

Business Intelligence-relaterade programspråk

Mikael Olsson

Webbutvecklare

- Driver företaget Emmio
- Jobbat med webbutveckling sedan 1999.

Kursplan

Innehåll och syfte

- Syftet med kursen är att den studerande ska få testa på att arbeta med ett av de mest förekommande programspråken som används inom Business Intelligence, Python. Målet är att de studerande ska kunna hantera något programspråk som används för BI-relaterade system. De studerande ska även förstå skillnaden mellan olika typer av programspråk, t.ex. scriptspråk och rena objektorienterade språk.

Kursplan

Mål

- Kursen ger kunskaper i/om...
 - Tillfällen och användningsområden för olika programspråk inom Business Intelligence
 - Skillnaden mellan olika typer av programspråk, t.ex. scriptspråk och objektorienterade språk
 - Programmering i BI-relaterade system
- Kursen ger färdigheter i att...
 - Utföra enklare programmering inom ett på marknaden aktuellt programspråk som används för BI-relaterade system
- Kursen ger kompetens för att...
 - För en verksamhet kunna utveckla en funktion i en BI-lösning

Kursplan

Examinationsform

- Gruppuppgift samt individuella inlämningsuppgifter

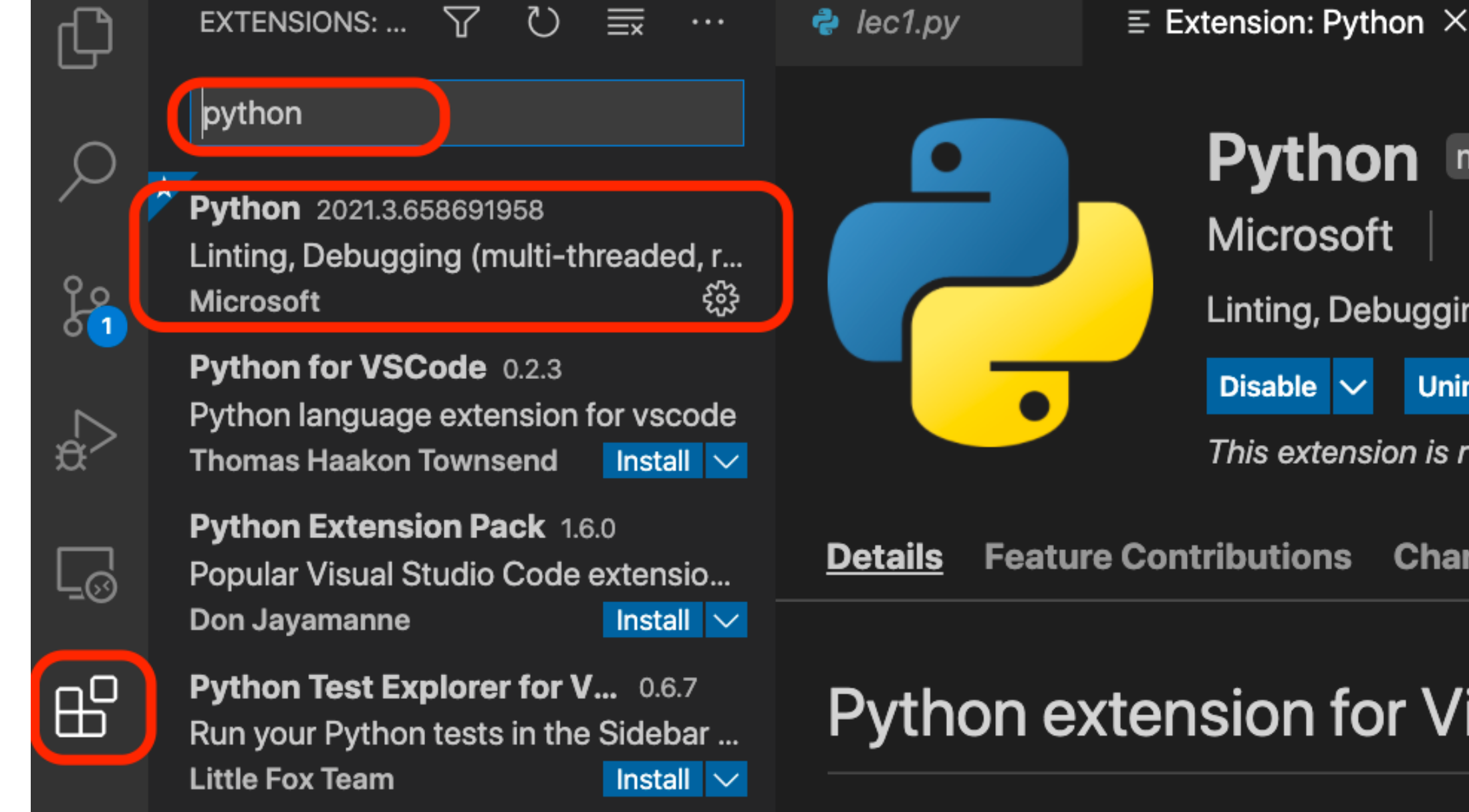
Kursplan

Betygskriterier

- **Icke Godkänt (IG):** om den studerande har genomfört kursen utan att nå alla kursens läranderesultat.
- För **Godkänt (G)** krävs att den studerande har genomfört kursen och nått nedanstående läranderesultat. Den studerande ska kunna:
 - För en verksamhet kunna utveckla en funktion i en BI-lösning
 - Med hjälp av Python transformera/modellera data
- För **Väl Godkänd (VG)** krävs att studenten nått alla läranderesultat för godkänt samt nedanstående VG-kriterier. Den studerande ska kunna:
 - Med säkerhet bedöma vilket programspråk är optimalt för olika BI-lösningar
 - Med säkerhet skapa optimala lösningar i Python som används för BI-relaterade system

