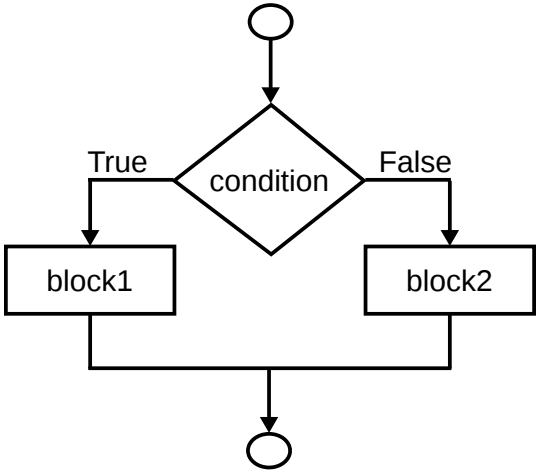
Compound Expression Examples

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 result of condition, choose block to execute

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



Conditional Execution Example

```
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

```
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 statements 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

```
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 result of condition,
  choose expression to evaluate

```
EXPRESSION1 if CONDITION else EXPRESSION2
```

Conditional Expression Example

- number of days in February

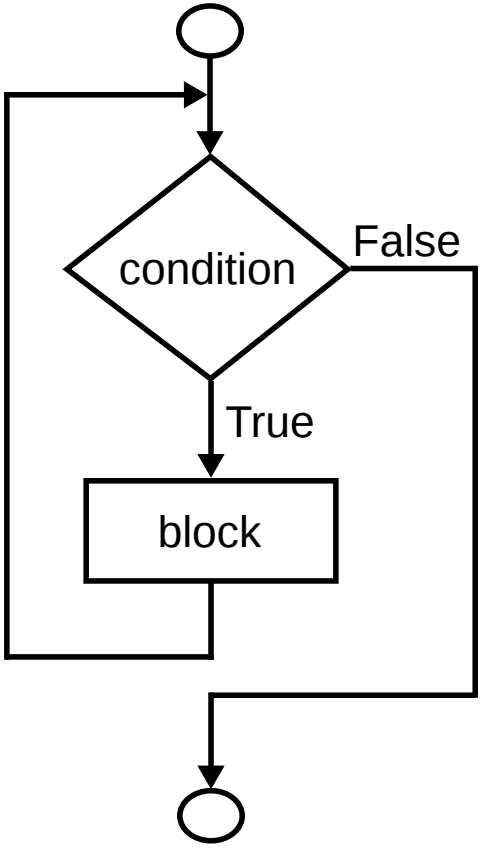
```
In [1]: year = 2016

In [2]: 29 if year % 4 == 0 else 28
Out[2]: 29
```

Iterative Execution

- based on result of condition, repeatedly execute block
- loop

```
while CONDITION:
    BLOCK
```



Infinite Loops

- block has to effect 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

```
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`

```
In [3]: grades = [85, 26, 40, 71, 85, 95]

In [4]: len(grades)
Out[4]: 6
```

Accessing List Items

- list indexing: `list_var[index]`

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

List Indexing Example

```
In [5]: grades
Out[5]: [85, 26, 40, 71, 85, 95]

In [6]: grades[0]
Out[6]: 85

In [7]: grades[1]
Out[7]: 26

In [8]: grades[5]
Out[8]: 95
```

Index Error

- index out of bounds: `IndexError`

```
In [9]: grades
Out[9]: [85, 26, 40, 71, 85, 95]

In [10]: grades[6]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent
<ipython-input-10-af75dd00e160> in <module>()
----> 1 grades[6]

IndexError: list index out of range
```

Membership Check

- whether an item is a member of a list or not
- `ITEM in LIST_VAR`

```
In [11]: grades
Out[11]: [85, 26, 40, 71, 85, 95]

In [12]: 26 in grades
Out[12]: True

In [13]: 61 in grades
Out[13]: False
```

Changing Items

- list items can be changed

```
In [14]: grades
Out[14]: [85, 26, 40, 71, 85, 95]

In [15]: grades[2] = 77

In [16]: grades
Out[16]: [85, 26, 77, 71, 85, 95]
```

String Indexing

- strings can be indexed the same way

```
In [17]: group = 'Monty Python'

In [18]: group[0]
Out[18]: 'M'

In [19]: group[9]
Out[19]: 'h'
```

Changing Strings

- strings can NOT be changed

```
In [20]: group
Out[20]: 'Monty Python'

In [21]: group[4] = 'e'
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent
<ipython-input-21-6888f351349a> in <module>()
----> 1 group[4] = 'e'

TypeError: 'str' object does not support item assignment
```

List Concatenation

- addition on lists: concatenation

```
In [22]: fibs1 = [1, 1, 2, 3, 5]

In [23]: fibs2 = [8, 13, 21, 34, 55, 89]

In [24]: fibs1 + fibs2
Out[24]: [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 the end

List Slicing Examples

```
In [25]: grades
Out[25]: [85, 26, 77, 71, 85, 95]

In [26]: grades[2:5]
Out[26]: [77, 71, 85]

In [27]: grades[3:]
Out[27]: [71, 85, 95]

In [28]: grades[:4]
Out[28]: [85, 26, 77, 71]
```

List Slicing Examples - 2

- assign back to the same variable

```
In [29]: group
Out[29]: 'Monty Python'

In [30]: group = group[:4] + 'e' + group[5:]

In [31]: group
Out[31]: 'Monte Python'
```

Deleting Items

- removing an item from a list
- `del LIST_VAR[INDEX]`

```
In [32]: grades
Out[32]: [85, 26, 77, 71, 85, 95]

In [33]: del grades[3]

In [34]: grades
Out[34]: [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?

```
# 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

```
# 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

```
# 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

```
# 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)`