Foundations of Computing

Variables, Strings, Conditionals

Ekaterina Vylomova
Summer Term 2023

THE UNIVERSITY OF
MELBOURNE

January, 19th: On this day...(random facts)

- **1419** – Hundred Years' War: Rouen surrenders to Henry V of England, completing his reconquest of Normandy
- **1788** – The second group of ships of the First Fleet arrive at Botany Bay.
- **1883** – The first electric lighting system employing overhead wires, built by Thomas Edison, begins service at Roselle, New Jersey.
- **1915** – Georges Claude patents the neon discharge tube for use in advertising.
- **1986** – The first IBM PC computer virus is released into the wild. A boot sector virus dubbed (c)Brain, it was created by the Farooq Alvi Brothers in Lahore, Pakistan, reportedly to deter unauthorized copying of the software they had written.

Lecture Agenda

- Last lecture
  - Literals
  - Basic Data Types
  - Variables and Assignment
- This lecture
  - Variables and assignment (cont.)
  - String basics
  - String Manipulation
  - Conditionals

Reminders

- Forums and Grok help in full swing — make use of them if in need
- We do passively monitor your activities on Grok, and will occasionally preemptively reach out to students to offer help ... don't be weirded out by it!

Announcements

- 1. In-person Drop-In Sessions starting next week: Tue/Wed 11am–12pm, PAR-Peter Hall-G01 (just stay after the lecture!)
- 2. 6.15pm Tutorials will be in PAR-Elec. Engineering-121

Lecture Outline

1. Variables and Assignment
2. Data Type: Strings
3. Strings as Sequences
4. Data Type: Boolean

Literals and Variables

- Variables are ‘named objects’ that have references to corresponding memory cells that store literals
- objects (literals) are not copied inside variables
- variables are assigned the type of the resulting expression (right-hand side)
- N.B. “=” is the assignment operator and NOT used to test mathematical equality (we’ll get to that later ...)

Class Exercise

- Python is an “imperative” language, meaning that it has “program state” and the values of variables are changed only through (re-)assignment:

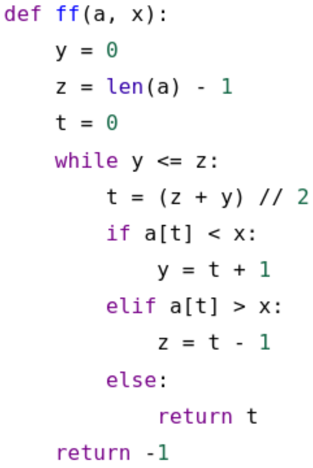
```
>>> a = 1
>>> b = 0.2
>>> a = a + 1
>>> b = b + a
>>> print(a)
>>> print(b)
```

What is the output of this code?

Variable Naming Conventions

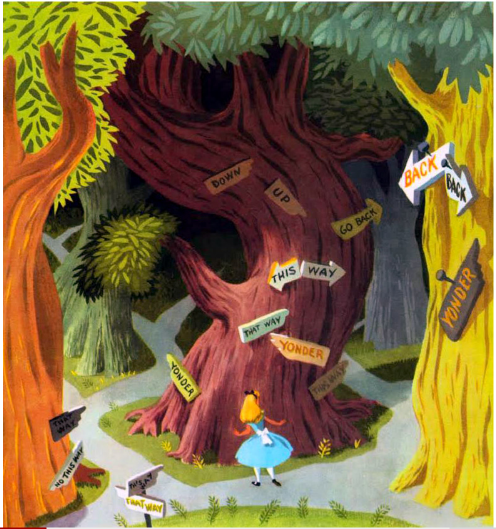
- Variable names must start with a character (`a-zA-Z`) or underscore (_), and consist of only alphanumeric (`0-9a-zA-Z`) characters and underscores (_)
- Casing is significant (i.e. `apple` and `Apple` are different variables)
- "Reserved words" (operators, literals and built-in functions) cannot be used for variable names (e.g. `in`, `print`, `not`, ...)
  - valid variable names: `a`, `dude123`, `_CamelCasing`
  - invalid variable names: `1`, `a-z`, `13CABS`, `in`