Installera program

- Språktolk - Python
 - <https://www.python.org/downloads/>
 - Mac är lite speciellt...
 - <https://opensource.com/article/19/5/python-3-default-mac>
- IDE - Integrated Development Environment
 - Visual Studio Code (icke att förväxla med Visual Studio)
 - <https://code.visualstudio.com/docs/languages/python>
 - <https://marketplace.visualstudio.com/items?itemName=ms-python.python>



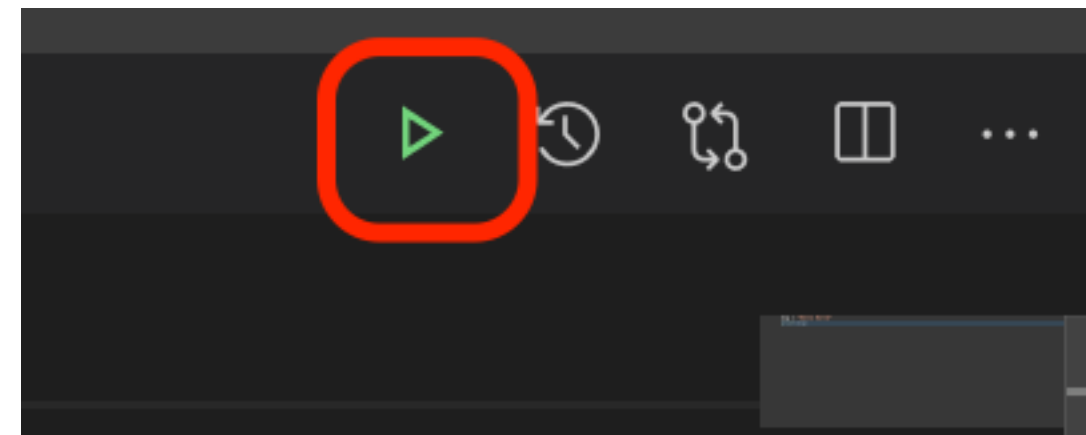
Ett första exempel

Hello world

- Skapa filen `hello.py`
- Skriv in följande kod:

```
msg = "Hello World"  
print(msg)
```

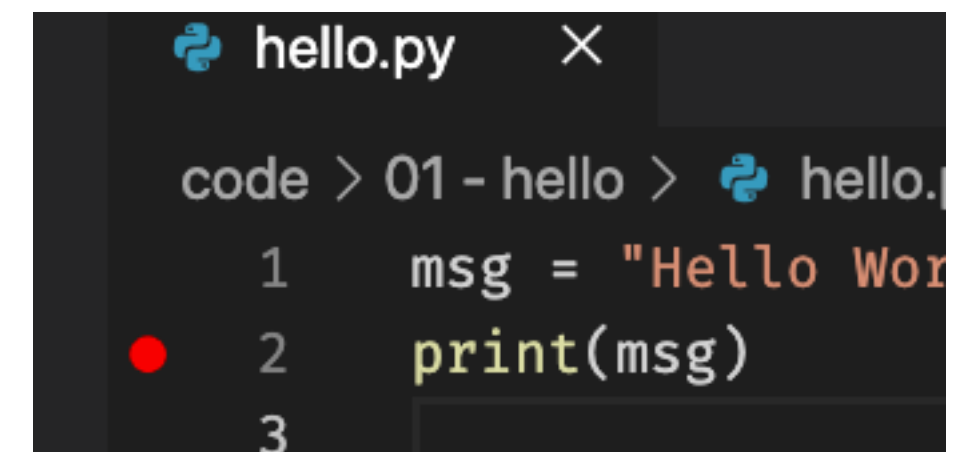
- Tryck på Run-knappen



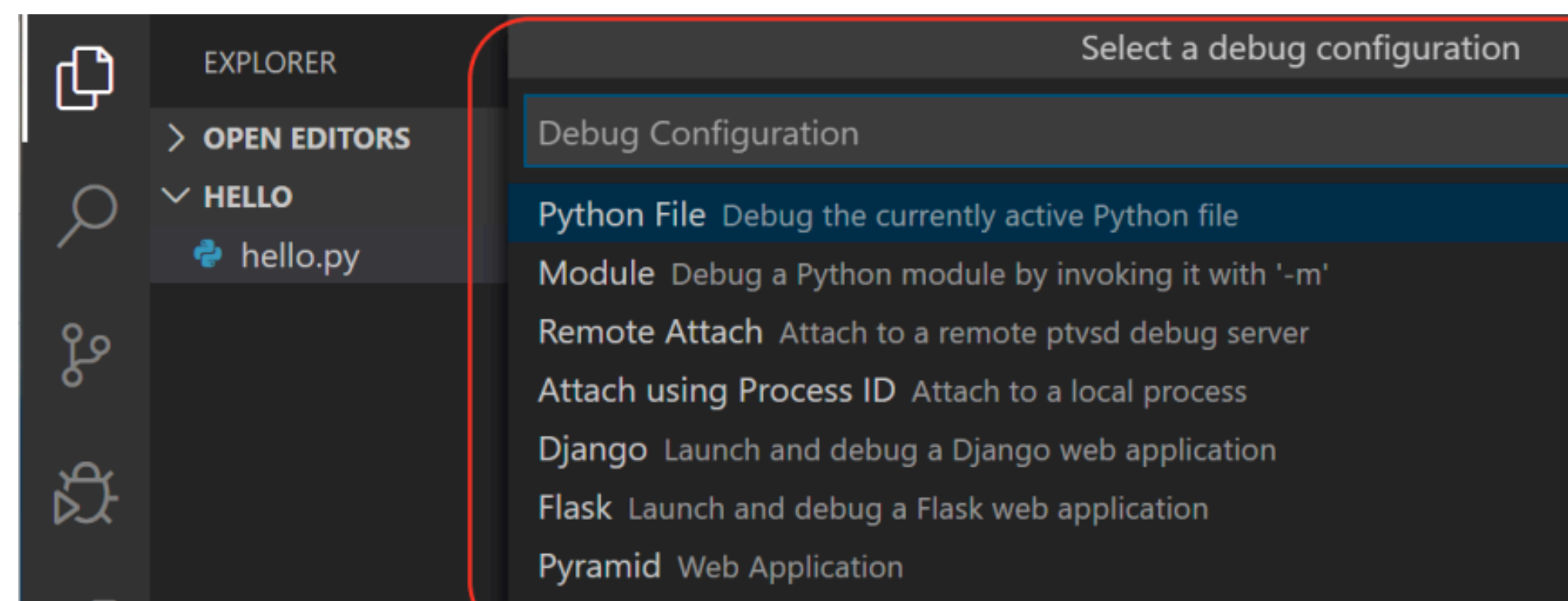
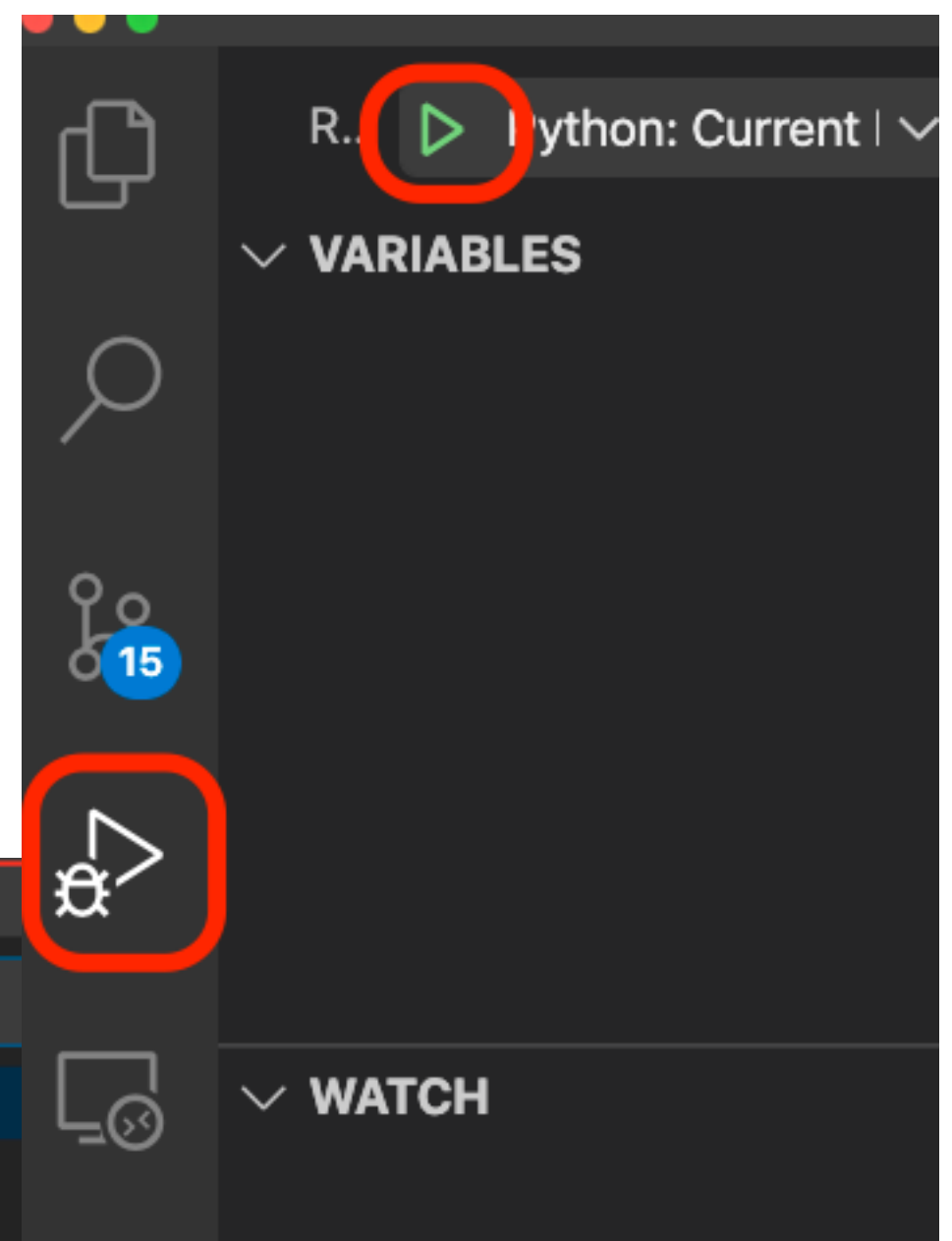
```
→ bi20python git:(main) x /usr/...  
ade_programspråk/bi20python/code/  
Hello World  
→ bi20python git:(main) x
```


Debugger

- Ibland kan man vilja se vad som händer när ett program körs.
- Klicka ganska långt till vänster på rad två för att skapa en *break point*.
- Byt till Debug-fliken och klicka på Debug-knappen.
- Första gången får du en fråga om vilken sorts debugging du vill göra. Det finns olika options, men just nu kan vi välja *Python File*.