Variable Names: Compare



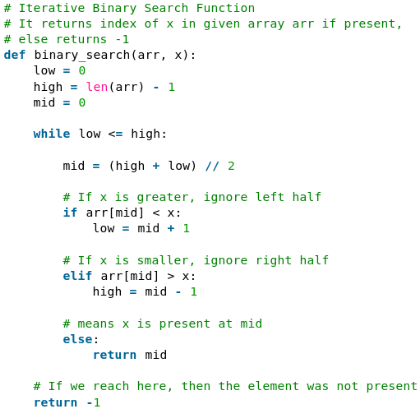
```python
def ff(a, x):
    y = 0
    z = len(a) - 1
    t = 0
    while y <= z:
        t = (z + y) // 2
        if a[t] < x:
            y = t + 1
        elif a[t] > x:
            z = t - 1
        else:
            return t
    return -1
```

Variable Names: Getting Lost



Variable Names: Giving Meaningful Names Increases Code Readability!

Page: https://www.geeksforgeeks.org/python-program-for-binary-search/



```python
# Iterative Binary Search Function
# It returns index of x in given array arr if present,
# else returns -1
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:

        mid = (high + low) // 2

        # If x is greater, ignore left half
        if arr[mid] < x:
            low = mid + 1

        # If x is smaller, ignore right half
        elif arr[mid] > x:
            high = mid - 1

        # means x is present at mid
        else:
            return mid

    # If we reach here, then the element was not present
    return -1
```

Class Exercise

- Calculate the *i*th Fibonacci number using only three variables

Lecture Outline

1. Variables and Assignment
2. Data Type: Strings
3. Strings as Sequences
4. Data Type: Boolean

A New Type: Strings

- A string (`str`) is a "chunk" of text, standardly enclosed within either single or double quotes:
  - `"Hello world"`
  - `'How much wood could a woodchuck chuck'`
- To include quotation marks (and slashes) in a string, "escape" them (prefix them with \):
  - `\"`, `\'` and `\\`
- Also special characters for formatting:
  - `\t` (tab), `\n` (newline)
- Use triple quotes (`'` or `"`) to avoid escaping/special characters:
  - `""""Ow," he said/yelled."""`

String Operators

- The main binary operators which can be applied to strings are:
  - + (concatenation)

```
>>> print("a" + "b")
ab
```

  - ∗ (repeat string *N* times)

```
>>> print('z' * 20)
zzzzzzzzzzzzzzzzzzzz
```

  - `in` (subset ... see next lecture for details)

```
>>> print('z' in 'zizzer zazzer zuzz')
True
```

Overloading

- But but but … didn't + and * mean different things for `int` and `float`?
  - Answer: yes; the operator is "overloaded" and functions differently depending on the type of the operands:

```
>>> print(1 + 1)
2
>>> print(1 + 1.0)
2.0
>>> print("a" + "b")
ab
>>> print(1 + 'a')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Functions Applicable to Strings

- Useful functions related to strings:
  - `len` (calculate the length of the string)

```
>>> print(len("a piece of string"))
17
```

  - `str` (convert an object to a string)

```
>>> str(2)
'2'
>>> str(2.0)
'2.0'
>>> str("string")
'string'
```

Class Exercise

- Given `num` containing an `int`, calculate the number of digits in it

Strings and Formatting I

- Often we want to insert variables into strings, optionally with some constraint on how they are formatted/presented
- We can do this in part through string concatenation (+), but it has its limitations:

```
>>> response = "yes"
>>> sentiment = 1/1
>>> print(response + ", " + response + ", " + \
... response + " ... I " + \
... str(100*sentiment) + "% agree")
yes, yes, yes ... I 100.0% agree
```

Strings and Formatting II

- A cleaner, more powerful way is with **format strings** ("f-strings"), marked with an "`f`" prefix at the start of the string:

```
>>> response = "yes"
>>> sentiment = 1/1
>>> print(f"{response}, {response}, {response}" + \
... " ... I {100 * sentiment:.0f}% agree")
yes, yes, yes ... I 100% agree
```

  - insert variables into strings with braces, possibly with some associated operators (e.g. 100 *)

Strings and Formatting III

- optionally add formatting specifiers with a colon (":"), e.g. to stipulate the number of decimal places to use for a float (e.g. "`.0f`" = zero decimal places)

Lecture Outline

1. Variables and Assignment
2. Data Type: Strings
3. Strings as Sequences
4. Data Type: Boolean

Sequences of Items

- One construct that pervades computing is a “sequence” (or “iterable” in Python-speak), i.e. the decomposition of an object into a well-defined ordering of items
  - text as sequences?
  - sounds as sequences?
  - images as sequences?
- Manipulation of objects tends to occur via “iteration” over iterables

String Manipulation

- As well as "assembling" strings via + and $*$, we are able to pull strings apart in the following ways:
  - "indexing" — return the single character at a particular location
  - "slicing" — extract a substring of arbitrary length
  - "splitting" — break up a string into components based on particular substrings

String Manipulation: Indexing

- Each character in a string can be accessed via "indexing" relative to its position from the left of the string (zero-offset) or the right of the string ([minus] one-offset):

I	t		w	a	s		a		d	a	r	k
0 −13	1 −12	2 −11	3 −10	4 −9	5 −8	6 −7	7 −6	8 −5	9 −4	10 −3	11 −2	12 −1

```
>>> story[-8]
's'
>>> story[5]
's'
```

String Manipulation: Slicing I

- It is possible to “slice” a string by specifying a `START` and (non-inclusive) `END` `int` value:

  ```
  >>> story[1:11]
  't was a da'
  ```

  N.B. the sliced substring length = `END` − `START`

- By default, `START=0` and `END` is the length of the string:

  ```
  >>> story[:-7]
  'It was'
  ```

String Manipulation: Slicing II

- It is also possible to specify slice “direction” (1 or -1):

```
>>> story[-1:-7:-1]
'krad a'
```

Here, the first argument is still the START and the second is still the END, but the default values are START=-1 and END = -(the length of the string + 1):

```
>>> s[-8::-1]
'saw tI'
>>> s[:-5:-1]
'krad'
```

More on String Manipulation

Lecture Outline

1. Variables and Assignment
2. Data Type: Strings
3. Strings as Sequences
4. Data Type: Boolean

In Search of the Truth ...

- Often, we want to check whether a particular value satisfies some condition:
  - does it have four legs?
  - is it over 18?
  - is it tall, with rabbit ears, a grey back, whiskers, a creme stomach with grey markings on it, and (at times) an umbrella?

Source(s): `http://fav.me/d4qp4si`

In Search of the Truth ...

- Often, we want to check whether a particular value satisfies some condition:
  - does it have four legs?
  - is it over 18?
  - is it tall, with rabbit ears, a grey back, whiskers, a creme stomach with grey markings on it, and (at times) an umbrella?

Source(s): `http://fav.me/d4qp4si`

In Search of the Truth ...

- For this, we require:
  - a way of describing whether the test is satisfied or not
  - a series of comparison operators
  - a series of logic operators for combining comparisons
  - a way of conditioning behaviour on the result of a given test

Capturing Truth: The bool Type

- We capture truth via the `bool` (short for "Boolean") type, which takes the two values: `True` and `False`
- As with other types, we can "convert" to a `bool` via the `bool()` function:

```
>>> bool(3)
True
>>> bool(0)
False
>>> bool("banana")
True
```

Every type has a unique value for which `bool()` evaluates to `False`

Evaluating Truth: Comparison

- We evaluate truth via the following Boolean comparison operators:

`==`	equality; NOT the same as $=$
`>`, `>=`	greater than (or equal to)
`<`, `<=`	less than (or equal to)
`!=`	not equal to
`in`	is an element of

```
>>> 2 == 3
False
>>> 'a' <= 'apple'
True
>>> 'bomp' in 'bomp, bomp, bomp'
True
```

Combining Truth

- We combine comparison operators with the following logic operators:
    - `and`, `or`, `not`:

`and`	`True`	`False`
`True`	`True`	`False`
`False`	`False`	`False`

`or`	`True`	`False`
`True`	`True`	`True`
`False`	`True`	`False`

`not`	`True`	`False`
	`False`	`True`

- NB: precedence: `not` > `and` > `or`

Lecture Summary

- What is a sequence/iterable?
- Strings: how are they formatted, and what operations/functions can be applied to them?
- Strings: what are indexing, slicing and splitting?
- What is the `bool` type?
- What Boolean comparison operators are commonly used in Python?