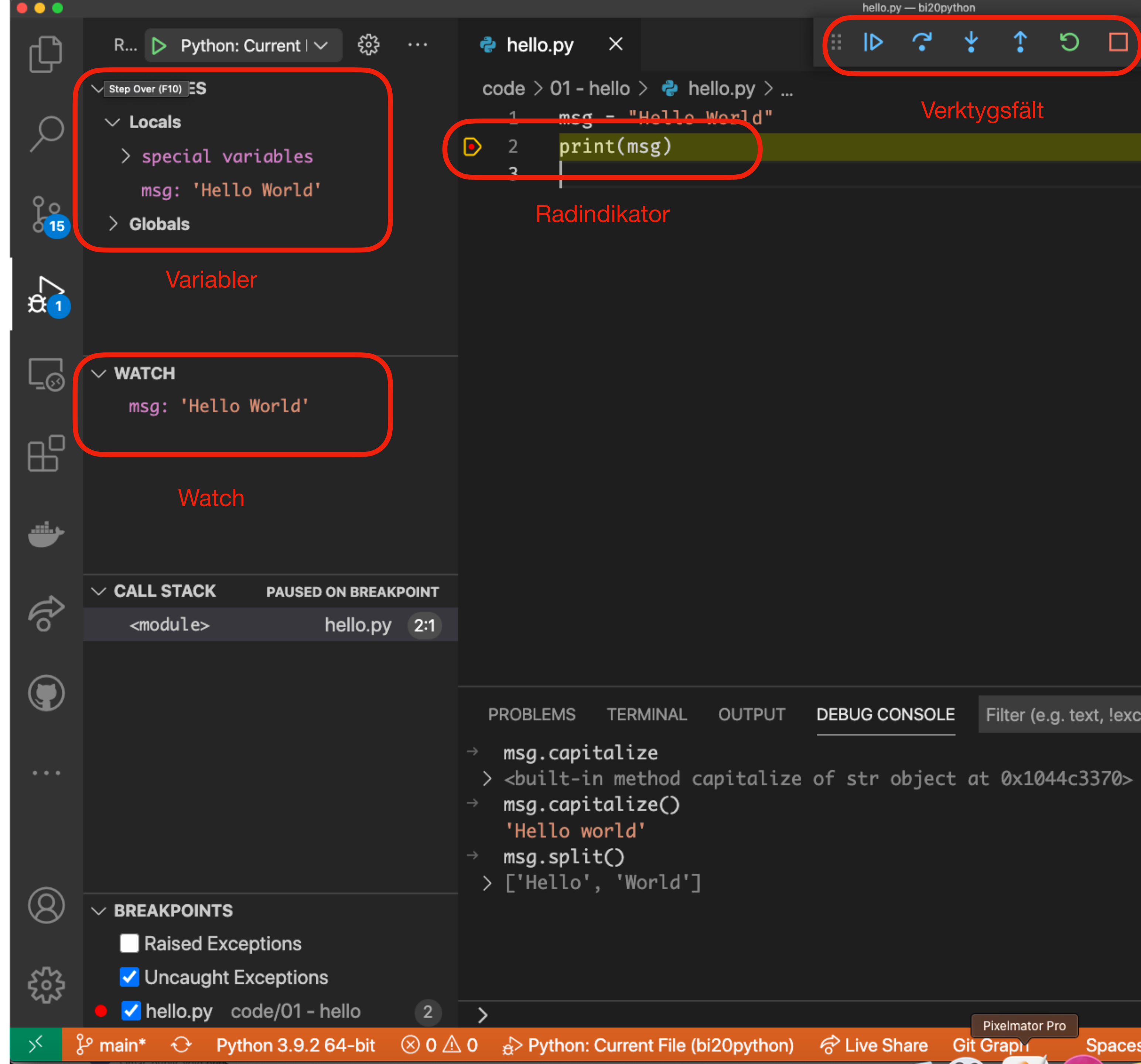


```
hello.py
code > 01 - hello > hello.py
1 msg = "Hello World"
2 print(msg)
3
```



Debugger

- Vi vet inte vad allt är här än, men vi bör få upp något sånt här.
- Verkttygsfältet innehåller knappar med följande funktioner, från vänster till höger:
 - continue, step over, step into, step out, restart, stop
- Vi kommer att prata mer om dessa senare.



VS Code

- Om man vill utforska grunderna i VS Code kan man kolla in följande artikel vid tillfälle.
 - <https://code.visualstudio.com/docs/python/python-tutorial>

Datatyper

- Datatyper är olika typer av innehåll, t ex text, tal osv.
- I python finns följande inbyggda datatyper

Text Type: `str`

Numeric Types: `int`, `float`, `complex`

Sequence Types: `list`, `tuple`, `range`

Mapping Type: `dict`

Set Types: `set`, `frozenset`

Boolean Type: `bool`

Binary Types: `bytes`, `bytearray`, `memoryview`

Variabler

- Variabler låter oss spara värden till senare. `msg` i vårt första program är ett exempel.
- Olika språk hanterar variabler olika.
 - I python behöver man inte *deklarera* variabler. Det betyder att variabeln bara uppstår när den används första gången.
 - Python är ett dynamiskt typat språk. Det innebär att variabler kan innehålla olika datatyper vid olika tillfällen.

```
x = 4          # x is of type int
x = "Sally"    # x is now of type str
```

Läsa in data från tangentbordet

- Vi kan läsa in data på olika sätt, t ex med kommandot `input`.

```
value = input("Please enter a string:\n")  
print(f'You entered {value}')
```

A formatted string literal or f-string is a string literal that is prefixed with 'f' or 'F'. These strings may contain replacement fields, which are expressions delimited by curly braces {}. While other string literals always have a constant value, formatted strings are really expressions evaluated at run time.

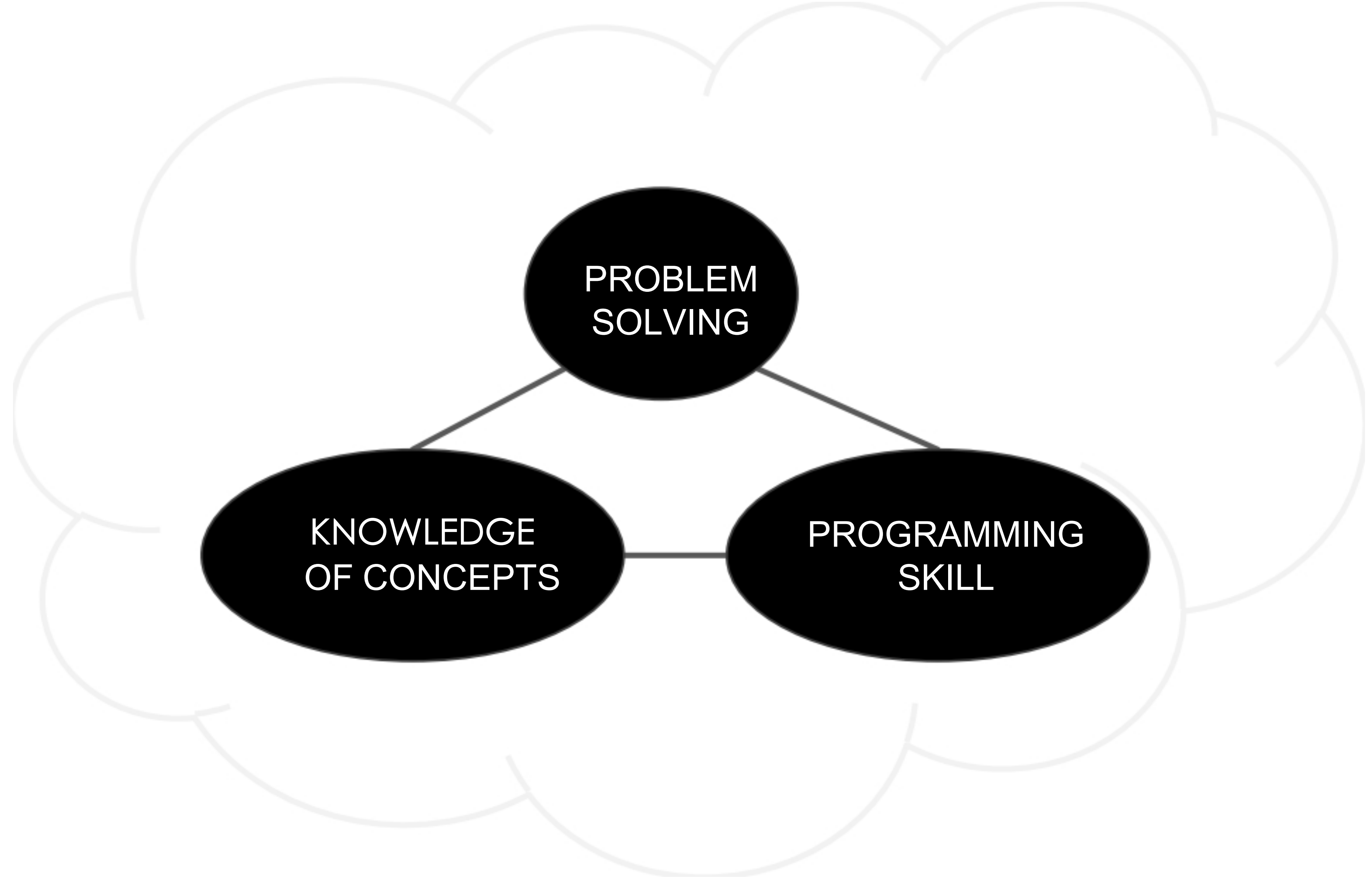
- https://docs.python.org/3.6/reference/lexical_analysis.html#formatted-string-literals

TODAY

- course info
- what is computation
- python basics
- mathematical operations
- python variables and types

FAST PACED COURSE

- New to programming? **PRACTICE. PRACTICE? PRACTICE!**
 - can't passively absorb programming as a skill
 - follow along coding
 - don't be afraid to try out Python commands!



TOPICS

- represent knowledge with **data structures**
- **iteration and recursion** as computational metaphors
- **abstraction** of procedures and data types
- **organize and modularize** systems using object classes and methods
- different classes of **algorithms** , searching and sorting
- **complexity** of algorithms

WHAT DOES A COMPUTER DO

- Fundamentally:
 - performs **calculations**
a billion calculations per second!
 - **remembers** results
100s of gigabytes of storage!
- What kinds of calculations?
 - **built-in** to the language
 - ones that **you define** as the programmer
- computers only know what you tell them

TYPES OF KNOWLEDGE

- **declarative knowledge** is **statements of fact** .
- **imperative knowledge** is a **recipe** or "how-to" .

A NUMERICAL EXAMPLE

- square root of a number x is y such that $y * y = x$
- recipe for deducing square root of a number x (16)
 - 1) Start with a **guess** , g
 - 2) If $g * g$ is **close enough** to x , stop and say g is the answer
 - 3) Otherwise make a **new guess** by averaging g and x/g
 - 4) Using the new guess, **repeat** process until close enough

g	$g * g$	x / g	$(g + x / g) / 2$
3	9	$16 / 3$	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

WHAT IS A RECIPE

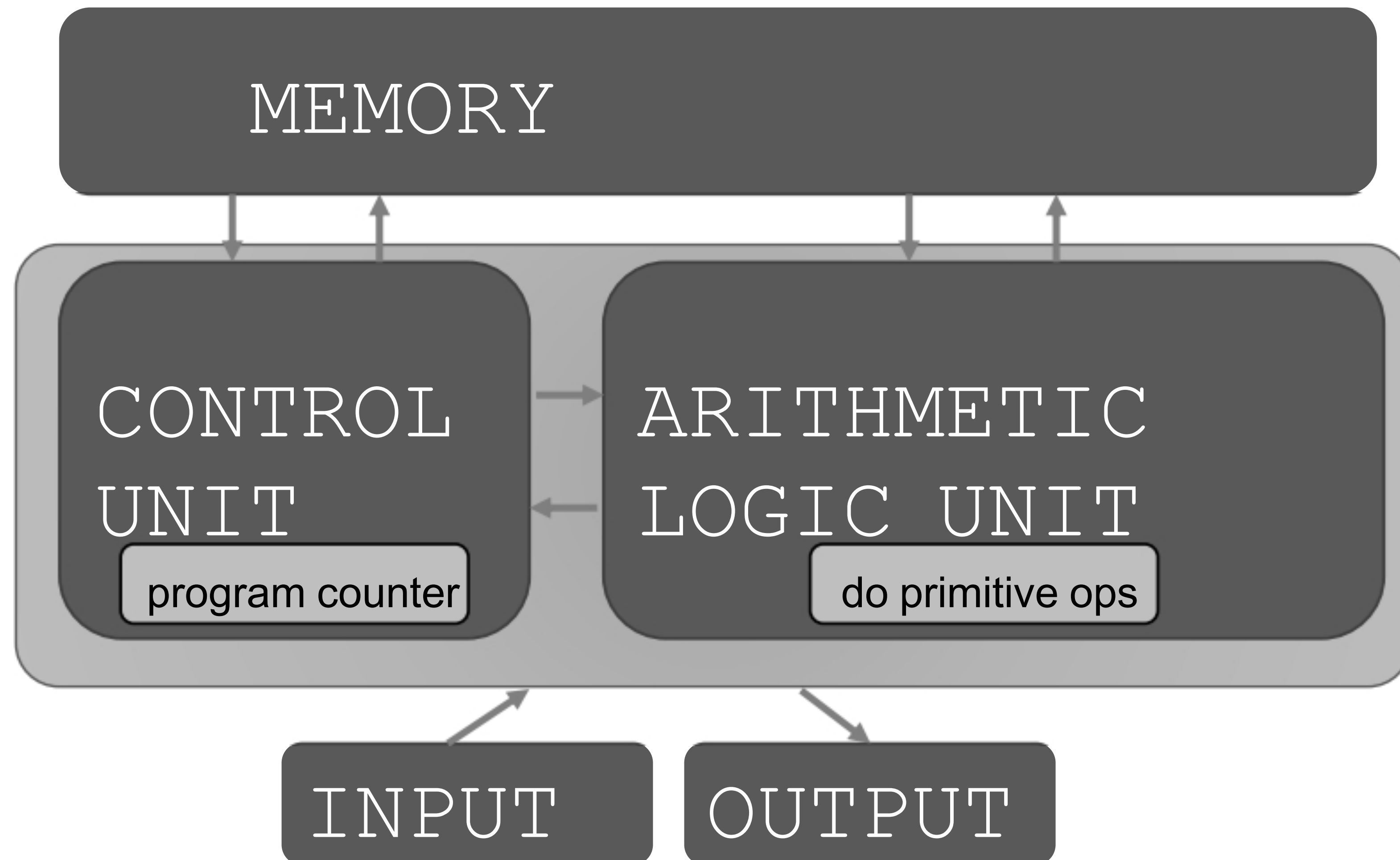
- 1) sequence of simple **steps**
- 2) **flow of control** process that specifies when each step is executed
- 3) a means of determining **when to stop**

1+2+3 = an **algorithm** !

COMPUTERS ARE MACHINES

- how to capture a recipe in a mechanical process
- **fixed program** computer
 - calculator
- **stored program** computer
 - machine stores and executes instructions

BASIC MACHINE ARCHITECTURE



STORED PROGRAM COMPUTER

- sequence of **instructions stored** inside computer
 - built from predefined set of primitive instructions
 - 1) arithmetic and logic
 - 2) simple tests
 - 3) moving data
- special program (interpreter) **executes each instruction in order**
 - use tests to change flow of control through sequence
 - stop when done

BASIC PRIMITIVES

- Turing showed that you can **compute anything** using 6 primitives
- modern programming languages have more convenient set of primitives
- can abstract methods to **create new primitives**
- anything computable in one language is computable in any other programming language

CREATING RECIPES

- a programming language provides a set of primitive **operations**
- **expressions** are complex but legal combinations of primitives in a programming language
- expressions and computations have **values** and meanings in a programming language

Boolean

- Vi kommer att titta mer strukturerat på olika flödeskontroller, men för att kunna komma igång och öva lite ska vi börja lite ostrukturerat.
- Ett booleskt värde är sant (True) eller falskt (False).

```
print(1 < 5) # Vad ger utskriften?  
print(10 < 5)
```

```
# #####
```

```
i = 5  
print(i < 5) # Vad ger utskriften?
```

While

- En while-loop låter oss upprepa kod så länge något är sant.

```
i = 1
while i < 6:
    print(i)
    i += 1
```

- Det här avståndet från vänsterkanten kallas *indentering*.
- I python är indentering viktig, den berättar för python hur kod hänger ihop.
 - I det här fallet betyder det att båda de två sista raderna ska utföras så länge villkoret är sant.

Slumptal

```
# generate random integer values  
from random import randint
```

```
# generate a random number from (and including) 0 to (and including) 10  
value = randint(0, 10)  
print(value)
```

Grupper

- Ta några minuter och skriv ner följande egenskaper om er själva i en fil, på en lapp eller liknande:
 - Kön
 - Åldersgrupp (<25 resp >25)
 - Erfarenhet av programmering 1-5 (där 1 är ingen erfarenhet och 5 är mycket)
 - Ser sig själv som mer introvert - ser sig själv som mer extrovert

Grupper

- Med hjälp av era kort vill jag att ni ska dela in er i grupper.
- Ni får en poäng för varje olika svar ni har.
 - Har ni t ex minst en man och minst en kvinna i er grupp får man två poäng, en för varje variant, s a s.
- Organisera grupper om 3-4 personer med så jämna poängantal som möjligt.

ASPECTS OF LANGUAGES

- **syntax**

- English: "cat dog boy" → not syntactically valid
"cat hugs boy" → syntactically valid
- programming language: "hi"5 → not syntactically valid
3.2*5 → syntactically valid

ASPECTS OF LANGUAGES

- **static semantics** is which syntactically valid strings have meaning
 - English: "I are hungry" → syntactically valid
but static semantic error
 - programming language: $3.2 * 5$ → syntactically valid
 $3 + "hi"$ → static semantic error

ASPECTS OF LANGUAGES

- **semantics** is the meaning associated with a syntactically correct string of symbols with no static semantic errors
 - English: can have many meanings "Flying planes can be dangerous"
 - programming languages: have only one meaning but may not be what programmer intended

WHERE THINGS GO WRONG

- **syntactic errors**
 - common and easily caught
- **static semantic errors**
 - some languages check for these before running program
 - can cause unpredictable behavior
- no semantic errors but **different meaning than what programmer intended**
 - program crashes, stops running
 - program runs forever
 - program gives an answer but different than expected

PYTHON PROGRAMS

- a **program** is a sequence of definitions and commands
 - definitions evaluated
 - commands executed by Python interpreter in a shell
- **commands** (statements) instruct interpreter to do something
- can be typed directly in a shell or stored in a file that is read into the shell and evaluated
 - Problem Set 0 will introduce you to these in Anaconda

OBJECTS

- programs manipulate **data objects**
- objects have a **type** that defines the kinds of things programs can do to them
 - Ana is a human so she can walk, speak English, etc.
 - Chewbacca is a wookiee so he can walk, “ mwaaarhrhh ”, etc.
- objects are
 - scalar (cannot be subdivided)
 - non-scalar (have internal structure that can be accessed)

SCALAR OBJECTS

- `int` – represent **integers** , ex. 5
- `float` – represent **real numbers** , ex. 3.27
- `bool` – represent **Boolean** values `True` and `False`
- `NoneType` – **special** and has one value, `None`
- can use `type()` to see the type of an object

```
>>> type(5)
```

```
int
```

```
>>> type(3.0)
```

```
float
```

*what you write into
the Python shell*

*what shows after
hitting enter*

TYPE CONVERSIONS (CAST)

- can **convert object of one type to another**
- `float(3)` converts integer 3 to float 3.0
- `int(3.9)` truncates float 3.9 to integer 3

EXPRESSIONS

- **combine objects and operators** to form expressions
- an expression has a **value** , which has a type
- syntax for a simple expression
`<object> <operator> <object>`

OPERATORS ON ints and floats

- $i + j$ → the **sum**
 - $i - j$ → the **difference**
 - $i * j$ → the **product**
 - i / j → **division**
- if both are ints, result is int
if either or both are floats, result is float
- result is float
-
- $i \% j$ → the **remainder** when i is divided by j
 - $i ** j$ → i to the **power** of j

SIMPLE OPERATIONS

- parentheses used to tell Python to do these operations first
- **operator precedence** without parentheses
 - **
 - *
 - /
 - + and – executed left to right, as appear in expression

BINDING VARIABLES AND VALUES

- equal sign is an **assignment** of a value to a variable name

variable

value

```
pi = 3.14159
```

```
pi_approx = 22/7
```

- value stored in computer memory
- an assignment binds name to value
- retrieve value associated with name or variable by invoking the name, by typing `pi`

ABSTRACTING EXPRESSIONS

- why **give names** to values of expressions?
- to **reuse names** instead of values
- easier to change code later

```
pi = 3.14159  
radius = 2.2  
area = pi*(radius**2)
```

PROGRAMMING vs MATH

- in programming, you do not “solve for x”

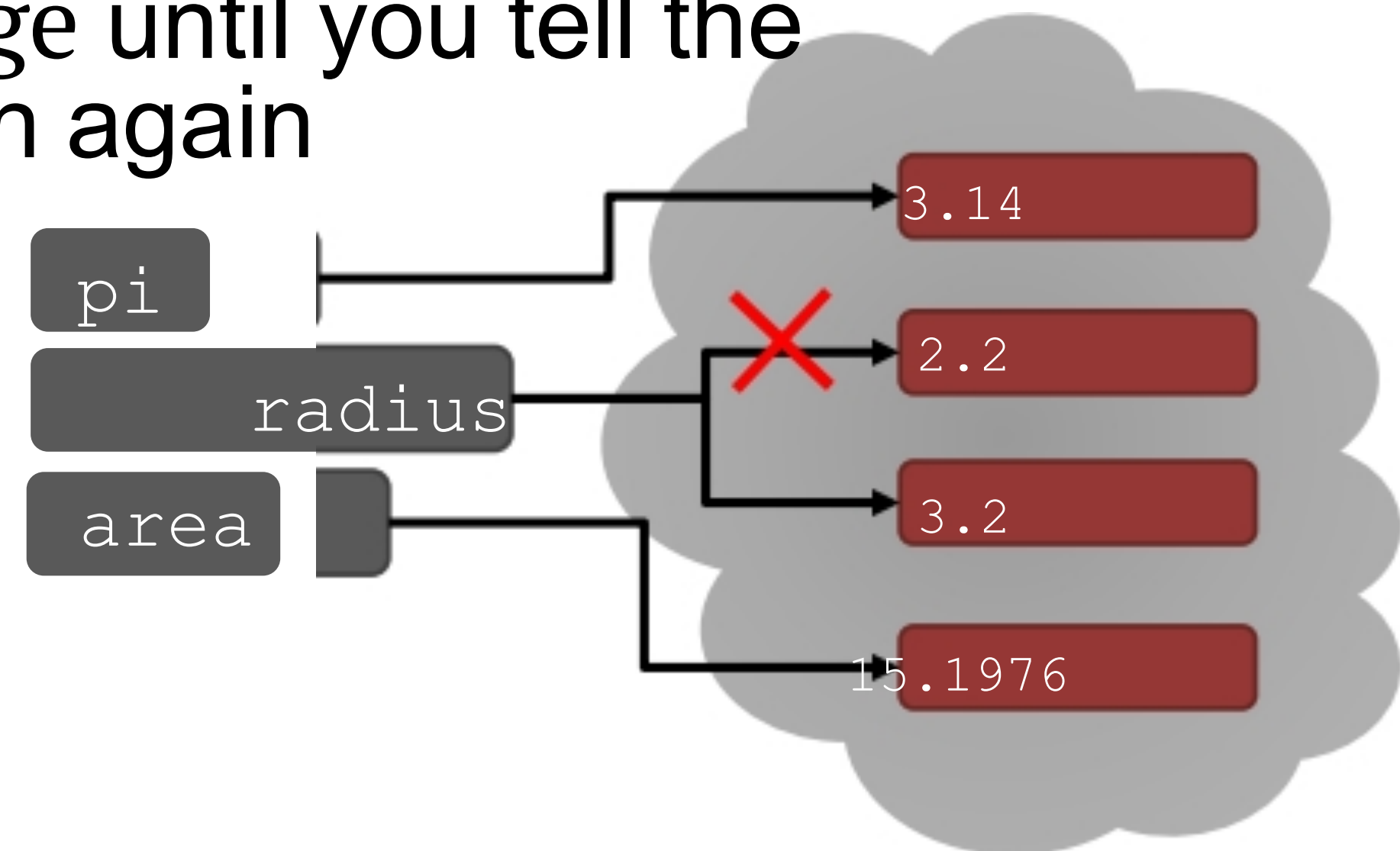
```
pi = 3.14159
radius = 2.2
# area of circle
area = pi*(radius**2)
radius = radius+1
```

an assignment
* expression on the right, evaluated to a value
* variable name on the left
* equivalent expression to `radius = radius + 1`
is `radius += 1`

CHANGING BINDINGS

- can **re-bind** variable names using new assignment statements
- previous value may still stored in memory but lost the handle for it
- value for area does not change until you tell the computer to do the calculation again

```
pi = 3.14  
radius = 2.2  
area = pi*(radius**2)  
radius = radius+1
```



Uppgifter

Samarbete

- Du uppmuntras att jobba tillsammans med andra studenter. Varje student ska dock lämna in sin uppgift separat. Skriv en kommentar om vilka du har jobbat med.

Uppgift 1

Raising a number to a power and taking a logarithm

- The goal of this programming exercise is to make sure your python installation is correct, to get you more comfortable with using VS Code, and to begin using simple elements of Python.
- Standard elements of a program include the ability to print out results (using the print operation), the ability to read input from a user at the console (for example using the input function), and the ability to store values in a variable, so that the program can access that value as needed.

Uppgift 1

Raising a number to a power and taking a logarithm

- Write a program that does the following in order:
 1. Asks the user to enter a number “x”
 2. Asks the user to enter a number “y”
 3. Prints out number “x”, raised to the power “y”.
 4. Prints out the log (base 2) of “x”.

Uppgift 1

Raising a number to a power and taking a logarithm

- An example of an interaction with your program is shown below. The words printed in blue are ones the computer should print, based on your commands, while the words in black are an example of a user's input. The colors are simply here to help you distinguish the two components.
- Enter number x: 2
Enter number y: 3
 $X^{**y} = 8$
 $\log(x) = 1$

Uppgift 1

Raising a number to a power and taking a logarithm

- Hints:
 - To see how to use the print command, you may find it convenient to look at the input and output of the Python Wikibook. This will show you how to use print statements to print out values of variables.
 - To see how to read input from a user's console into the Python environment, you may find it convenient to look at the same section (see for example the input() function)
 - Reference the basic math section of the Python Wikibook to read more about using basic mathematical operators in Python
 - Remember that if you want to hold onto a value, you need to store it in a variable (i.e., give it a name to which you can refer when you want that value). You may find it convenient to look at the variables and strings section of the Python Wikibook. (As you read through, remember that in Python 3.x you should be using input() not raw_input()). Take a look at the “Combining Numbers and Strings” sub-section, because you will be working with numbers and strings in this problem and will have to convert between the two using the str() and int() functions.